Entwicklung eines rationalen Entscheidungsprozesses für Architekturentscheidungen

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von Diplom-Wirtschaftsinformatiker Sven Wohlfarth
geboren am 14. Februar 1980 in Ilmenau

vorgelegt am 9. April 008

wissenschaftliche Aussprache am 24. September 2008

Gutachter:

- Priv.-Doz. Dr.-Ing. habil. Matthias Riebisch
- Univ.-Prof. Dr.-Ing. habil. Wolfgang Fengler
- a. Univ.-Prof. Dr. Johannes Sametinger

urn:nbn:de:gbv:ilm1-2008000177

Kurzfassung

In Softwareentwicklungsprozessen müssen permanent die *richtigen* Design- und Architekturentscheidungen getroffen werden, damit die mit dem Entwicklungs- oder Reengineeringprojekt verbundenen Ziele in vollem Umfang erfüllt werden können. Diese Entscheidungen können dabei von unterschiedlicher Natur sein. So werden einerseits Entscheidungen getroffen, die nur geringe Auswirkungen auf das Softwaresystem haben. Auf der anderen Seite existieren Entscheidungen mit strategischem Charakter, die sich auf große Teile der Architektur und auf zentrale Systemeigenschaften auswirken. Gerade die strategischen Architekturentscheidungen sind in Großprojekten mit 50 oder mehr Entwicklern von hoher kombinatorischer Komplexität und beinhalten große Unsicherheiten über versteckte Abhängigkeiten. Der Entscheidungsträger, meist der Architekt oder der Projektleiter, ist mit einer Vielzahl unterschiedlicher Faktoren und Bedingungen konfrontiert. Hierzu zählen konkurrierende Ziele oder alternative Lösungsansätze, für die meist nur unvollständige Informationen vorliegen. Unter diesen Voraussetzungen führen unsystematische Entscheidungen zu unkalkulierbaren Risiken mit gravierenden Folgen für das Softwaresystem und das Entwicklungsprojekt, wie z. B. eine deutliche Erhöhung der Entwicklungskosten oder zeitliche Verzögerungen.

Die bereits existierenden Methoden zur Entscheidungsunterstützung berücksichtigen die spezifischen Eigenschaften von Softwarearchitekturen zu wenig. Sie sind zu feingranular, codeorientiert und benötigen Informationen in einer formalen Genauigkeit und Vollständigkeit, die bei Architekturentscheidungen in Großprojekten aus Aufwandsgründen nicht erhoben werden können. Somit fehlt eine Unterstützung des Entscheidungsträgers, um die Vielzahl an Einzelinformationen und subjektiven Einschätzungen zu strukturieren sowie die Entscheidungsfindung systematisch und fokussiert durchzuführen.

Mit der vorliegenden Dissertation wird das Ziel verfolgt, die Komplexität, Unsicherheiten und Risiken bei Architekturentscheidungen zu reduzieren, um aufwandsintensive Korrekturen zu vermeiden und die Architekturziele in vollem Umfang zu erfüllen. Auf der Grundlage des in der Entscheidungstheorie beschriebenen generischen Vorgehens zur Entscheidungsfindung wird ein Vier-Phasen-Entscheidungsprozess entwickelt. Dieser Prozess beinhaltet Methoden und Konzepte, um ausgehend von den Zielen, Rahmenbedingungen und der existierenden Architektur systematisch alternative Lösungsansätze zu entwickeln. Im Anschluss werden die Lösungsansätze nach rationalen Gesichtspunkten im Hinblick auf die Zielerreichung bewertet, um eine ausgewogene Entscheidung zu treffen. Der entwickelte Entscheidungsprozess berücksichtigt dabei die speziellen Eigenschaften von Softwarearchitekturen:

- Trotz unvollständiger Informationen und Unsicherheiten können versteckte Abhängigkeiten mit einem szenariobasierten Analyse- und Bewertungsansatz, auf der Grundlage der Architecture-Level-Modifiability-Analysis (ALMA), sichtbar gemacht werden.
- Die systematische Aufteilung komplexer Entscheidungen in handhabbare Einzelentscheidungen wird durch die Anwendung eines gestuften Verfahrens mit Grob- und Feinplanung erreicht.
- Um ein ökonomisch sinnvolles Verhältnis zwischen dem Aufwand zur Entscheidungsfindung und dem Nutzen in Form von reduzierten Risiken, Unsicherheiten und einer geringeren Komplexität zu ermöglichen, kann die Detailtiefe der Analysen anhand eindeutiger Kriterien flexibel angepasst werden.

Zwei praktische prototypische Anwendungen des Entscheidungsprozesses zeigen auf, wie eine Architekturentscheidung systematisch und nach rationalen Gesichtspunkten durchgeführt werden kann. Die während der Entscheidungsfindung getroffenen Annahmen und Erwartungen werden im Anschluss mit den Ergebnissen der realen Implementierung verglichen. Anhand des Vergleichs wird klar erkennbar, welche versteckten Abhängigkeiten durch den Einsatz des Entscheidungsprozesses bereits frühzeitig erkannt wurden sowie welche Vorteile die *richtige* Entscheidungsfindung für das Softwaresystem und das Entwicklungsprojekt hat.

Abstract

It is one of the critical tasks to make the *right* design- and architectural-decisions in huge and complex developing or reengineering projects. Such decisions have different types. On the one hand there are decisions with minimal effects on the architecture and the software system. On the other hand there are more strategic decisions which effect the architecture widely and change the central characteristics of the software system. Particularly the strategic decisions are very complex, risky and include many uncertain facts about hidden dependencies. The complexity and risks rise if such decisions have to be made in huge projects with 50 or more developers. The decisionmaker, mostly the project manager or the client, is confronted with various factors, assumptions and constraints. Typical examples are competing objectives, alternative solutions and incomplete information about external third-party systems. If such complex decisions have to be made in an unsystematic way, they will lead to uncalculatable risks with enormous bad consequences for the software system and the development project. Examples are changed or missed deadlines, risen development costs or monetary losses due to an outage of a business critical system.

However, the specific characteristics of architectural decisions are not considered by existing methods and concepts to support decision making. They are too detailed, focussed on source code and require information in a formal quality and completeness. These information can not be gathered within such huge projects because of the high effort, time pressure and lacking resources. Therefore an architectural decision process is missing to structure the various information, assumptions and subjective estimations and so you can make such complex and risky decisions in a systematic and focussed way.

The main objective of the following dissertation is to reduce the complexity, uncertainty and risks of architectural decisions in order to avoid additional changes and adjustments as well as to achieve the desired objectives. An architectural decision process with four phases is developed on the basis of the generic proceeding of the decision theory. This process includes methods and concepts in order to establish alternative solutions on the basis of the objectives, conditions and the model of the existing architecture. The various alternative solutions are evaluated through a systematic proceeding in order to identify and select the best solution. The developed process includes the specific characteristics of software architectures:

- Besides incomplete information and uncertainties, it is possible to observe hidden dependencies through scenario-based analysis methods, establishted by the concepts of the Architecture-Level-Modifiability-Analysis (ALMA).
- Due to the complexity and risks, huge architectural changes have to be separated into smaller tasks. This is supported by a stepped planning, from abtract analysis to more detailed planning.
- To achieve a reasonable relation between the analysis effort and the benefis from the analysis in terms of reduced risks, complexity and uncertainty, the depths of the analysis can be adjusted flexibly by clear objectives.

Two practical applications show, how to make architectural decisions in a systematic way by using the decision process. Afterwards, the assumptions and expectations, which have been used for the decision making, are evaluated by comparing with the consequences of the real implementation. Due to the results of the comparison it can be described clearly, which advantages and disadvantages the application of the decision process has.

Danksagung

Ich möchte mich bei Priv.-Doz. Dr.-Ing. habil. Matthias Riebisch für die hervorragende Betreuung meiner Dissertation, die vielen interessanten Diskussionen und seine unendliche Geduld bedanken. Ein großes Dankeschön geht zudem an meine Frau und an meine Familie, deren tatkräftige Unterstützung mir die Dissertation erst ermöglichte.

Außerdem gilt mein Dank den Mitarbeitern des Fachgebietes Softwaresysteme/Prozessinformatik, die durch die zahlreichen Vorschläge und Ideen meine Dissertation vervollständigten.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Erläuterung
ADL	Architectural Description Language
ALMA	Architecture Level Modifiability Analysis
ANSI	American National Standards Institute
ATAM	Architecture Tradeoff Analysis Method
COM	Component-Object-Model
CPM	Critical Path Methd
CPM	Critical Path Method
CPU	Central Processing Unit
DLL	Dynamic-Link-Library
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
EV	Expected Value
FCM	Factor Criteria Metrics
FTP	File Transport Protocol
GoF	Gang of Four
GPL	GNU General Public License
GQM	Goal Question Metric
GUI	Graphical User Interface
ISO	International Organization for Standardization
IIS	Internet Information Server
IuK	Information und Kommunikation
J2EE	Java 2 Enterprise Edition
LDAP	Lightweight Directory Access Protocol
MFC	Microsoft-Foundation-Class
MPEG	Moving Picture Experts Group
MVC	Model View Controller
PERT	Program Evaluation and Review Technique

Abkürzung	Erläuterung
PT	Personentage
SAAM	Software Architecture Analysis Method
SMS	Short Message Service
SOA	Service-Oriented-Architectures
SQL	Structured Query Language
SSL	Secure Sockets Layer
STL	Standard-Template-Library
UI	User-Interface
UML	Unified Modelling Language

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Quelltextauszüge

Kapitel 1

Einleitung, Zielstellung und Vorgehensweise

In allen Softwareentwicklungsprojekten müssen permanent die *richtigen* Entscheidungen getroffen werden, um mit den knappen Ressourcen Zeit, Budget, Mitarbeiter etc. die Ziele im gewünschten Umfang zu erreichen. Dabei werden alltäglich Entscheidungen getroffen, die in ganz unterschiedlicher Art und Weise sowohl in den Ablauf des Entwicklungsprojektes als auch in die Qualität des Softwareproduktes eingreifen. So werden einerseits Entscheidungen zu den organisatorischen und technologischen Rahmenbedingungen des Projektes getroffen, deren Auswirkungen und Folgen für die Architektur, für das Softwaresystem sowie für das Projekt überschaubar sind und valide prognostiziert werden können. Da diese Entscheidungen im Verhältnis zu den schwerwiegenden Entscheidungen (erläutert im folgenden Absatz) nur geringfügig in die Entwicklungsarbeit eingreifen, können Fehlentscheidungen meist mit planbarem Aufwand neben dem Tagesgeschäft korrigiert werden. Entscheidungen über die organisatorischen und technologischen Rahmenbedingungen sind beispielsweise die Art und Weise der Testautomatisierung oder der Einsatz eines bestimmten Konfigurationsmanagement-Tools für die Entwicklungsarbeit. Sollte sich bei einem mittelgroßen Entwicklungsprojekt mit 20 Entwicklern im Anschluss an die Entscheidung herausstellen, dass das gewählte Konfigurationsmanagement-Tool zu unflexibel und bei einer Verdopplung der Nutzerzahlen nicht performant genug ist, ist der Aufwand für einen Austausch planbar und in wenigen Wochen durchgeführt.

Andererseits werden in Entwicklungsprojekten alltäglich Entscheidungen getroffen, die weitreichende, schwer zu überblickende und schwer zu korrigierende Veränderungen an den Eigenschaften des Softwaresystems beinhalten. Den Entscheidungsträgern ist zudem oftmals gar nicht bewusst, dass sie auch mit einer einzelnen, überhastet getroffenen Entscheidung eine Kettenreaktion in Gang setzen können, die am Ende die Ziele des Entwicklungsprojektes insgesamt gefährdet. Bei der Filialsoftware in Deutschlands größtem Retail- und Logistikunternehmen wurde beispielsweise aufgrund einer Architekturentscheidung indirekt ein sog. Stiller-Alarm deaktiviert, der im Falle einer Notsituation oder eines Überfalls für einen Hilferuf benutzt werden konnte. Da dieser Fehler beim Systemtest nicht erkannt wurde und die Funktion nur selten benutzt wird, stand diese wichtige Sicherheitsfunktionalität zehntausenden Arbeitsplatzsystemen nicht zur Verfügung.

Zu den schwerwiegenden Entscheidungen in einem Entwicklungsprojekt zählen vor allem Architekturentscheidungen; als Architekturentscheidung werden im Rahmen der vorliegenden Dissertation Entscheidungen verstanden, mit denen Komponentenstrukturen, Verantwortlichkeiten und Beziehungen zwischen Komponenten (intern) und den Komponenten von Drittsystemen (extern) festgelegt werden (siehe u. a. [Hof+00] u. [Pos+04]). Speziell die Auswirkungen von Architekturentscheidungen auf die Systemeigenschaften und Seiteneffekte sowie auf den Grad der Zielerreichung können bei großen Softwareprojekten mit 50 und mehr Entwicklern in ihrer Gesamtheit nicht mehr von Einzelpersonen überblickt werden. Die Ursachen liegen u. a. in der Vielzahl konkurrierender Ziele, den unterschiedlichen Eigenschaften der Lösungsansätze sowie den Unsicherheiten über fachliche und technische Abhängigkeiten (siehe Abb. 1.1). Gerade die Abhängigkeitsbeziehungen sind oftmals nicht klar erkennbar, da die relevanten Informationen unvollständig sind oder nicht die erforderliche formale Genauigkeit besitzen. Architekturentscheidungen zählen daher zu den komplexesten und riskantesten Entscheidungen bei Entwicklungsprojekten.

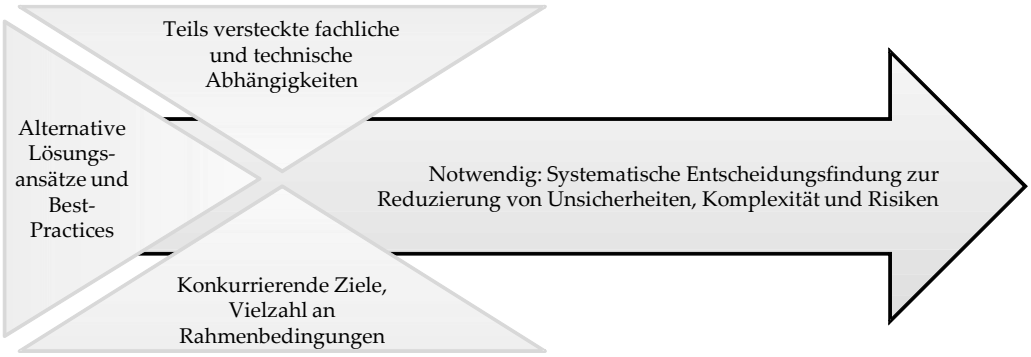


Abb. 1.1: Notwendigkeit für eine systematische Entscheidungsfindung

Die Erfahrungen aus der Praxis zeigen zudem, dass Architektur-Fehlentscheidungen sowie die daraus resultierenden Probleme und Schwachstellen erst im Laufe der Zeit erkannt werden und sich schleichend vergrößern, da die notwendigen Korrekturen aufgrund von engen Rollout- und Testterminen nicht im erforderlichen Umfang durchgeführt werden können. So ist in derart großen Entwicklungsprojekten häufig zu beobachten, dass selbst performance-mindernde Provisorien und Workarounds eine sehr lange Lebensdauer besitzen. In diesen Fällen wird der schleichende Verfall der Architektur (der sog. 'Architectural-Decay') so lange ignoriert, wie die damit verbundenen Aufwände, z. B. Kosten für manuelle Korrekturen oder ein höherer Analyse- und Testaufwand, toleriert werden können. Die Kombination aller Architekturprobleme führt ab einem bestimmten Zeitpunkt, z. B. Implementierung einer neuen Funktionalität oder Anpassungen an ein Drittsystem, zu unerwarteten und plötzlichen Problemen, deren Beseitigung mit einem hohen Kosten- und Zeitaufwand verbunden ist. Konkrete Beispiele für Architektur-Fehlentscheidungen und deren verheerende Auswirkungen lassen sich in allen Anwendungsdomänen finden. Ein Beispiel aus eigener Erfahrung ist wiederum die Filialsoftware, die bereits im vorangegangenen Beispiel Verwendung fand. Um 15, 30 oder 50 Euro Prepaid-Telefonkarten verkaufen zu können, wurde die Filialsoftware um die entsprechende Funktionalität ergänzt und restrukturiert. Aufgrund unberücksichtigter Abhängigkeiten zwischen dem Buchungssystem des Mobilfunkanbieters und der Filialsoftware traten während der Laufzeit Schnittstellen-Probleme auf. Für jeden Verkauf einer Prepaid-Telefonkarte waren umfangreiche manuelle Nachbuchungen im Backend notwendig, deren Kosten mit 50 Euro je verkaufter Telefonkarte geschätzt wurden – bei mehreren hundert verkauften Telefonkarten pro Tag. Dieses Beispiel zeigt, dass ein historisch gewachsenes Schnittstellendesign bei einem neuen Anwendungsfall plötzlich zu Problemen, aufwendigen Nacharbeiten und spürbaren finanziellen Verlusten führen kann.

Die bereits existierenden Methoden zur Entscheidungsunterstützung berücksichtigen jedoch die spezifischen Eigenschaften von Softwarearchitekturen zu wenig. Sie unterstützen einerseits Entwurfsentscheidungen auf Klassen und Methodenebene und sind daher zu feingranular und codeorientiert (z. B. Qasar [Bosc00]). Die Planung und die Entscheidung über Architekturveränderungen erfordert hingegen eine abstraktere und gröbere Sicht auf das zugrunde liegende Softwaresystem, da gerade in Großprojekten meist nur unvollständige Informationen vorliegen und diese nicht die erforderliche formale Genauigkeit haben. Aufgrund enger Release- und Rolloutpläne steht zudem nur selten die Zeit für umfassende und detailliere Analysen und Recherchen zur Verfügung. Die Entscheidungstheorie bietet andererseits eine Entscheidungsunterstützung auf einer groben und abstrakten Ebene. Aufgrund der fehlenden Berücksichtigung softwaretechnischer Aspekte ist das Vorgehen jedoch für Architekturentscheidungen zu generisch und allgemein. So ist beispielsweise eine Unterstützung bei der Erkennung und Berücksichtigung von Abhängigkeitsbeziehungen erforderlich. Zwischen der zu feingranularen und zu generischen Entscheidungsunterstützung fehlt ein an die Spezifika von Architekturentscheidungen angepasstes Vorgehen.

Ziele

Mit der vorliegenden Dissertation wird das Ziel verfolgt, die Komplexität, Unsicherheiten und Risiken bei Architekturentscheidungen zu reduzieren, um aufwandsintensive Korrekturen zu vermeiden und die Ziele der Entscheidung im erforderlichen Umfang zu erreichen. Es ist daher ein geeigneter Entscheidungsprozess für Architekturentscheidungen zu entwickeln, um ausgehend von den Zielen und Rahmenbedingungen systematisch alternative Lösungsansätze zu entwickeln, zu bewerten und eine Entscheidung nach rationalen Gesichtspunkten treffen zu können. Ausgehend von der Kritik an bereits existierenden Methoden und Konzepten zur Entscheidungsunterstützung muss der entwickelte Entscheidungsprozess die speziellen Eigenschaften von Softwarearchitekturen berücksichtigen. So ist es einerseits erforderlich, dass versteckte Abhängigkeitsbeziehungen sichtbar gemacht werden – trotz Unsicherheiten aufgrund unvollständiger Informationen. Andererseits ist es erforderlich, dass komplexe Entscheidungen in handhabbare Einzelentscheidungen zerlegt werden, um dadurch Unsicherheiten und Risiken aufgrund der kombinatorischen Komplexität reduzieren zu können.

In diesem Zusammenhang ist ein ökonomisch sinnvolles Verhältnis zwischen dem Aufwand zur Entscheidungsfindung und dem erwarteten Nutzen einzuhalten, damit die ohnehin knappen Entwicklungskapazitäten nicht mehr als unbedingt notwendig belastet werden. Ein gerade für die Implementierungsphase wichtiges Ziel ist die Dokumentation und Nachvollziehbarkeit der Entscheidungsfindung. Nach Abschluss der Entscheidung muss sichergestellt werden, dass nachvollzogen und begründet werden kann, unter welchen Bedingungen, Umfeldfaktoren sowie Annahmen die Entscheidung getroffen wurde. Die Nachvollziehbarkeit dient einerseits der Rechtfertigung und stellt andererseits die Grundlage für die Wiederverwendung von Analyse- und Bewertungsergebnissen dar.

Zudem muss der Entscheidungsprozess in die zugrunde liegenden Entwicklungsprozesse der Projekte integriert werden können. Ein besonderer Fokus der Dissertation liegt hierbei auf der Anwendbarkeit bei architekturorientiertem Refactoring. Die Erfahrungen aus der Praxis zeigen, dass ein großer Bedarf an Qualitätsverbesserungen bei existierenden Systemen besteht, da eine Vielzahl starrer Altsysteme schnell wachsende Unternehmen behindert. Das Refactoring ist ein geeignetes Verfahren zur Wiederherstellung einer hohen Softwarequalität, ohne funktionale Systemeigenschaften zu verändern [Opdy92]. Dies kann einerseits durch das bekannte Quellcode-Refactoring erfolgen, bei dem u. a. Methoden oder Klassen umbenannt, aufgeteilt oder verschoben werden, um Qualitätsmerkmale wie die Wart-, Analysier- oder Testbarkeit zu erhöhen [Fowl06]. Andererseits kann ein Refactoring der Architektur vorgenommen werden, um durch die Restrukturierung der Komponentenstruktur und deren Beziehungen die Architekturqualität zu erhöhen und die funktionalen Systemeigenschaften unverändert zu lassen [Sun+01].

Vorgehensweise

Zu Beginn der Dissertation wird untersucht, welche der existierenden Methoden und Konzepte geeignet sind, um Architekturentscheidungen systematisch und nach rationalen Gesichtspunkten treffen zu können. Ein besonderer Fokus liegt hierbei auf der Analyse, in welcher Art und Weise das in der Entscheidungstheorie beschriebene generische Vorgehen zur Entscheidungsfindung bei Architekturentscheidungen ergänzt und erweitert werden kann. Auf dieser Grundlage wird ein Vier-Phasen-Entscheidungsprozess entwickelt, der Methoden und Konzepte beinhaltet, um ausgehend von den Zielen, Rahmenbedingungen und der existierenden Architektur systematisch alternative Lösungsansätze zu entwickeln, zu bewerten und eine Entscheidung zu treffen.

Der entwickelte Entscheidungsprozess wird zunächst insgesamt vorgestellt, um die charakteristischen Eigenschaften, wie Phasen und Tätigkeiten, zu erläutern. Ein wichtiger Aspekt bei dieser phasenübergreifenden Prozessbeschreibung ist die Erläuterung, wie die systematische Berücksichtigung der Abhängigkeitsbeziehungen erfolgt und wie die Aufteilung komplexer Architekturentscheidungen in handhabbare Einzelentscheidungen durchgeführt wird. Im Anschluss werden anhand der einzelnen vier Phasen die Methoden und Konzepte im Detail vorgestellt, die zur systematischen Entscheidungsfindung erforderlich sind. Anhand von zwei Praxisbeispielen wird die Anwendung des Entscheidungsprozesses prototypisch aufgezeigt. In der abschließenden Evaluierung wird ermittelt, in welchem Umfang die Ziele der Dissertation erfüllt und welche der offenen Fragen in zukünftigen Arbeiten thematisiert werden.

Beitrag

Erwartetes Ergebnis der vorliegenden Dissertation ist ein strukturiertes und planvolles Verfahren für Architekturentscheidungen nach rationalen Gesichtspunkten. Es wird erwartet, dass das systematische Verfahren dazu beträgt, dass alle Stakeholder – Entwickler, Kunde oder Projektleitung – ein einheitliches und tiefgründiges Verständnis über die geplanten Architekturveränderungen, die zur Verfügung stehenden Lösungsansätze und deren Vor- und Nachteile erhalten.

Durch den Entscheidungsprozess werden einerseits Unsicherheiten, die Komplexität und Risiken von Architekturentscheidungen reduziert, um Fehlentscheidungen und deren negative Konsequenzen möglichst zu vermeiden. Andererseits tragen die während der Entscheidungsfindung erstellten Modelle, Entscheidungsbäume und Analysen dazu bei, die Nachvollziehbarkeit der Entscheidung zu gewährleisten.

Da Analyse- und Bewertungstätigkeiten der Entscheidungsfindung in optionale und obligatorische Tätigkeiten unterteilt werden, kann der Aufwand zur Entscheidungsfindung flexibel an das Risiko und die Komplexität der Architekturentscheidung angepasst werden. Es wird erwartet, dass damit der Aufwand zur Entscheidungsfindung in einem ökonomisch sinnvollen Verhältnis zum Aufwand steht.

Kapitel 2

Relevante Methoden und Konzepte zur Unterstützung von Architekturentscheidungen

In den folgenden Abschnitten werden die existierenden Methoden und Verfahren auf ihren Beitrag zur Entscheidungsunterstützung bei Architekturentscheidungen untersucht und bewertet. Die Auswahl erfolgt auf der Grundlage des generischen Vorgehens zur Entscheidungsfindung aus der Entscheidungstheorie. Die Untersuchung und Bewertung fokussieren auf jene Methoden, Konzepte und Verfahren, welche einen Beitrag zur **Strukturierung der Ziele,** zur **Vorauswahl der alternativen Lösungsansätze,** zur **Ermittlung der Eigenschaften** oder zur **Bewertung der Lösungsansätze** leisten (siehe Abschnitt 2.1 auf S. 23 für eine detailliertere Vorgehensbeschreibung). Die identifizierten Lücken in der Entscheidungsunterstützung sind die Treiber für Risiken, die Komplexität und Unsicherheiten bei Architekturentscheidungen und daher die Grundlage für die Verfeinerung der Ziele der Dissertation (siehe Abschnitt 2.11 auf S. 42ff.) sowie die Entwicklung des Entscheidungsprozesses für Architekturentscheidungen.

2.1 Umfang der Unterstützung durch die präskriptive Entscheidungstheorie

Eine methodische Unterstützung für die Handhabung und Lösung komplexer Entscheidungsprobleme stellt die präskriptive Entscheidungstheorie dar [EiWe03]. Diese verfolgt das Ziel, die Risiken komplexer Entscheidungen auch bei unvollständiger Informationsbasis durch ein rationales Vorgehen bei der Entscheidungsfindung zu reduzieren. Dabei ist jedoch zu überprüfen, welcher Grad an Rationalität bei Architekturentscheidungen in Großprojekten unter Aufwand-Nutzen-Aspekten erreicht werden kann. Die deskriptive Entscheidungstheorie, die das Entscheidungsverhalten von Individuen auf kognitiver Ebene beschreibt, wird in der vorliegenden Dissertation nicht betrachtet.

Ableitung von Rationalitätskriterien für Architekturentscheidungen

Der Begriff Rationalität wird in der Entscheidungstheorie gemäß dem lateinischen Wort 'Ratio' einerseits als die Vernunft definiert [EiWe03]; ein rationales Handeln ist demnach ein von der Vernunft bestimmtes Handeln. Andererseits wird die Rationalität auch als Verhältnismäßigkeit zwischen Mittel und Zweck aufgefasst. Ein rationales Handeln liegt nach dieser Definition vor, wenn der angestrebte Zweck und die verwendeten Mittel in einem vernünftigen Maß zueinanderstehen. Bezogen auf Architekturentscheidungen liegt ein rationales Handeln dann vor, wenn die Reduzierung von Unsicherheiten, Risiken und der Komplexität der Entscheidung (Zweck) in einem vernünftigen Maß zum Aufwand der Entscheidungsfindung (Mittel) steht. Ob das Maß zwischen Mittel und Zweck *vernünftig* ist, kann jedoch meist nur nach Abschluss der Entscheidung (ex post) ermittelt werden.

Um die Einhaltung der Rationalität bereits zum Zeitpunkt der Entscheidungsfindung (und damit ex ante) bestimmen zu können, wird in der Entscheidungstheorie eine Vielzahl von Kriterien aufgeführt (siehe u. a. [EiWe03], [Simo78] oder [Eis+07]). Zu den wichtigsten Kriterien zählt zum einen die prozedurale Rationalität, bei der die Entscheidungsfindung, der Prozess der Erhebung, Strukturierung, Auswertung und Bewertung der Entscheidungsgrundlagen, objektiv und an Fakten orientiert sein muss. Zum anderen müssen die Entscheidungsgrundlagen (Ziele, Analysen, Annahmen) vollständig und konsistent sein.

Die prozedurale Rationalität sowie das vernünftige Maß zwischen Mittel und Zweck sind wichtige Voraussetzungen, um mit einem Entscheidungsprozess für Architekturentscheidungen die Risiken, Komplexität und Unsicherheiten reduzieren zu können. Die hohen Anforderungen an die Entscheidungsgrundlagen nach Vollständigkeit und Konsistenz können in Großprojekten aufgrund des meist vorherrschenden Kosten- und Budgetdrucks nicht vollständig umgesetzt werden. Die Erfahrungen aus der Praxis zeigen zudem, dass aus vollständigen und konsistenten Entscheidungsgrundlagen nicht in jedem Fall ein entsprechend hoher Nutzen erzielt werden kann. Für die Entwicklung des Entscheidungsprozesses ist unter pragmatischen Gesichtspunkten zu konkretisieren, welcher Anspruch auf Rationalität bei Architekturentscheidungen sinnvoll und gerechtfertigt ist. Dies erfolgt im Zusammenhang mit der Entwicklung der einzelnen Phasen des Entscheidungsprozesses.

Beitrag des generischen Vorgehens zur Entscheidungsfindung

Zur Erreichung der rationalen Prozeduralität wird in der Entscheidungstheorie ein generisches Verfahren zur Entscheidungsfindung vorgeschlagen. Die grundlegende Hypothese ist dabei die Reduzierung der Komplexität und Risiken einer Entscheidung durch die Dekomposition des Entscheidungsproblems und die isolierte Analyse und Bewertung der Bestandteile. Erst im Anschluss an die Dekomposition werden die Bestandteile der Entscheidung in Kombination betrachtet, um den besten Lösungsansatz zur Erfüllung der Ziele systematisch und rational ermitteln zu können. Das generische Vorgehen umfasst die folgenden Aktivitäten (siehe für eine detaillierte Beschreibung [EiWe03] oder [Eis+01]):

- *Schritt 1: Strukturierung der Ziele und Rahmenbedingungen:* Der Ausgangspunkt der Entscheidungsfindung sind die Ziele, welche durch die Umsetzung eines Lösungsansatzes oder einer bestimmten Handlung erreicht werden sollen. Die Ziele sind zu identifizieren und widerspruchsfrei zu strukturieren. Neben den Zielen sind auch die Rahmenbedingungen der Entscheidung zu erheben und widerspruchsfrei zu strukturieren, z. B. Budget- oder Terminvorgaben.
- *Schritt 2: Vorauswahl der alternativen Lösungsansätze:* Zur Erreichung der Ziele können meist alternative Lösungsansätze (sog. Alternativen oder Handlungsoptionen) eingesetzt werden. Im Rahmen der Vorauswahl werden jene Lösungsansätze ausgewählt, die zur Erreichung der Ziele unter den gegebenen Rahmenbedingungen geeignet sind.
- *Schritt 3: Ermittlung der Eigenschaften je Lösungsansätze:* Für einen systematischen Vergleich der Lösungsansätze sind die Eigenschaften im Detail zu ermitteln. Dazu zählen u. a. der Umfang der Zielerreichung, Risiken oder Seiteneffekte.
- *Schritt 4: Bewertung der Lösungsansätze und Entscheidung:* Die Ergebnisse aus dem vorangegangenen Schritt sind die Grundlage für den Vergleich der Lösungsansätze und die Bestimmung des besten Ansatzes. Ein Lösungsansatz wird umso höher bewertet, je stärker die Eigenschaften zur Zielerreichung beitragen.

Für das generische Vorgehen wird in der Entscheidungstheorie eine Vielzahl von Methoden und Formeln vorgestellt (siehe [EiWe03] oder [Simo78]), um u. a. die Ziele widerspruchsfrei zu strukturieren, Wahrscheinlichkeiten für den Eintritt von Ereignissen zu ermitteln oder die alternativen Lösungsansätze bei einem oder mehreren Zielen zu bewerten. Die Methoden und Formeln sind jedoch für Architekturentscheidungen zu ergänzen und zu verfeinern. Das Vorgehen zur Entscheidungsfindung ist zwar im betriebswirtschaftlichen Bereich ein allgemein akzeptierter Prozess zur Entscheidungsfindung [Saat01], z. B. bei riskanten und komplexen Investitionsentscheidungen mit unsicheren Annahmen über Kosten und zukünftige Erlöse. Im Hinblick auf Architekturentscheidungen fehlen jedoch Konzepte und Methoden, um die softwaretechnischen Details über die Architektur und die Lösungsansätze in der erforderlichen Tiefe analysieren zu können. Detaillierte Analysen sind jedoch die Voraussetzung, um Risiken und die Gefahr von Seiteneffekten reduzieren zu können.

Zur Entwicklung eines Entscheidungsprozesses für Architekturentscheidungen ist das generische Vorgehen angesichts der folgenden Fragestellungen entsprechend anzupassen und zu erweitern:

- Welche Ziele und welche Rahmenbedingungen sind speziell für Architekturentscheidungen relevant? Durch welche Analyseverfahren können die Schwachstellen und die kritischen Bereiche der Architektur des Softwaresystems identifiziert und verfeinert werden? Wie können Einschränkungen aufgrund wichtiger Vorentscheidungen berücksichtigt werden?
- Welche alternativen Lösungsansätze stehen zur Behebung der Schwachstellen und zur Erfüllung der Ziele zur Verfügung? Wie kann ein systematischer Auswahl- und Verfeinerungsprozess gestaltet werden?
- Wie können die für den Vergleich und die Bewertung der Lösungsansätze notwendigen Eigenschaften, u. a. Grad der Zielerreichung oder notwendiger Implementierungsaufwand, realistisch abgeschätzt werden, ohne mit großem Aufwand lauffähige Prototypen entwickeln zu müssen?

2.2 Vergleich verschiedener Ansätze zur Messung von Wahrscheinlichkeiten

Um die Unsicherheiten und Risiken einer Architekturentscheidung reduzieren zu können, sind unsichere Entscheidungsfaktoren durch systematische Verfahren zu erheben. Zu den unsicheren Entscheidungsfaktoren zählen u. a. Grad der Zielerreichung durch die Implementierung eines Lösungsansatzes oder die Wahrscheinlichkeit des Eintritts von Ereignissen. Die Erfahrungen aus der Praxis zeigen, dass unsichere Entscheidungsfaktoren oft nur subjektiv geschätzt werden, selten realistisch sind und daher häufig die zentrale Ursache von Fehlentscheidungen sind. So kann beispielsweise mit Sicherheit ermittelt werden, was der Kauf von Softwarelizenzen kostet, aber es kann oftmals nur grob abgeschätzt werden, welcher Aufwand mit der Implementierung des Softwareproduktes verbunden ist. Um unsichere Entscheidungsfaktoren systematisch erheben zu können, steht eine Reihe von Methoden und Konzepten zur Verfügung. Bei der Anwendung der Methoden und Konzepte muss jedoch beachtet werden, ob ein ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis erreicht werden kann.

Zur Messung von Wahrscheinlichkeiten können grundsätzlich die folgenden vier Ansätze unterschieden werden (u. a. [EiWe03] oder [Eis+07]):

(1) *Frequentistischer Ansatz*

(2) Symmetrie-Ansatz

(3) Subjektiver Ansatz

(4) Objektiver Ansatz

(1) Frequentistischer Ansatz: Bei dieser Art der Wahrscheinlichkeitsmessung werden relative Häufigkeiten von identisch wiederholbaren Vorgängen und deren Ereignissen untersucht. Grundlage dieser Messungsart sind identische Vorgänge aus der Vergangenheit. Aus deren Häufigkeit wird die Wahrscheinlichkeit von zukünftigen Ereignissen ermittelt. So kann beispielsweise anhand der Anzahl fehlerhafter Buchungen ermittelt werden, wie hoch die Fehlerquote bei identischen Buchungen in der Zukunft sein wird. Da identisch wiederholbare Vorgänge bei der Softwareentwicklung die Ausnahme sind, ist diese Art der Wahrscheinlichkeitsmessung bei Architekturentscheidungen nicht anwendbar.

(2) Symmetrie-Ansatz: Der Aspekt der Symmetrie bezieht sich auf die Gleichverteilung der Eintrittswahrscheinlichkeiten von elementaren Ereignissen. So liegt die Wahrscheinlichkeit, dass mit einem Wurf eines idealen Würfels die Zahl eins oder sechs gewürfelt werden kann, bei allen Zahlen identisch bei 1/6. Aus den Eintrittswahrscheinlichkeiten der elementaren Ereignisse kann auf komplexere Sachverhalte geschlossen werden. So liegt die Wahrscheinlichkeit, mit einem Wurf eine gerade Zahl zu würfeln, bei 1/6 + 1/6 + 1/6 und somit bei 50 %. Diese Sicht auf Wahrscheinlichkeiten ist ebenfalls nicht bei Architekturentscheidungen anwendbar, da in der Softwareentwicklung keine elementaren Einzelereignisse mit gleich verteilten Wahrscheinlichkeiten existieren.

(3) Subjektiver Ansatz: Die Wahrscheinlichkeit eines Ereignisses ist nach dieser Auffassung der Grad des Vertrauens einer oder mehrerer Personen in den Eintritt des Ereignisses. Die Wahrscheinlichkeit wird bei diesem Ansatz auf der Grundlage von Vermutungen und Annahmen geschätzt. Daher ist der Informations- und Wissensstand der Person oder der Personengruppe über die Entscheidungssituation von großer Bedeutung für die Qualität der Schätzung. Nur wenn dieser Personenkreis mit den technischen Details der Architektur vertraut ist, können verwertbaren Aussagen über die Eintrittswahrscheinlichkeit von Ereignissen im Zusammenhang mit einer Architekturentscheidung getroffen werden. Für die subjektive Ermittlung der Wahrscheinlichkeiten ist jedoch nur ein geringer Erhebungsaufwand erforderlich. Unter Aufwand-Nutzen-Aspekten eignet sich diese Art der Messung von Wahrscheinlichkeiten zumindest für risikoarme Architekturentscheidungen.

(4) Objektiver Ansatz: Aufgrund der hohen Varianz ist die Qualität subjektiver Wahrscheinlichkeitsmessungen umstritten. Demgegenüber steht die Objektivität, die sich als Wert einer Eigenschaft intersubjektiv nachprüfen lässt. Im engeren Sinne ist die Objektivität von Wahrscheinlichkeiten jedoch nicht erreichbar. Zwar gelten relative Häufigkeiten von Ereignissen als objektive Datengrundlage; daraus können Wahrscheinlichkeiten jedoch nur in Form einer subjektiven Hypothese abgeleitet werden. Auf der Grundlage objektiver Daten können Wahrscheinlichkeiten allerdings mit einer höheren Qualität abgeleitet werden als ausschließlich auf der Grundlage subjektiver Einschätzungen. Da dabei kein hundertprozentiger Grad an Objektivität erreicht wird, ist abzuwägen, ob das verbleibende Restrisiko im Rahmen der Entscheidung akzeptiert werden kann oder ob zusätzliche Analysen erforderlich sind. Abgesehen von Architekturentscheidungen für gefährliche oder sehr sicherheitskritische Anwendungen, z. B. Steuerung von Raketensystemen oder Atomkraftwerken, kann angenommen werden, dass ein hoher Grad an Objektivität, möglichst nahe bei hundertprozentiger Objektivität, ausreichend ist. Gerade bei dieser Art der Wahrscheinlichkeitsmessung muss das Aufwand-Nutzen-Verhältnis beachtet werden. Die Erfahrungen aus der Praxis zeigen, dass bei risikoarmen Entscheidungen ein sehr hoher Grad an Objektivität oft keinen zusätzlichen Nutzen erbringt. Dieser Ansatz ist daher nur bei riskanten Architekturentscheidungen anzuwenden, bei denen die objektive Ermittlung der Wahrscheinlichkeiten einen hohen Beitrag zur Reduzierung von Unsicherheiten und Risiken bietet.

In Tab. 2.1 sind die verschiedenen Arten zur Messung von Wahrscheinlichkeiten und deren Eignung[1] für Architekturentscheidungen zusammenfassend dargestellt.

Messungsansatz	Eignung	Begründung	Methoden
Frequentistischer Ansatz	-	Bei Architekturentscheidungen kann diese Form der Messung nicht durchgeführt werden.	---
Symmetrie-Ansatz	-	Bei Architekturentscheidungen kann diese Form der Messung nicht durchgeführt werden.	---
Subjektiver Ansatz	+	Trotzdem die Güte subjektiver Messungen stark von der Erfahrung der beteiligten Personen abhängt, sind sie aufgrund des guten Aufwand-Nutzen-Verhältnisses gerade bei risikoarmen Entscheidungen geeignet.	Fragen nach Wahrscheinlichkeiten, Fragen nach Werten einer unsicheren Variable, direkte oder indirekte Messung (siehe Anhang 10.1, S. 168ff.)
Objektiver Ansatz	++	Zwar ist eine Ermittlung der Wahrscheinlichkeiten mit sehr hoher Objektivität aus Aufwand-Nutzen-Gründen nur bei riskanten und komplexen Architekturentscheidungen gerechtfertigt; dieser Ansatz verspricht jedoch den höchsten Beitrag zur Beherrschung von Unsicherheit sowie zur Senkung von Risiken und Komplexität.	Messung der Verteilung von diskreten und kontinuierlich steigenden Zufallsvariablen durch Wahrscheinlichkeits-/Dichtefunktionen oder durch Verteilungsfunktionen (siehe Anhang 10.1, S. 169ff.); Nutzung des Theorems von Bayes (siehe Anhang, S.170ff.)

Tab. 2.1: Übersicht über die Eignung der verschiedenen Ansätze zur Messung von Wahrscheinlichkeiten

1 (-) nicht geeignet, (+) bedingt geeignet, (++) gut geeignet

2.3 Abgrenzung von Qualitätsmerkmalen und Vergleich von Strukturierungsansätzen

Abgrenzung von Merkmalen der Architekturqualität

Die Softwarequalität allgemein und die Architekturqualität im Speziellen können durch eine Vielzahl von Merkmalen beschrieben werden, die sich häufig nur in der Bezeichnung unterscheiden. Eine strukturierte und systematische Entscheidung erfordert jedoch ein einheitliches Verständnis über die Qualitätsmerkmale, gerade bei der Beschreibung der Schwachstellen und Ziele der Entscheidung. Die Erfahrungen aus der Praxis zeigen beispielsweise, dass das Qualitätsmerkmal Wartbarkeit ganz unterschiedlich verstanden wird. Während die Entwickler eher auf die Aspekte Analysier- und Testbarkeit fokussieren, wird die Wartbarkeit auf Managementebene oftmals nur als Flexibilität und Erweiterbarkeit eines Softwaresystems verstanden. Um ein einheitliches Verständnis über die Merkmale der Architekturqualität zu erhalten, sind aus der Vielzahl an unterschiedlichen Qualitätsmerkmalen jene Merkmale abzugrenzen und im Detail zu definieren, welche die Architekturqualität von Softwaresystemen beschreiben.

Die Softwarequalität ist definiert als Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht [IS8402]. Es gibt ausgehend von dieser und von anderen Definitionen (siehe z. B. [IEE610]) verschiedene Interpretationsansätze für die Softwarequalität [Garv84]. Für die Architekturqualität ist die Differenzierung nach Prozess- und Produktqualität wichtig [Balz00]. Die *Prozessqualität* beinhaltet die Merkmale des Prozesses der Softwareentwicklung, z. B. Effizienz oder Transparenz der Entwicklungsarbeit. Die *Produktqualität* fokussiert auf die Eigenschaften und Merkmale des Softwareproduktes, z. B. auf eine leichte Verständlichkeit und Wartbarkeit. Mit der ISO-Norm 9126 existiert eine Standardisierung dieser Qualitätsmerkmale [IS9126].

In Abb. 2.1 sind die Qualitätsmerkmale der ISO-Norm 9126 dargestellt. Sie reflektieren eine innere und eine äußere Sicht auf die Qualitätsmerkmale eines Softwaresystems [IS9126]. Die Architekturqualität, als innere Sicht auf die strukturellen Eigenschaften des Softwaresystems (sog. innere Qualitätsmerkmale), wird durch die Qualitätsmerkmale Zuverlässigkeit, Wartbarkeit, Portabilität, Funktionalität, Sicherheit und Effizienz beschrieben. Die Benutzungsqualität sowie die Benutzbarkeit beschreiben hingegen die äußere Sicht des Anwenders auf die Software (sog. äußere Qualitätsmerkmale). Veränderungen an der Architekturqualität können sich direkt oder indirekt auf die äußerlich sichtbaren Qualitätsmerkmale eines Softwaresystems auswirken [Beva99]; dies hängt vom Umfang und der Reichweite der Architekturveränderungen ab. Die Erfahrungen aus der Praxis zeigen, dass beispielsweise eine nur minimal verbesserte Zuverlässigkeit nur indirekt an den äußerlich sichtbaren Qualitätsmerkmalen erkennbar ist. Wurde die Zuverlässigkeit hingegen grundlegend und strukturell verbessert, ist dies äußerlich meist direkt erkennbar.

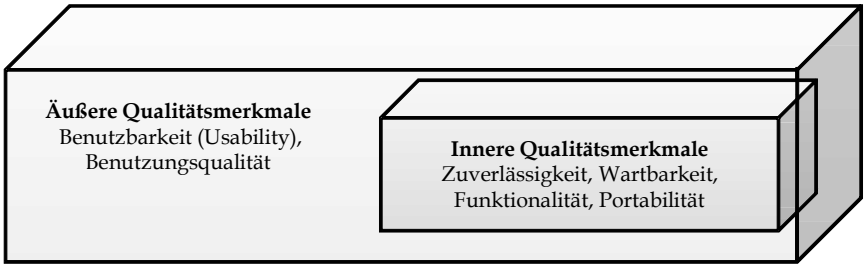


Abb. 2.1: Produkt-Qualitätsmerkmale nach ISO 9126

Mit der ISO-Norm 9126 steht eine standardisierte Sicht auf die Qualitätsmerkmale von Softwarearchitekturen zur Verfügung, welche ein einheitliches Verständnis der Merkmale erleichtert. Vor allem die inneren Qualitätsmerkmale, wie Zuverlässigkeit oder Wartbarkeit, können durch Architekturveränderungen direkt beeinflusst werden. Die inneren Qualitätsmerkmale sind daher eine Grundlage für die Entwicklung des Entscheidungsprozesses, vor allem bei der Identifizierung und Strukturierung der Ziele (siehe Abschnitt 5.1, S. 56ff.) sowie bei der Gestaltung eines Katalogs mit alternativen Lösungsansätzen zur Verbesserung der Architekturqualität (siehe Abschnitt 7, S. 138ff.). Je nach Entscheidungssituation

kann es jedoch erforderlich sein, ergänzende Qualitätsmerkmale zur ISO-Norm 9126 zu definieren, z. B. für die *Flexibilität*, die *Variabilität* oder die *Testbarkeit*, die in der ISO-Norm 9126 nur am Rande betrachtet werden.

Vergleich von Verfahren zur Strukturierung von Qualitätszielen

Da die Qualitätsziele in der Praxis oft nur lückenhaft bekannt und abstrakt sind sowie in Widersprüchen und Ziel-Mittel-Beziehungen zueinander stehen [Beva99], ist ein Verfahren zur widerspruchsfreien Strukturierung der Qualitätsziele erforderlich. Zudem stehen Qualitätsziele – im Gegensatz zu funktionalen Zielen – nur in indirektem Zusammenhang mit den funktionalen Eigenschaften des Softwaresystems und bedürfen einer Transformation, um die erforderlichen funktionalen Veränderungen zur Erfüllung der Qualitätsziele erkennen zu können. Ein methodisches Strukturierungsverfahren ist speziell im Hinblick auf die Entwicklung von alternativen Lösungsansätzen zur Umsetzung der Qualitätsziele erforderlich. Verbleiben Widersprüche und ungenau definierte Ziele, steigt das Risiko, dass die falschen Lösungsansätze ausgewählt werden.

Zur Strukturierung von Qualitätszielen können Qualitätsmodelle eingesetzt werden. Ein Qualitätsmodell ist eine Systematik, nach welcher die Qualitätsziele festgelegt, in Kriterien operationalisiert und mit entsprechenden Indikatoren untersetzt werden [Balz00]. Über ein Qualitätsmodell werden die Qualitätsanforderungen auf Qualitätsmerkmale übertragen; dies sind im Rahmen der vorliegenden Dissertation insbesondere jene der ISO-Norm 9126 [IS9126] (siehe vorangegangener Abschnitt). Zur Reduzierung der Risiken, Unsicherheiten und Komplexität einer Architekturentscheidung muss ein Qualitätsmodell widerspruchsfrei sein (siehe zu den Rationalitätsanforderungen Abschnitt 2.1, S. 22). Es können drei Ansätze zur Strukturierung von Qualitätszielen unterschieden werden, für die jeweils unterschiedliche Strukturierungsverfahren existieren:

(1) Abbildung aller Ziele in einem einzelnen Modell

(2) Strukturierung der Ziele durch verschiedene Sichten

(3) Klassifizierung der Ziele in Fundamental- und Instrumentalziele

(1) Abbildung aller Ziele in einem einzelnen Modell: Der erste Ansatz umfasst die Strukturierungsverfahren, die alle Ziele in einem Modell integriert abbilden. Die wichtigsten sind das Factor-Criteria-Metrics-Model (FCM) [Fen+97] oder die Goal-Question-Metric (GQM) [Bas+02]. Im Rahmen des FCM werden grobe und ungenaue Qualitätsziele schrittweise verfeinert sowie anhand von konkreten Indikatoren bzw. Metriken operationalisiert. Es existiert zwar eine Vielzahl von Codemetriken; für die Beschreibung von Merkmalen der Architekturqualität stehen jedoch nur wenige Metriken zur Verfügung (siehe Abschnitt 2.4, S. 28f.). Das Strukturierungsverfahren des FCM kann auch rekursiv angewendet werden, um detaillierte Qualitätsanforderungen auf abstraktere Qualitätsziele zurückzuführen. Bei der GQM wird beschrieben, in welchem Kontext (Question) das Qualitätsziel (Goal) zum Softwaresystem steht und über welche Metriken es gemessen werden kann. Das FCM ist eher als Strukturmodell zu verstehen, während die GQM ein Vorgehen beschreibt, wie Qualitätsveränderungen gemessen werden können. Da bei diesen Strukturierungsverfahren alle Ziele in einem Modell abgebildet werden, ist eine widerspruchsfreie Beschreibung der Softwarequalität i. d. R. nicht möglich. Das zeigt sich beispielsweise daran, wenn die Performance *und* die Wartbarkeit gleichzeitig zu verbessern sind und bei der Abbildung der beiden Ziele in einem Modell Widersprüche verbleiben.

(2) Strukturierung der Ziele durch verschiedene Sichten: Der zweite Ansatz beinhaltet Qualitätsmodelle, die die Qualitätsziele nach verschiedenen Sichten strukturieren, z. B. die Sicht des Entwicklers auf das Softwareprodukt. Die wichtigsten Vertreter sind die Arbeiten von McCall, Richards und Walters [McC+77] sowie das Qualitätsmodell von Boehm [Boeh78]. Zentraler Ansatzpunkt der Arbeiten von McCall, Richards und Walters sind Sichten der 'Entwickler', der 'Revision und Weiterentwicklung' und des 'Benutzers' auf die Qualitätsmerkmale des Softwaresystems. Diesen Sichten sind typische Qualitätsziele zugeordnet. Bei Boehm werden die Qualitätsziele aus Sicht des Anwendernutzens, der Portabilität und der Wartbarkeit betrachtet. Durch die Fokussierung auf verschiedene Sichten der Qualitätsmerkmale eines Softwaresystems können die Qualitätsziele systematisch strukturiert und gruppiert werden. Da jedoch Ziel-Mittel-Beziehungen, die mehrere Sichten betreffen, durch diese Qualitätsmodelle nicht abgebildet werden können, ist eine widerspruchsfreie Strukturierung der

Qualitätsziele nur eingeschränkt gegeben. Solche sichten-übergreifenden Ziel-Mittel-Beziehungen können – am Beispiel der Sichten von Boehm – auftreten, wenn die Verbesserung eines Zieles der Sicht Wartbarkeit Auswirkungen auf Ziele der Sichten Portabilität und Wartbarkeit hat.

(3) Klassifizierung der Ziele in Fundamental- und Instrumentalziele: Der dritte betrachtete Ansatz zur Strukturierung von Zielen wird im Rahmen der Entscheidungstheorie angewendet (siehe zur Entscheidungstheorie Kapitel 2.1, S. 22). Die Strukturierung der Ziele erfolgt über eine Klassifizierung nach Fundamental- und Instrumentalzielen, mit Abbildung in einem Ziel-Wirkungs-Modell [Cle+01] (u. [EiWe03]). Die Erfüllung von stets niederrangigen Instrumentalzielen ist die Voraussetzung zur Erfüllung der Fundamentalziele. Die Instrumentalziele sind somit das Mittel zur Zielerfüllung und geben direkte Hinweise auf die durchzuführenden Architekturveränderungen, beispielsweise bei Restrukturierung einer komplexen Komponentenbeziehung zur Verbesserung des Fundamentalziels Wartbarkeit. Zudem können Ziel-Wirkungsmodelle flexibel aufgeteilt werden. So kann ein Fundamentalziel in einem anderen Ziel-Wirkungsmodell als Instrumentalziel dienen. Aufgrund dieser Freiheitsgrade eignet sich dieses Klassifizierungsverfahren für die widerspruchsfreie Strukturierung der Qualitätsziele in besonderem Maße. Ein Beispiel für ein Ziel-Wirkungs-Modell ist in Abb. 4.2 auf S. 54 abgebildet.

2.4 Vergleich verschiedener Verfahren zur Architekturanalyse

Architekturanalysen sind bei Architekturentscheidungen in zweierlei Hinsicht erforderlich. Zum einen dienen sie zur systematischen Erkennung und Untersuchung von Schwachstellen in der Architektur sowie zur Identifizierung und Verfeinerung der für die Entscheidung relevanten Ziele. Dies ist gerade bei großen und über Jahre gewachsenen Softwaresystemen erforderlich, bei denen die Architektur lange Zeit einem schleichenden Verfall ausgesetzt war. Zum anderen sind Architekturanalysen ein wichtiges Hilfsmittel, um alternative Lösungsansätze – die Alternativen der Entscheidung – im Hinblick auf Zielerreichung, Risiken und Seiteneffekte vergleichen und bewerten zu können. Für die Analyse von Softwaresystemen stehen quantitative und qualitative Verfahren zur Verfügung [Lin+03], mit denen die Architektur einerseits möglichst vollständig und umfassend analysiert werden muss; die Erfahrungen aus der Praxis zeigen, dass neben der Quellcodeanalyse die Analyse eines bereits existierenden, ggf. informalen und unvollständigen Architekturmodells möglich sein muss. Andererseits müssen die Ergebnisse der Analyse valide und widerspruchsfrei sein, damit die Komplexität und Unsicherheit bei einer Architekturentscheidung gesenkt werden können.

Quantitative Architekturanalyseverfahren

Bei quantitativen Analyseverfahren wird die Qualität des Quellcodes mittels Indikatoren oder Metriken analysiert und ausgewertet [Pos+04]. Deutliche Abweichungen von der erforderlichen Quellcode-Qualität geben Hinweise auf Schwachstellen und einen notwendigen Verbesserungsbedarf. Zur Durchführung einer quantitativen Analyse ist daher zunächst die erforderliche Quellcode-Qualität in Form von Metriken zu definieren und mit entsprechenden Schwell- und Grenzwerten zu hinterlegen. Dazu kann z. B. die Goal-Question-Metric (siehe vorangegangener Abschnitt 2.3) angewendet werden.

Für die quantitative Architekturanalyse werden zwei Gruppen an Metriken unterschieden: codeorientierte und architekturorientierte Metriken. In der Gruppe der codeorientierten Metriken steht eine Vielzahl an Metriken zur Verfügung, welche die Qualität des Quellcodes bewerten, z. B. das Verhältnis von Fehlern zur Anzahl der Codezeilen [Pos+04]. Jedoch lassen sich nicht mit allen codeorientierten Metriken strukturelle Schwächen ermitteln. Ausnahmen sind z. B. Analysen der Änderungshäufigkeiten in bestimmten Komponenten. Dabei wird ermittelt, in welchen Komponenten Änderungen und Fehlerhäufigkeiten korrelieren bzw. in welchen Komponenten nach Änderungen viele Fehlerkorrekturen notwendig gewesen sind. In der Gruppe der architekturorientierten Metriken steht nur eine geringe Anzahl an Metriken zur Verfügung [Lin+03], z. B. das Distanzmaß [Sim+99] oder die von Shereshevsky vorgestellten Kopplungs- und Kohäsionsmetriken [She+01]. Mit diesen Metriken wird auf Komponentenebene ermittelt, wie stark die Kopplung und Kohäsion zwischen einzelnen Komponenten sind. Damit lassen sich Schwachpunkte in der Komponentenstruktur ermitteln.

Wie alle quantitativen und damit kennzahlenbasierten Analysen zeichnen sich die Analyseergebnisse durch eine hohe Validität aus, da die Werte objektiv sind und automatisiert erhoben werden können. Jedoch steht der Vielzahl an – für die Architekturanalyse wenig brauchbaren – codeorientierten Metriken nur eine geringe Anzahl an Architekturmetriken gegenüber. Mit quantitativen Analysen kann zudem ein Architekturmodell oder eine existierende Dokumentation nicht berücksichtigt werden. Ein weiterer problematischer Aspekt der quantitativen Architekturanalyse betrifft die unterschiedliche Interpretierbarkeit der Ergebnisse, die aufgrund des indirekten Bezugs zur Architekturqualität zu Widersprüchen führen kann. Ein Beispiel ist die hohe Kohäsion und lose Kopplung von Komponenten, die nach allgemeinem Verständnis in der Praxis eine Voraussetzung für eine gute Analysier- und Testbarkeit sind. Da jedoch für jedes Softwaresystem individuell entschieden werden muss, welcher Grad an Kohäsion und Kopplung erforderlich ist, führen die Analyseergebnisse in der Praxis oft zu unterschiedlichen Interpretationen und Widersprüchen.

Qualitative Architekturanalyseverfahren

Zu den Methoden der qualitativen Architekturanalyse zählen neben der Analyse mit sog. Architecture-Smells hauptsächlich szenariobasierte Analysemethoden. Eine weitere Analysemethode mit der sich im Detail Algorithmen und einzelne Systembereiche untersuchen lassen ist die Graphenanalyse ([Lan+97] und [Men+02]). Da für die Graphenanalyse zunächst eine formale Beschreibung des betreffenden Teils der Architektur mittels Graphen erforderlich ist, welche sehr komplex ist und einen hohen Beschreibungsaufwand erfordert, wird dieses Analyseverfahren aufgrund des oft ungünstigen Aufwand-Nutzen-Verhältnisses nicht weiter betrachtet.

Eines der wichtigsten szenariobasierten Architekturanalyseverfahren ist die Architecture-Tradeoff-Analysis-Method (ATAM) [Kaz+00]. Sie stellt eine Erweiterung und Konkretisierung der Software-Architecture-Analysis-Method (SAAM) dar [Kaz+95]. Die Architekturanalyse bei ATAM gliedert sich in zwei Phasen mit verschiedenen Tätigkeiten. In der ersten Phase erfolgt die Identifizierung der erforderlichen funktionalen und vor allem der nicht-funktionalen Anforderungen (der sog. Utility-Tree) sowie der relevanten Komponenten der Architektur des Softwaresystems. In der zweiten Phase werden Szenarien entworfen, die typische oder extreme Ereignisse und Zustände im Zusammenhang mit den Zielen beschreiben. Ein Beispiel ist der Ausfall einer Hardwarekomponente für das Qualitätsziel Zuverlässigkeit. Anhand der Szenarien wird analysiert, wie geeignet die aktuelle Architektur für die Erfüllung der Qualitätsziele ist und welche Stärken und Schwächen existieren. In die Analyse der Szenarien und in die Diskussion über die Stärken und Schwächen werden die Kunden sowie andere Stakeholder explizit mit einbezogen. Das Konzept von ATAM wurde bei der Architecture-Level-Modifiability-Analysis (ALMA) weiterentwickelt, um die Modifizierbarkeit von Architekturen anhand von Veränderungsszenarien zu untersuchen [Ben+04]. Jedoch umfassen ATAM wie auch ALMA über die reine Szenarioanalyse hinaus eine Vielzahl von Aktivitäten, die einen hohen Aufwand erfordern und bei Architekturentscheidungen oftmals zu keinem erkennbaren Nutzen führen. So wird bei ALMA und ATAM u. a. eine Vielzahl von Präsentationen und Diskussionen durchgeführt. Da bei Architekturentscheidungen hingegen nur ein Entscheidungsträger oder ein Entscheidungsgremium existiert, kann eine Vielzahl der zeitaufwendigen Präsentationen und Diskussionen entfallen.

Neben der Szenarioanalyse kann die Architektur auch anhand von Architecture-Smells analysiert werden [RoLi04]. Architecture-Smells beschreiben typische problematische Muster, Strukturen und Schwachstellen in einem Softwaresystem und deren Auswirkungen auf die Softwarequalität. Die Architektur eines Softwaresystems kann anhand der Smells zielgerichtet im Hinblick auf Schwachstellen und Problembereiche analysiert sowie bewertet werden. Roock und Lippert führen fünf verschiedene Gruppen von Architecture-Smells und deren Auswirkungen auf die Softwarequalität auf, die in Tab. 2.2 (siehe S. 30) dargestellt sind.

Gruppen von Architecture-Smells	Beschreibung der Problematik
Smells in Benutzungsgeflechten	Ein typischer Smell ist eine Klasse, von der zur Laufzeit keine Instanzen gebildet oder deren Instanzen nicht verwendet werden.
Smells in Vererbungshierarchien	Zwischen Klassen existieren i. d. R. Vererbungshierarchien. Ein Smell ist eine spekulative Generalisierung oder eine zu tiefe Vererbungshierarchie.
Smells in Packages	Eine Vielzahl von Programmiersprachen bietet Packages als Strukturierungshilfe an (z. B. Java, C++). Problematisch sind eingebundene Packages, die nicht oder nur selten benutzt werden.
Smells in Subsystemen	Ein Subsystem ist ein gekapselter Teil eines Softwaresystems. Ein Beispiel für einen Smell ist ein zu großes oder kleines Subsystem.
Smells in Schichten	Schichten dienen der logischen Strukturierung eines Softwaresystems. Problematisch sind zyklische Abhängigkeiten zwischen den Schichten.

Tab. 2.2: Übersicht über verschiedene Architecture-Smells

Die betrachteten qualitativen Analyseverfahren sind bei der Durchführung zeitaufwendiger als quantitativen Analysen, da für die Analyse eines informalen Architekturmodells nur eine geringe Werkzeugunterstützung verfügbar ist. Für eine bessere Werkzeugunterstützung und eine automatisierte Durchführung der Analysen, z. B. durch Simulationsverfahren wie bei Rapide [Rap+97] (siehe Tab. 2.3, S. 31), müsste eine formale oder semi-formale Architekturbeschreibung vorliegen. Der Aufwand für eine formale oder semi-formale Architekturbeschreibung übersteigt jedoch gerade bei Komplexen Systemen die Einsparungsmöglichkeiten durch die automatisierte Durchführung der Analysen. Zudem sind die Ergebnisse der qualitativen Analyseverfahren nicht so klar und eindeutig wie Kennzahlen oder Metriken, sondern häufig widersprüchlich und von geringer Objektivität. Das ist insbesondere dann der Fall, wenn die Szenarien unpassend, nicht detailliert und aussagekräftig genug sind oder die Architecture-Smells nur oberflächlich geprüft werden.

Andererseits können die Methoden der qualitativen Architekturanalyse sehr flexibel eingesetzt werden. Über eine Untersuchung des Quellcodes hinaus kann ein Architekturmodell mit umfangreichen Rahmenbedingungen systematisch in die Analyse einbezogen werden. Qualitative Analysen sind vor allem dann schon einsetzbar, wenn – wie bei der Entwicklung neuer Softwaresysteme – erst ein informales Grobkonzept der Architektur vorhanden ist. Die Ergebnisse der qualitativen Analyseverfahren haben trotz der geringeren Objektivität eine höhere Aussagekraft als Kennzahlen und Metriken; daher sind die qualitativen Analyseverfahren, vor allem die szenariobasierten Verfahren wie beispielsweise ATAM oder ALMA, einer quantitativen Untersuchung vorzuziehen. Da die szenariobasierten Analyseverfahren jedoch ergänzende Aktivitäten umfassen (z. B. Diskussionen und Präsentationen), die oftmals keinen erkennbaren Nutzen bei konkreten Architekturentscheidungen erbringen, ist der Umfang der Analyseaktivitäten aus Aufwand-Nutzen-Gründen auf ein Minimum zu reduzieren.

2.5 Eignung von Methoden und Konzepten zur Architekturbeschreibung

Eine Architekturbeschreibung, bzw. die Erstellung eines Modells der Architektur, ist eine der wichtigsten Grundlagen für die Entscheidungsfindung. Das Modell wird u. a. dafür benötigt, um die Architektur über die reine Quellcodeanalyse hinaus untersuchen oder die Eignung alternativer Lösungsansätze zur Behebung der Schwachstellen und Erreichung der Ziele bewerten zu können. Für den Einsatz im Rahmen einer Architekturentscheidung ist nur bei hohen Risiken eine formale oder semi-formale Architekturbeschreibung erforderlich. Der hohe Aufwand, um eine Architektur in dieser Qualität beschreiben zu können, steht nur selten in einem ökonomisch sinnvollen Verhältnis zum dadurch erzielbaren Nutzen. Zudem steht gerade bei der Entwicklung eines neuen Softwaresystems oft nur ein Grobkonzept der Architektur und der Umsysteme zur Verfügung. Ein Architekturmodell muss in jedem Fall aktuell sein und alle relevanten Komponenten, Bezie-

hungen und Umgebungsfaktoren enthalten, welche durch die Architekturentscheidung betroffen sind.[2] Der dafür notwendige Aufwand zur Modellierung muss in einem sinnvollen Verhältnis zum Nutzen für die Analysen und die Bewertungen stehen. Ergänzend dazu muss ein Architekturmodell – auch ein Modell, welches nur einen kleinen Teilbereich der Architektur abbildet – widerspruchsfrei und eindeutig interpretierbar sein, um eine systematische Entscheidungsfindung nach rationalen Gesichtspunkten zu ermöglichen (siehe zu den Rationalitätskriterien Abschnitt 2.1, S. 22). Die Erfahrungen aus der Praxis zeigen, dass vage Architekturbeschreibungen in Textform oder auf Präsentationsfolien eine Vielzahl von Widersprüchen enthalten und von den Stakeholdern unterschiedlich interpretiert und verstanden werden.

Vergleich verschiedener Architektur-Beschreibungssprachen

Zur widerspruchsfreien Architekturbeschreibung steht eine Vielzahl von Modellierungssprachen und Modellierungskonzepten zur Verfügung. Modellierungssprachen, mit denen speziell Architekturmodelle beschrieben werden können, werden auch als Architectural-Description-Language (ADL) bezeichnet; eine der bekanntesten ADL ist ACME [Gar+00]. Die wichtigsten und in der Praxis häufig eingesetzten Sprachen werden in Tab. 2.3 kurz beschrieben und auf ihre Eignung zur Erfüllung der Anforderungen (siehe vorangegangener Absatz) bewertet.

ADL	Beschreibung	Bewertung
ACME [Gar+00]	Statische Beschreibung von Softwaresystemen mittels Komponenten, Ports und Konnektoren	Aufgrund der breiten Vielfalt an Strukturierungs- und Darstellungsmitteln können mit ACME alle wesentlichen Komponenten und Beziehungen eines Softwaresystems in einer statischen Sicht beschrieben werden. Da dynamische Verhaltensaspekte nicht dargestellt werden können, verbleiben Unsicherheiten und Widersprüche in Bezug auf Ablaufsteuerung oder Kontroll- und Datenflussstrukturen (siehe Abb. 2.2 auf S. 32).
Rapide [Rap+97]	Beschreibungssprache mit dem Ziel einer späteren Simulierung des Softwaresystems	Rapide wird vorwiegend zur Modellierung ereignisbasierter Softwaresysteme eingesetzt, da hier eine Simulierung (z. B. des Laufzeitverhaltens zur Analyse der Performance) sinnvoll ist. Sofern eine Simulierung erforderlich ist, ist Rapide eine geeignete Beschreibungssprache. Es gilt jedoch zu bedenken, dass zur Beschreibung einer simulationsfähigen Architektur oftmals Teile der Architektur modelliert werden müssen, die für die aktuell betrachtete Architekturentscheidung nicht relevant sind.
UML [Boo+06]	Modellierung der statischen und dynamischen Aspekte eines Softwaresystems	Trotzdem die Modelle eher auf die Klassen und Methodenebene fokussieren sind sie prinzipiell zur Beschreibung von Softwarearchitekturen geeignet. Die Erfahrungen aus der Praxis zeigen, dass die Modelle der UML allgemein bekannt sind und von den Stakeholdern eindeutig interpretiert werden können. Darüber hinaus existiert eine Vielzahl von Werkzeugen zur Überprüfung der Modelle auf Widersprüche und Inkonsistenzen.
UniCon [Zele96]	Beschreibung der grundlegenden Komponenten von Systemen mit Schwerpunkt auf Konnektoren	UniCon bietet aufgrund einer rudimentären Syntax und des Schwerpunkts auf der Beschreibung der Konnektoren nur eine geringe Unterstützung für die Beschreibung komplexer Softwaresysteme. Die Modelle sind zu verkürzt und abstrakt, enthalten dadurch zu viele Widersprüche.
xADL [Das+01]	Erweiterbare, XML-basierte Beschreibungssprache (aktuelle Version xADL 2.0)	Mit xADL können die verschiedenen Ebenen einer Architektur einheitlich beschrieben werden (z. B. Verteilung, Kommunikation etc.). Durch die XML-basierte Syntax sind die Modelle untereinander kompatibel. Die Sprache ist gut zur Modellierung komplexer Softwarearchitekturen geeignet – im Vergleich zur UML jedoch kaum verbreitet. Analog zur UML existieren verschiedene Werkzeuge zur Überprüfung der Modelle auf Widersprüche und Inkonsistenzen.

Tab. 2.3: Eignung von Architektur-Beschreibungssprachen

2 Zur Identifikation von Komponenten bei einem existierenden Softwaresystem kann auf geeignete Hilfsmittel zurückgegriffen werden, z. B. auf die Architecture-Reconstruction-Method [Yan+99]. Mithilfe der Rekonstruktionsmethode werden die Grundzüge der Architektur aus dem Quelltext des Softwaresystems heraus extrahiert. Es werden jene zusammengehörigen Teile des Quellcodes identifiziert, die als eigenständige Komponenten betrachtet werden können. Gerade bei Softwaresystemen, die über Jahre gewachsen sind, herrscht in der Praxis meist keine Klarheit über Komponenten und Komponenten-Strukturen mehr.

Eine exemplarische Beschreibung einer Client-Server-Struktur in der Syntax der Beschreibungssprache ACME ist in Abb. 2.2 dargestellt. 'Client' und 'Server' stellen Komponenten eines Systems dar, die 'Ports' für den Nachrichtenaustausch besitzen. Der Austausch erfolgt über einen 'Connector', der die Rollen 'requestor' und 'requestee' einnimmt. Über 'Attachments' wird die Richtung des Nachrichtenaustausches zwischen Client und Server festgelegt. Auf der rechten Seite sind die beiden Komponenten und deren Beziehung grafisch abgebildet. Durch die semiformale ACME-Syntax ist eine weitgehend widerspruchsfreie der Architektur möglich.

Textuelle Beschreibung

```
System ClientServerSystem = {
  Component server = {
    Port requests;
  };
  Component client = {
    Port makeRequest;
  };
  Connector req = {
    Role requestor;
    Role requestee;
  };
  Attachments {
    server.requests to req.requestor;
    client.makeRequest to req.requestee;
  }
}
```

Grafische Darstellung

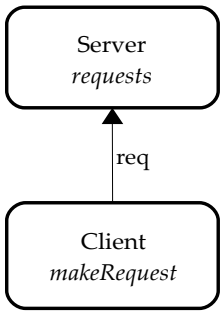


Abb. 2.2: Beispiel einer Architekturbeschreibung mittels ACME

Entsprechend den Ausführungen in Tab. 2.3 eignen sich ACME und vor allem die UML zur Beschreibung der Architekturen komplexer, geschäftskritischer Softwaresysteme. Im Gegensatz zu ACME können mit der UML auch dynamische Aspekte eines Systems modelliert werden. Die Modelle der UML sind außerdem allgemein bekannt und es existiert eine Vielzahl an Modellierungs- und Überprüfungswerkzeugen, z. B. zur Überprüfung auf Inkonsistenzen und Widersprüche. Jedoch muss berücksichtigt werden, dass die UML-Modelle maßgeblich darauf ausgerichtet sind, ein System auf der Abstraktionsebene von Klassen, Objekten und Methoden zu beschreiben. Die xADL ist aufgrund der Fokussierung auf die Beschreibung von Softwarearchitekturen, der Erweiterbarkeit und der XML-basierten Syntax besser geeignet, aber in der Praxis nahezu unbekannt. Das erschwert die Interpretation und Analyse einer in xADL beschriebenen Architektur. Mit steigendem Bekanntheitsgrad sollte in Erwägung gezogen werden, die Sprache xADL anstatt oder als Ergänzung zur UML einzusetzen.

Beitrag von Sichten-Konzepten zur Reduzierung der Komplexität

Speziell bei großen und über Jahre gewachsenen Softwaresystemen ist die Architektur oft sehr komplex und erfordert einen hohen Modellierungsaufwand. Zur Reduzierung der Komplexität kann das Architekturmodell in verschiedene Sichten aufgeteilt werden [Kruc01]. Die Sichten fokussieren auf unterschiedliche organisatorische oder technische Aspekte der Architektur und erleichtern dadurch die Analyse der Architektur hinsichtlich von Schwachstellen und die Bewertung alternativer Lösungsansätze. Typische Sichten auf eine Architektur sind in der Praxis vor allem Sichten auf die Kontroll- und Datenflüsse oder auf die Komponentenstrukturen. In der Literatur existieren verschiedene Konzepte, um ein Architekturmodell in zusammengehörige Sichten aufzuteilen. Zur Reduzierung der Komplexität bei der Entscheidungsfindung müssen die Sichten speziell auf die technischen Aspekte der Architektur fokussieren, da dort die größten Unsicherheiten und Risiken stecken. Zu den wichtigsten Sichten-Konzepten zählen die folgenden:

- *4+1 Sichten von Kruchten [Kruc01]:* Kruchten unterscheidet zwischen 'Logical-', 'Process-', 'Implementation-', 'Deployment-' und einem übergreifenden 'Use-Case-View'. Die Sichten fokussieren auf einen bestimmten Bereich der Architektur und blenden andere Aspekte aus. Der 'Process-View' beschreibt z. B. die Kontroll- und Datenflüsse im Softwaresystem. Implementierungsaspekte werden ausschließlich in der 'Implementation-View' abgebildet.

- *4 Sichten von Hofmeister, Nord und Soni [Hof+00]:* Die Autoren unterscheiden die folgenden Sichten: 'Conceptual-', 'Module-', 'Execution-' und 'Code-Architecture-View'. Während die ersten drei Sichten denen von Kruchten im wesentlichen entsprechen, fokussiert der 'Code-Architecture-View' auf die Module und den Quellcode des Softwaresystems.
- *3 Viewtypes von Clements [Cle+03]:* Die Viewtypes von Clements beschreiben hierarchische Sichten auf die Architektur. Es werden 'Module-', 'Component-Conntector-' und 'Allocation-Viewtypes' unterschieden. Durch den in der Hierarchie obersten 'Module-Viewtype' erfolgt eine statische Abbildung der Strukturen auf Modulbasis. Der mittlere Viewtype bildet die Prozesssicht ab und der unterste 'Allocation-Viewtype' die Implementierungssicht, z. B. Zuordnung von Aufgaben an Teams oder Allokation von Ressourcen.

Durch den Einsatz von Sichten wird die Verständlichkeit und Übersichtlichkeit bei komplexen Architekturmodellen verbessert. Die 4+1 Sichten von Kruchten und die 4 Sichten von Hofmeister et al. sind zur Architekturbeschreibung gut geeignet, da sie auf die Abbildung der technischen Aspekte der Architektur fokussieren. Das Sichten-Konzept von Clements ist breiter gefasst und beinhaltet auch organisatorische Aspekte, z. B. u. a. Ressourcenallokation. Aufgrund der Vielzahl an spezifischen Eigenschaften der Sichten-Konzepte kann keine allgemeine Aussage über die Eignung eines bestimmten Sichten-Konzeptes für Architekturentscheidungen getroffen werden. Es muss im konkreten Entscheidungsfall entschieden werden, welcher Bereich der Architektur für die Architekturentscheidung zu betrachten ist, und welches Sichten-Konzept zur Reduzierung der Komplexität und Unsicherheiten und zur Erleichterung der Verständlichkeit beiträgt.

2.6 Abgrenzung von Quellen für die Herkunft von Lösungsansätzen

Zur Erreichung der Ziele werden im Rahmen der Entscheidungsfindung alternative Lösungsansätze (sog. Architekturalternativen oder Handlungsoptionen) ausgewählt und verfeinert (siehe generisches Vorgehen in Absatz 2.1, S. 23). Die Auswahl geeigneter Lösungsansätze kann einerseits auf der Grundlage des Erfahrungswissens[3] des Entscheidungsträgers (z. B. des Architekten) mit ähnlichen Problemstellungen erfolgen. Andererseits stehen verschiedene Quellen, u. a. Muster und Stile, zur Verfügung, anhand derer Lösungsansätze systematisch entwickelt werden können. Soll z. B. die Wartbarkeit verbessert werden, indem häufig zu wartende Komponenten gekapselt werden, kann dies mittels der Entwurfsmuster 'Fassade' oder mit einem 'Kompositum' erfolgen (siehe Abschnitt 7.1.3, S. 141). Da eine sehr große Menge an unterschiedlichen Quellen in der Literatur sowie Praxis existiert, gilt es abzugrenzen, welche zur Entwicklung und Veränderung von Architekturen am besten geeignet sind. So sind Entwurfsmuster, welche rudimentäre Veränderungen am Quellcode beschreiben, aufgrund ihres geringen Abstraktionsgrades nur bedingt als Grundlage für die Entwicklung weitreichender Architekturveränderungen geeignet.

Entwurfsmuster, Architekturmuster und Architekturstile

Die größte Quelle für die Entwicklung alternativer Lösungsansätze sind Entwurfsmuster, Architekturmuster und Architekturstile. Entwurfs- und Architekturmuster sind dokumentierte, allgemein akzeptierte und häufig implementierte Musterlösungen für typische Probleme bei der Softwareentwicklung [Gamm01]. Architekturstile beschreiben hingegen grundlegende Strukturen von Softwaresystemen; im Detail wird die Art der Kontrollstrukturen und der Datenflüsse zwischen den Komponenten beschrieben [Bosc00].

Entwurfsmuster, die erste Gruppe von Quellen für die Entwicklung von Lösungsansätzen zur Architekturentwicklung und -veränderung, beinhalten Lösungen für Probleme beim Softwareentwurf. Sie haben einen starken Fokus auf die Quellcodeveränderungen der Klassen- und Objektebene. Ein Beispiel für ein typisches Entwurfsproblem ist die Vielzahl von Abhängigkeitsbeziehungen zwischen Klassen oder Objekten, welche die Analysier- und Testbarkeit und somit die Wartung sowie Weiterentwicklung eines Softwaresystems erschweren. Eine passende Lösung beschreibt beispielsweise das Ent-

3 Wissen ist aufgrund der Immaterialität ein schwer einzugrenzender Begriff. Eine in der Praxis anwendbare Definition wird von Romhardt vorgeschlagen, nach der Wissen als die Gesamtheit der Kenntnisse und Fähigkeiten verstanden wird, die Individuen zur Lösung von Problemen einsetzen [Romh98].

wurfsmuster 'Fassade'. Eine 'Fassade' dient als Schnittstelle zu einem Subsystem, in welchem Klassen oder Objekte gekapselt sind; dadurch können versteckte Abhängigkeitsbeziehungen zu Klassen oder Objekten außerhalb des Subsystems sichtbar gemacht werden. Die Entwurfsmuster gehen maßgeblich auf die Arbeiten von Gamma et al. (sog.' Gang of Four' – GoF) zurück ([Gamm01] und [Keri04]). In den Arbeiten von Gamma werden 37 Entwurfsmuster beschrieben, welche in die Kategorien Konstruktion, Struktur und Verhalten aufgeteilt werden (siehe Tab. 2.4).

Kategorien	**Beispiele für Entwurfsmuster**
Konstruktionsmuster	Abstrakte Fabrik, Fabrikmethoden
Strukturmuster	Observer, Fassade
Verhaltensmuster	Vermittler (Mediator)

Tab. 2.4: Übersicht über die Entwurfsmuster der GoF

Einige Entwurfsmuster fokussieren stark auf den Quellcode des Softwaresystems. Das Muster 'Singelton' stellt z. B. lediglich sicher, dass eine einzige Instanz einer Klasse erzeugt werden kann. Aufgrund des geringen Abstraktionsgrades und der starken Fokussierung auf den Quellcode stellen diese Entwurfsmuster, wie das 'Singelton' und einige andere Entwurfsmuster (siehe Abschnitt 7.1, S. 139ff.), keine Quelle für die Entwicklung von Lösungsansätzen zur Architekturveränderung dar.

Wesentlich abstrakter und weiter reichend als Entwurfsmuster sind Architekturmuster [Bosc00]. Dies sind häufig verwendete Komponentenstrukturen für Probleme beim Entwurf von Architekturen. Während Entwurfsmuster Strukturen innerhalb einer Komponente beschreiben, fokussieren Architekturmuster auf die Strukturierung und die Zusammenarbeit mehrerer Komponenten [Pos+04]. Das Architekturmuster 'Model-View-Controller (MVC)' beschreibt z. B. eine Struktur, wie die Interaktionen zwischen Benutzer und dem System möglichst flexibel verarbeitet werden können. Ein Controller fungiert als Vermittler zwischen der Datenverarbeitung (Model) sowie der Ein- und Ausgabe der Daten (View) (siehe Abb. 2.3).

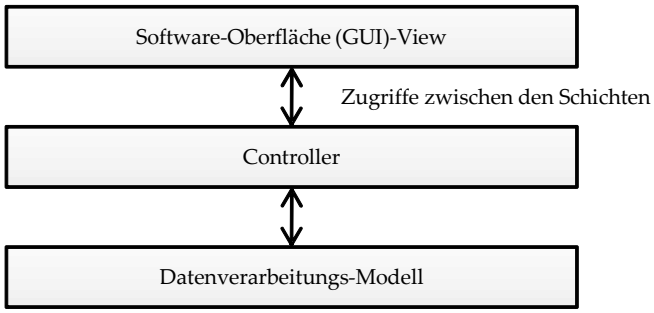


Abb. 2.3: Architekturmuster Model-View-Controller (MVC)

Ein Architekturstil beschreibt den charakteristischen Aufbau eines Softwaresystems oder eines eigenständigen Teils davon [Pos+04]. Ein Architekturstil hat somit einen höheren Abstraktionsgrad als ein Architekturmuster. Mit der Entscheidung für einen Architekturstil werden die Struktur, die Rollen und die Verantwortlichkeiten der Komponenten im Softwaresystem festgelegt [Sha+96]. Ein Beispiel für einen Architekturstil ist die Aufteilung eines Softwaresystems in hierarchische Schichten (sog. 'Layer-Stil'). Durch den 'Layer-Stil' wird festgelegt, welche Interaktionen zwischen den Schichten zulässig sind und wie Schichten übergreifende Aufrufe weiterzuleiten haben.

In einem Softwaresystem können mehrere Architekturstile gleichzeitig implementiert sein; diese Architekturen werden als hybride Architekturen bezeichnet [Fiel00]. Ein Beispiel stellt ein Softwaresystem dar, welches in Schichten strukturiert ist. Innerhalb einer Schicht kann ein weiterer Architekturstil implementiert sein, z. B. 'Pipes-and-Filters'.

Technologien und Softwareprodukte

Neben Mustern und Stilen stellen Technologien sowie Softwareprodukte eine wichtige Quelle für die Entwicklung von Lösungsansätzen zur Architekturentwicklung und -veränderung dar. Die folgenden Kategorien von Technologien und Softwareprodukten sind für die Architekturentwicklung von Bedeutung [Pos+04]:

- *Betriebssysteme und Programmiersprachen:* Ein Betriebssystem stellt die Laufzeitumgebung für ein Softwaresystem zur Verfügung. Mit der Entscheidung für ein geeignetes Betriebssystem kann z. B. die Zuverlässigkeit, aber auch die Effizienz entscheidend verbessert werden. Darüber hinaus können durch den Einsatz einer speziellen Programmiersprache bestimmte Qualitätseigenschaften gezielt verbessert werden, z. B. die Portabilität durch den Einsatz von Java.
- *Bibliotheken:* Bibliotheken stellen eine Sammlung an Funktionen oder Prozeduren in einer bestimmten Programmiersprache zur Verfügung. Sie sind ein Strukturierungsmittel und erleichtern dadurch die Wiederverwendung von Funktionalität. In Java stellt z. B. die Bibliothek 'mysql' die notwendigen Funktionen zur Verfügung, um auf MySQL-Datenbanken zugreifen zu können.
- *Komponententechnologien:* Komponenten sind bereits entwickelte Softwareprodukte, die flexibel bei der Entwicklung von Softwaresystemen und vor allem bei der Wiederverwendung von Funktionalität genutzt werden können. Im Gegensatz zu Bibliotheken ist der Einsatz von Komponenten nicht von der Programmiersprache abhängig, die der Implementierung der Komponente zugrunde liegt.
- *Frameworks:* Unter Frameworks wird eine Sammlung von Funktionalität (z. B. Funktionen, Bibliotheken) zu einem bestimmten Themengebiet verstanden. Ein Beispiel ist die Microsoft-Foundation-Class (MFC), die dem Entwickler ein ereignisorientiertes Framework zur Entwicklung von Benutzeroberflächen zur Verfügung stellt.

Aufgrund der vielfältigen Eigenschaften der Technologien sowie der Softwareprodukte können keine allgemeinen Aussagen über die Eignung getroffen werden. Es ist im konkreten Entscheidungsfall zu entscheiden, mit welcher Technologie und mit welchen Softwareprodukten die Ziele der Architekturentscheidung am besten erfüllt werden können.

2.7 Kategorisierungsansätze zur systematischen Auswahl alternativer Lösungsansätze

Für einen systematischen Auswahlprozess der alternativen Lösungsansätze müssen die nicht-funktionalen Qualitätseigenschaften der Muster und Stile sowie der Technologien und Softwareprodukte bekannt sein, da diese einen positiven oder negativen Beitrag zur Zielerreichung leisten. Jedoch können gerade die Qualitätseigenschaften im Gegensatz zu den funktionalen Eigenschaften oftmals nur durch aufwendige zusätzliche Analysen ermittelt werden (siehe Abschnitt 2.3, S. 26). Nur für wenige bekannte Muster und Stile sind die Qualitätseigenschaften in ausreichendem Umfang dokumentiert und verfügbar. In Tab. 2.5 ist eine Übersicht über die Qualitätseigenschaften einzelner Entwurfsmuster nach Bosch dargestellt [Bosc00]; vergleichbare Bewertungen von Bosch sind für einzelne Architekturmuster und -stile verfügbar.

Entwurfsmuster	Auswirkungen auf die Qualitätsmerkmale
Abstrakte Fabrik, Fabrikmethoden	Diese Entwurfsmuster verbessern die Wartbarkeit, da die Instanziierung von Objekten von deren Verwendung getrennt ist.
Fassade	Durch die Kapselung von Objekten hinter einer Fassade wird die Wart- und Erweiterbarkeit verbessert, da alle Zugriffe auf die gekapselten Klassen über klar definierte Schnittstellen erfolgen.
Vermittler	Ein Vermittler koordiniert und verwaltet komplexe Interaktionen zwischen Objekten. Die Objekte sind lose an den Vermittler gekoppelt, was sich positiv auf die Wartbarkeit auswirkt.

Tab. 2.5: Übersicht der Qualitätseigenschaften einzelner Entwurfsmuster

Um trotz der großen Zahl an Mustern, Stilen, Technologien und Softwareprodukten, bei denen die Qualitätseigenschaften nicht im Detail bekannt sind, eine systematische Auswahl durchführen zu können, ist es erforderlich, die Qualitätseigenschaften auf der Ebene von Kategorien zu betrachten. Durch die Kategorisierung wird ersichtlich, welche Qualitätseigenschaften und Beiträge zur Verbesserung der Qualität der jeweiligen Kategorien zu erwarten sind. Somit kann der Auswahlprozess systematischer gestaltet werden, indem unpassende Kategorien unberücksichtigt bleiben und nur noch innerhalb der verbleibenden Kategorien die besten Muster, Stile, Technologien und Softwareprodukte ausgewählt werden. Darüber hinaus ist es durch die qualitätsorientierte Kategorisierung möglich, individuelle Erfahrungslösungen im Entscheidungsprozess zu berücksichtigen. Erfahrungslösungen sind probate Lösungsansätze, die bei ähnlichen Problemstellungen bereits erfolgreich eingesetzt wurden. Um die Erfahrungslösungen systematisch im Auswahlprozess zu berücksichtigen, müssen sie in die Kategorien eingeordnet werden können. Das Ziel der Dissertation ist es, die Quellen der Lösungsansätze nicht auf Muster, Stile, Technologien und Softwareprodukte zu beschränken, sondern die Erfahrungen der Mitarbeiter explizit – aber systematisch – mit einzubeziehen.

Kategorisierungsansätze für Muster und Stile

Für Muster und Stile stehen die folgenden Kategorisierungsansätze bereits zur Verfügung:

- *Entwurfsmuster:* Gamma verwendet die in Tab. 2.4 (siehe S. 34) dargestellten Kategorien: 'Konstruktions'-, 'Struktur'- und 'Verhaltensmuster'. Buschmann und Zimmer ordnen die Entwurfsmuster Abstraktionsstufen zu. Die Bandbreite reicht von Mustern, die fundamentale Systemstrukturen beschreiben (höchste Stufe), bis hin zu elementaren Mustern, die Probleme im Quellcode thematisieren (niedrigste Stufe) [Bus+05] (u. [Zimm97]).
- *Architekturmuster:* Buschmann unterscheidet Architekturmuster zur grundlegenden 'Strukturierung von Softwaresystemen', 'Muster für verteilte und interaktive Softwaresysteme' sowie 'Muster für adaptierbare Softwaresysteme' [Bus+05]. Ähnliche Kategorien werden auch von anderen Autoren vorgeschlagen (z. B. [Pom+04]).
- *Architekturstile:* Shaw und Garlan differenzieren die Architekturstile nach den folgenden Kategorien: 'Datenflusssysteme', 'Call-and-Return-Systeme', 'Unabhängige Komponenten', 'Zustandsübergangssysteme', 'Virtuelle Maschinen' und 'Datenzentrierte Systeme' [Sha+96]. Dieser Kategorisierungsansatz wird auch von anderen Autoren aufgegriffen (z. B. [Bas+02]).

Die dargestellten Kategorien orientieren sich an funktionalen Gesichtspunkten oder an den möglichen Einsatzgebieten der Muster und Stile. Sie sind für eine qualitative Betrachtung zu grob und ungenau und gehen zu wenig auf die Qualitätseigenschaften ein. So besitzen z. B. die Entwurfsmuster 'Zuständigkeitskette' oder 'Fassade', die beide der Kategorie 'Verhaltensmuster' zugeordnet werden (nach [Gamm01]), unterschiedliche Qualitätseigenschaften. Während die 'Fassade' eine Schnittstelle zu einem Subsystem darstellt und durch die Kapselung die Analysier- und Testbarkeit verbessert, koordiniert die 'Zuständigkeitskette' Objektaufrufe über mehrere Instanzen hinweg und erhöht durch komplexe Kontrollflüsse den Testaufwand. Da diese Kategorisierungsansätze für eine systematische Auswahl der Muster und Stile als Lösungsansätze ungeeignet sind, werden in Abschnitt 7 (siehe S. 138ff.) qualitätsorientierte Kategorien für Muster und Stile vorgestellt.

Kategorisierungsansätze für Technologien und Softwareprodukte

Analog zu den Mustern und Stilen fehlt eine umfassende qualitative Betrachtung der Kategorien von Technologien und Softwareprodukten. Es ist daher ebenfalls eine Analyse der Qualitätseigenschaften der Kategorien notwendig. Da im Vergleich zu Mustern und Stilen eine deutlich größere Anzahl an unterschiedlichen Softwareprodukten und Technologien existiert, ist die Erarbeitung einer speziellen, qualitätsorientierten Kategorisierung im Rahmen der Dissertation nicht sinnvoll, da zu viele spezifische Kategorien entstehen würden. Die Analyse der Qualitätseigenschaften orientiert sich dementsprechend an den von Posch vorgeschlagenen Kategorien 'Betriebssysteme und Programmiersprachen', 'Bibliotheken', 'Komponententechnologien' und 'Frameworks' [Pos+04] (siehe dazu den vorangegangenen Abschnitt 2.6, S. 35).

2.8 Untersuchung und Bewertung von Abhängigkeitsbeziehungen

Die Erfahrungen aus der Praxis zeigen, dass bei der Entscheidungsfindung eine Vielzahl von Abhängigkeitsbeziehungen zwischen Zielen, Rahmenbedingungen, Architekturkomponenten und zwischen einzelnen Entscheidungen berücksichtigt werden muss. Da eine fehlende oder unzureichende Berücksichtigung der Abhängigkeitsbeziehungen eine der Hauptursachen für Fehlentscheidungen ist, müssen einerseits versteckte Abhängigkeitsbeziehungen sichtbar gemacht werden; andererseits sind konkurrierende oder widersprüchliche Beziehungen aufzulösen. Eine systematische Identifizierung und Berücksichtigung der Abhängigkeitsbeziehungen schafft die Grundlage zur Senkung von Unsicherheiten, Komplexität und Risiken bei der Entscheidungsfindung. In der Literatur wird eine Fülle von unterschiedlichen Abhängigkeitsbeziehungen beschrieben. Die aufgeführten Beziehungen sind einerseits darauf hin zu untersuchen, ob sie alle Bestandteile einer Architekturentscheidung – Ziele, Rahmenbedingungen, Architekturkomponenten – abdecken. Andererseits ist zu überprüfen, ob die Vielzahl an Abhängigkeitsbeziehungen überschneidungsfrei ist und ob zur Erleichterung der Verständlichkeit sowie zur Reduzierung der kombinatorischen Komplexität ähnliche Beziehungen gruppiert werden können.

Eine zentrale Referenz für Beziehungen zwischen Architekturentscheidungen stellen die 10 Beziehungsarten von Kruchten dar, die in folgender Liste verkürzt aufgeführt werden (siehe für eine detaillierte Beschreibung der Beziehungen [Kru+06]). Vergleichbare Arbeiten zu Abhängigkeitsbeziehungen finden sich u. a. bei Zimmermann (siehe [Zim+08]).

- *Zwingend – Constrains*: Die Implementierung eines Lösungsansatzes kann Anpassungen an anderen Teilen des Softwaresystems 'erzwingen'. Sofern die Schnittstelle zu einer Datenbank verändert wird, kann es 'zwingend' erforderlich sein, auch die Komponenten anzupassen, die auf die veränderte Schnittstelle zugreifen.
- *Verhindernd – Forbids, Excludes:* Die Implementierung eines Lösungsansatzes kann die Implementierung eines anderen Lösungsansatzes 'verhindern'. So kann der Einsatz einer Schichtenarchitektur dazu führen, dass übergreifende Aufrufe über mehrere Schichten 'verhindert' werden. Dadurch kann einerseits die Verständlichkeit der Kontroll- und Datenstrukturen verbessert werden; andererseits kann es zu Verzögerungen beim Zugriff auf die betreffenden Komponenten kommen.
- *Ermöglicht – Enables*: Die Implementierung eines Lösungsansatzes kann wiederum die Implementierung eines anderen Lösungsansatzes 'ermöglichen'. Der Einsatz eines Adapters für eine Corba-Komponente kann es 'ermöglichen', dass die Komponente auch in einem .NET-Umfeld eingesetzt werden kann.
- *Weitere Beziehungsarten:* 'Zusammenfassend' (Subsumes), 'Konflikt mit' (Conflicts with), 'Überschreibt' (Overrides), 'Enthalten' (Comprises, Is Made of, Decomposes into), 'Ist eine Alternative für' (Is an Alternative to), 'Ist gebunden an' (Is Bound to), 'Steht in Beziehung mit' (Is Releated to).

Die von Kruchten (wie auch von Zimmermann) dargestellten Beziehungen umfassen die in der Praxis häufig auftretenden Abhängigkeitsbeziehungen zwischen verschiedenen Lösungsansätzen bzw. Einzelentscheidungen. Damit steht einerseits eine Übersicht an Abhängigkeitsbeziehungen zur Verfügung, anhand der nach Abhängigkeiten gezielt gesucht werden kann. Andererseits können auf dieser Grundlage Maßnahmen entwickelt werden, um widersprüchliche und problematische Abhängigkeiten im Rahmen der Entscheidungsfindung aufzulösen, was zu einer Reduzierung der Komplexität, der Risiken und der Unsicherheiten beiträgt.

Ein Kritikpunkt betrifft die Überschneidungen zwischen den Beziehungsarten. Trotzdem die Beziehungen unterschiedlich bezeichnet werden, beschreiben mehrere Beziehungen häufig identische Zusammenhänge und Sachverhalte. Da die Überschneidungen eine Quelle für Widersprüche und Fehlinterpretationen sind, ist eine überschneidungsfreie Strukturierung der Abhängigkeiten erforderlich. Nur unter dieser Voraussetzung ist eine systematische und kontinuierliche Berücksichtigung der Abhängigkeitsbeziehungen bei der Entscheidungsfindung möglich. So beschreiben beispielsweise die Beziehungsarten 'verhindern' und 'in Konflikt mit' den identischen Sachverhalt, dass eine Kombination der betreffenden Lösungsansätze nicht möglich ist. Die dargestellten Beziehungsarten decken zudem nur einen Teil der Beziehungen ab, die im Rahmen einer Architekturentscheidung berücksichtigt werden müssen. Neben den Abhängigkeitsbeziehungen zwischen Lösungsansätzen bzw. Einzelentscheidungen müssen auch Abhängigkeiten zwischen den Zielen, den Rahmenbedingun-

gen, den Architekturkomponenten und den Maßnahmen zur Architekturentwicklung und -veränderung berücksichtigt werden. In Abschnitt 4 (siehe S. 50ff.) werden die von Kruchten und Zimmermann aufgezeigten Beziehungen entsprechend ergänzt und erweitert.

2.9 Verfahren zum Vergleich und zur Bewertung alternativer Lösungsansätze

Gegenstand der Entscheidungsfindung sind *zwei* bis *n* alternative Lösungsansätze, mit denen die Architektur in unterschiedlicher Art und Weise verändert und weiterentwickelt werden kann. Die Lösungsansätze sind somit die Alternativen der Entscheidung und müssen durch ein systematisches Verfahren anhand ihres Beitrags zur Erfüllung der Ziele und evtl. auftretender negativer Seiteneffekte verglichen und bewertet werden, um eine Entscheidung nach rationalen Gesichtspunkten treffen zu können (siehe zur rationalen Prozeduralität Abschnitt 2.1, S. 22). Gerade wenn die Entscheidung von einer Vielzahl unterschiedlicher Entscheidungsfaktoren abhängt, ist ein systematischer Vergleich der Lösungsansätze erforderlich, um Unsicherheiten und die daraus resultierenden Risiken zu reduzieren. Die Erfahrungen aus der Praxis zeigen, dass z. B. der Vergleich von fünf Alternativen im Hinblick auf fünf Ziele, die eine unterschiedliche Gewichtung besitzen, ohne methodische Unterstützung ein sehr komplexes Verfahren ist, da 25 unterschiedliche Kombinationen überprüft werden müssen. Können zudem einzelne Eigenschaften der Lösungsansätze nicht mit Sicherheit ermittelt werden, z. B. eine Überschreitung der Budgetgrenze mit einer Wahrscheinlichkeit von 40 %, führen unsystematische und subjektive Vergleiche sowie Bewertungen zu hohen Risiken für die Entscheidung.

Um die alternativen Lösungsansätze bzw. die Alternativen der Architekturentscheidung auch bei mehreren Zielen sowie bei Unsicherheiten systematisch zu vergleichen und zu bewerten, existiert in der Entscheidungstheorie ein systematisches Bewertungsverfahren [EiWe03]. Das Verfahren umfasst im Groben die folgenden drei Schritte:

- *Schritt 1:* Normalisierung der Eigenschaftswerte
- *Schritt 2:* Gewichtung nach Relevanz
- *Schritt 3:* Berücksichtigung wahrscheinlicher und unwahrscheinlicher Varianten

(1) Schritt 1: Um die unterschiedlich skalierten Werte zusammenzufassen und zu einem Gesamtwert je Lösungsansatz addieren zu können, erfolgt eine Normalisierung der Eigenschaftswerte der Lösungsansätze. Die Normalisierung ist vor allem dann erforderlich, wenn die Eigenschaftswerte mit relativen und absoluten Werten beschrieben werden. So wird der Zielerreichungsgrad meist in Prozentwerten angegeben, z. B. eine Verbesserung der Wartbarkeit zu 75 %; der Implementierungsaufwand wird hingegen meist in Form von Personentagen, Stunden oder auch als monetärer Wert dargestellt. Diese verschiedenartig skalierten Werte werden im ersten Schritt auf ein Intervall zwischen null und eins normalisiert und drücken damit den Grad der Zielerreichung aus; dies wird auch als Präferenz des Entscheidungsträgers bezeichnet. Die Eigenschaften eines Lösungsansatzes, die den Zielen am stärksten entsprechen, werden mit einem Wert nahe bei eins normalisiert. Die Eigenschaften, die nur einen geringen Beitrag zur Zielerreichung darstellen, werden mit einem Wert nahe bei null normalisiert.

Die Art und Weise der Normalisierung erfolgt anhand einer Wertfunktion. Die Aufstellung einer Wertfunktion kann vom Entscheidungsträger entweder individuell oder anhand eines speziellen Verfahrens vorgenommen werden. Die beiden bekanntesten sind die 'Methode gleicher Wertdifferenzen' und das 'Direct-Ratio-Verfahren' [Fish67] (und [EiWe03]). Die Normalisierung anhand einer Wertfunktion ist in Abb. 2.4 dargestellt. Betrachtet werden die Eigenschaften (mit dem Index *r*) der Lösungsansätze *a* und *b*. Die Eigenschaften eines Lösungsansatzes haben einen höheren Wert (Value *v* der Eigenschaften a_r oder b_r), je stärker der Grad der Zielerreichung ist. Die alternativen Lösungsansätze *a* und *b* sind Elemente der Menge von Lösungsansätzen (sog. Alternativenmenge) *A*.

$$v(a_r) \geq v(b_r) \Leftrightarrow a \succ b, \qquad a, b \in A$$

Abb. 2.4: Normalisierung der Eigenschaftswerte

(2) Schritt 2: Sofern die Ziele eine unterschiedliche Gewichtung bzw. Priorisierung besitzen, erfolgt im zweiten Schritt die Gewichtung der verschiedenen Eigenschaften der Lösungsansätze. Dabei werden die normalisierten Eigenschaftswerte mit Gewichtungsfaktoren multipliziert. Für den Entscheidungsträger kann z. B. eine Verbesserung der Wartbarkeit eine höhere Priorität als ein möglichst geringer Implementierungsaufwand besitzen. Der Gewichtungsfaktor für die Qualitätsverbesserung ist in diesem Fall größer als der für den Implementierungsaufwand. In der Summe müssen alle Gewichtungsfaktoren den Wert eins ergeben.

Im Anschluss an die Gewichtung kann der Wert der Lösungsansätze ermittelt werden. Dies wird im Rahmen der Entscheidungstheorie als Bewertung unter Sicherheit verstanden; in Abb. 2.5 ist die entsprechende Bewertungsfunktion abgebildet. Diese Bewertungsfunktion wird in der Entscheidungstheorie als 'Additives-Modell' bezeichnet [EiWe03]. Es wird der Wert (Value) eines Lösungsansatzes *a* ermittelt, repräsentiert durch $v(a)$. Zur Berechnung werden die einzelnen, normalisierten Eigenschaften $v(a_r)$ mit dem Gewichtungsfaktor w_r multipliziert und addiert. Die Eigenschaften sind indiziert mit *r* im Intervall von eins bis *m*.

$$v(a)=\sum_{r=1}^{m} w_r v(a_r) \quad .$$

Abb. 2.5: Bewertung unter Sicherheit

(3) Schritt 3: Treten einzelne Eigenschaften nur mit einer bestimmten Wahrscheinlichkeit ein, ist der Erwartungswert zu berechnen. Zur Berechnung des Erwartungswertes werden die Werte der Eigenschaften jeder Variante mit den entsprechenden Wahrscheinlichkeiten multipliziert und zu einer Summe addiert. Ein Beispiel ist ein Lösungsansatz mit einem normalen Implementierungsaufwand (30 Personentage – PT) und einem erhöhten Aufwand (40 PT) bei einer Terminüberschreitung mit einer Wahrscheinlichkeit 40 %. Für den Normalfall werden die 30 PT mit 0.6, für den Fall der Terminüberschreitung werden die 40 PT mit 0.4 multipliziert und addiert. Das Ergebnis ist der Erwartungswert für die Eigenschaft Implementierungsaufwand des Lösungsansatzes. In Abb. 2.6 ist die Formel für die Ermittlung eines Erwartungswertes (Expected Value – EV) dargestellt. Die normalisierten Eigenschaften des Lösungsansatzes *a*, indiziert durch *r* im Intervall von eins bis *m*, werden mit den entsprechenden Gewichtungsfaktoren w_r multipliziert. Zusätzlich werden die verschiedenen Varianten, charakterisiert durch *i* im Intervall von eins bis *n*, mit den zugehörigen Wahrscheinlichkeiten p_i multipliziert.

$$EV(a)=\sum_{i=1}^{n} p_i[\sum_{r=1}^{m} w_r v(a_{ir})] \quad .$$

Abb. 2.6: Bewertung unter Unsicherheit

Durch dieses Verfahren wird den Lösungsansätzen ein Gesamtwert zwischen null und eins zugeordnet. Auf dieser Basis erfolgt im Anschluss die Entscheidung; gemäß den Kriterien einer rationalen Entscheidungsfindung sollte sich der Entscheidungsträger für den Lösungsansatz mit dem höchsten Wert entscheiden.

Die Ausführungen zeigen, dass es unter Anwendung des Bewertungsverfahrens aus der Entscheidungstheorie möglich ist, die Lösungsansätze einer Architekturentscheidung auch bei mehreren Zielen sowie bei Unsicherheiten systematisch zu vergleichen und zu bewerten. Da das Verfahren zudem bei betriebswirtschaftlichen Entscheidungen, z. B. bei riskanten und komplexen Investitionsentscheidungen mit unsicheren Annahmen über Kosten und zukünftige Erlöse, bereits erfolgreich eingesetzt wird [Saat01], wird es auch für Architekturentscheidungen adaptiert und eingesetzt.

2.10 Methoden zur Visualisierung komplexer Entscheidungssituationen

Vor allem für die Dokumentation und Nachvollziehbarkeit einer Entscheidung ist eine vereinfachte Abbildung der komplexen Entscheidungssituation notwendig. Es existieren verschiedene Methoden, um die Entscheidungssituation bzw. die alternativen Lösungsansätze und deren unterschiedliche Eigenschaften zu visualisieren. Die folgenden vier Methoden werden in der Praxis häufig eingesetzt:

- *Entscheidungsbäume*
- *Netzpläne und Entscheidungsgraphen*
- *Aktivitätsdiagramm der UML*
- *Zustandsdiagramme*

Die vier Methoden werden in den folgenden Abschnitten dahingehend analysiert, inwieweit die visuelle Darstellung dazu beiträgt, die Unsicherheit und Komplexität der Entscheidungsfindung zu reduzieren. Ziel ist es, durch eine visuelle Darstellung der relevanten Lösungsansätze, deren Eigenschaften und weiterer Entscheidungsfaktoren einen direkten und systematischen Vergleich der Lösungsansätze zu ermöglichen.

(1) Entscheidungsbäume: Zur Visualisierung der Entscheidungssituation können Entscheidungsbäume verwendet werden [EiWe03] (oder [Eis+07]), welche im Bereich der Entscheidungstheorie häufig eingesetzt werden. Die alternativen Lösungsansätze und deren Eigenschaften werden in Form sich verzweigender Äste dargestellt. Am Ende ist abgebildet, in welchem Umfang die Lösungsansätze zur Zielerreichung beitragen. In Abb. 2.7 ist ein Beispiel für einen Entscheidungsbaum mit zwei Lösungsansätzen zur Lösung eines Qualitätsproblems dargestellt. Beide Lösungsansätze führen zu einem unterschiedlichen Aufwand bei der Implementierung und zu unterschiedlichen Folgerisiken. Aufgrund von Einflussfaktoren, z. B. Wechselwirkungen mit bereits implementierten Mustern oder Stilen, kann der Aufwand zur Implementierung der 'Architekturalternative 1' entweder 'gering' oder 'hoch' sein. Entscheidungsbäume sind ein ideales Mittel zur Abbildung des Verlaufs der Entscheidungsfindung von der Wurzel bis hin zur Zielerreichung; wegen der Baumstruktur können jedoch bestimmte Aspekte der Entscheidungssituation nicht abgebildet werden, u. a. Iterationen oder Wiederholungen.

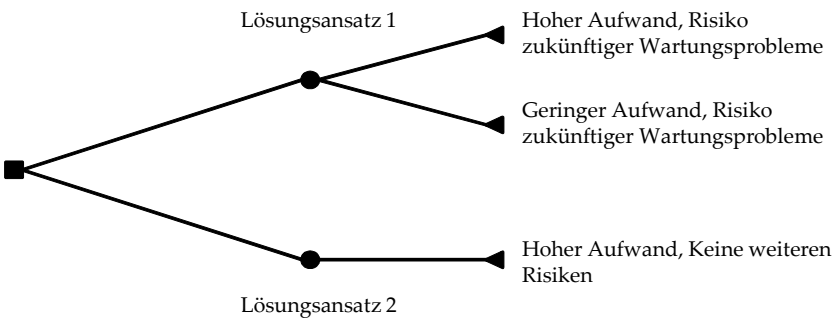


Abb. 2.7: Beispiel eines Entscheidungsbaumes mit zwei Lösungsansätzen

(2) Netzpläne oder Entscheidungsgraphen: Ein Netzplan ist eine auf Graphen basierende Darstellung von Abläufen und Abhängigkeiten [Zimm05]. Zwei häufig eingesetzte Methoden zur Erstellung und Auswertung von Netzplänen sind die PERT-Methode (Program Evaluation and Review Technique) und die Methode des kritischen Pfades (Critical-Path-Method – CPM). Bei beiden Methoden werden die Aktivitäten in Form eines Netzes aus Knoten und Kanten dargestellt. Für die einzelnen Aktivitäten werden Durchführungszeiten ermittelt, z. B. auf Basis optimistischer oder pessimistischer Schätzungen. Somit kann für einen Lösungsansatz detailliert ermittelt werden, welche Zeit die Implementierung benötigt. Bei CPM wird speziell der zeitkritische Pfad zur Implementierung ermittelt. Neben dem Aufwand zur Implementierung können jedoch keine weiteren Eigenschaften der Lösungsansätze abgebildet werden.

Eng verwandt mit den Netzplänen sind Entscheidungsgraphen, bei denen die einzelnen Bestandteile einer Entscheidung (Ziele, alternative Lösungsansätze, Zustände sowie Eigenschaften der Lösungsansätze) gemeinsam in einem Diagramm abgebildet werden ([EiWe03] und [Kru+06]). Im Gegensatz zu Entscheidungsbäumen ist das Ziel eines Entscheidungsgraphen nicht die Abbildung des Verlaufs der Entscheidungsfindung von der Wurzel bis hin zur Zielerreichung, sondern die Abbildung der gesamten Entscheidungssituation mit allen Annahmen, Bedingungen und Abhängigkeiten. Dadurch können konkrete Lösungsansätze und deren Eigenschaften nur indirekt abgeleitet werden. In Abb. 2.8 ist ein Beispiel dargestellt.

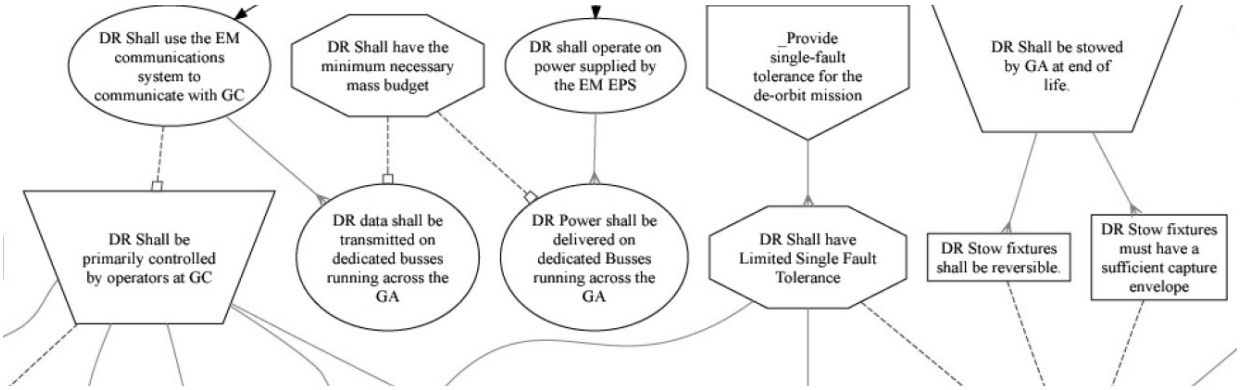


Abb. 2.8: Beispiel für Entscheidungsgraphen [Kru+06]

(3) Aktivitätsdiagramm der UML: Die dritte Methode zur Darstellung einer komplexen Entscheidungssituation sind die Aktivitätsdiagramme der UML. Die Aktivitätsdiagramme zählen zu den Verhaltensdiagrammen der UML, um einen Vorgang, z. B. den Prozess einer Architekturentwicklung oder -veränderung, zu beschreiben [Boo+06]. Modelliert werden Aktivitäten (abgerundete Vierecke), die mit Kontroll- und Datenflüssen verbunden sind. Neben Aktionen können Zustände im Aktivitätsdiagramm durch Vierecke dargestellt werden. Ein Aktivitätsdiagramm kann Ein- und Ausgangszustände umfassen, die bei Beginn und Ende der Aktivität eintreten. Da Rücksprünge und zyklische Abhängigkeiten abgebildet werden können, kann die Entscheidungssituation, speziell der Prozess zur Implementierung eines Lösungsansatzes, detailliert und realistisch abgebildet werden.

In Abb. 2.9 ist ein verkürztes Beispiel für ein Aktivitätsdiagramm einer Architekturentwicklung dargestellt. Der Eingangszustand ist eine 'geringe Wartbarkeit'. Durch eine Folge von Aktionen wird die Architektur verändert und weiterentwickelt, um die Architekturqualität zu erhöhen. Waren die Verbesserungen nicht ausreichend, sind 'zusätzliche Anpassungen' an der Architektur durchzuführen. Die Aktivitätenfolge terminiert, wenn die Wartbarkeit 'in ausreichendem Maße' verbessert ist. Der Ausgangszustand ist die gewünschte 'hohe Wartbarkeit'.

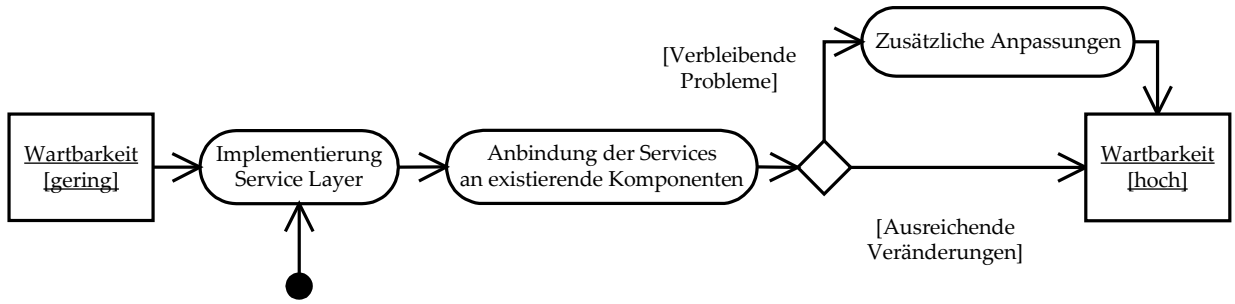


Abb. 2.9: Beispiel eines Aktivitätsdiagramms

(4) Zustandsdiagramme (State Machines): Ein weiteres Verfahren zur Visualisierung komplexer Entscheidungssituationen sind Zustandsdiagramme [Kru+06], wie beispielsweise UML-Zustandsdiagramme, State-Machines oder Statecharts. Dabei wird der Verlauf der Entscheidungsfindung als Abfolge von Zuständen abgebildet. Typische Zustände sind z. B. 'Entscheidung ist getroffen', 'gewünschte Zielerreichung wird erreicht' oder 'Ergänzungen und Korrekturen sind notwendig'. Ein vereinfachtes Beispiel für ein 'State-Chart' einer Entscheidung ist in Abb. 2.10 dargestellt. Das Beispiel umfasst einen Prozess mit sieben Zuständen zur Bewertung und Weiterentwicklung von Ideen. Mit Zustandsdiagrammen können zwar auch kom-

plexe Entscheidungssituationen abgebildet werden, jedoch sind die Lösungsansätze durch gemeinsame Zustandsübergänge und Rücksprünge zwischen den Zuständen eng verwoben. Dies behindert den klaren Vergleich alternativer Lösungsansätze anhand der Eigenschaften bzw. anhand der Zielerreichung.

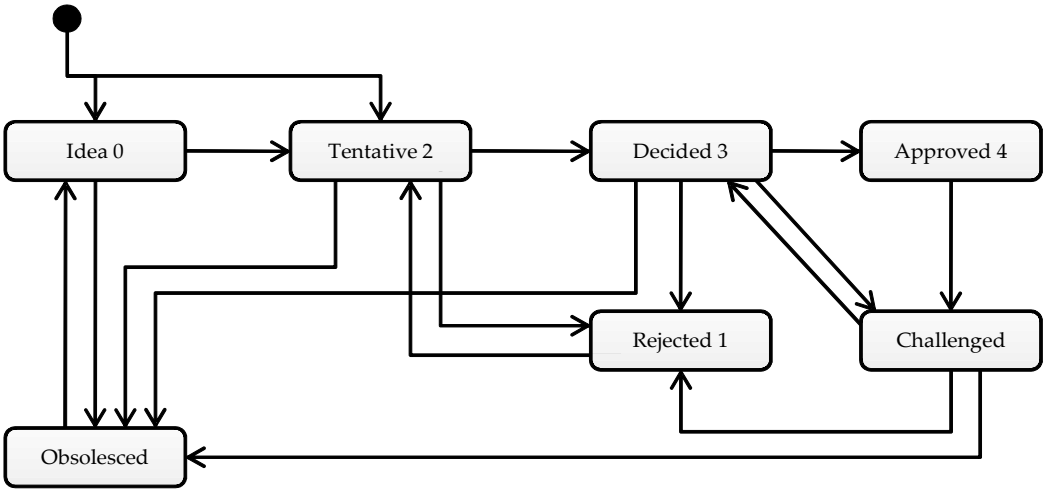


Abb. 2.10: Beispiel eines Entscheidungsgraphen [Kru+06]

Die Ausführungen zeigen, dass der Verlauf der Entscheidung – die Auswahl, Verfeinerung und Bewertung der verschiedenen Lösungsansätze mit den Varianten und Zuständen – am besten mit Entscheidungsbäumen abgebildet werden kann. Bei Entscheidungsbäumen wird klar dargestellt, welche Lösungsansätze zur Verfügung stehen und welche unterschiedlichen Eigenschaften (Grad der Zielerreichung, Implementierungsaufwand etc.) sie haben (siehe Abb. 2.7, S. 40). Gerade die Entscheidungsgraphen oder Zustandsdiagramme fokussieren im Gegensatz dazu auf die Abbildung der gesamten Entscheidungssituation mit allen Annahmen und Bedingungen (siehe z. B. Abb. 2.9, S.41). Daraus lassen sich jedoch nur indirekt konkrete Lösungsansätze ableiten. Zur Abbildung des Prozesses der Implementierung eines Lösungsansatzes eignen sich indes die Aktivitätsdiagramme der UML am besten, da im Gegensatz zu Netzplänen oder Entscheidungsbäumen u. a. Verzweigungen und Wiederholungen abgebildet werden können.

2.11 Fazit und Verfeinerung der Zielstellung

In den vorangegangenen Kapiteln wurde untersucht, welche Lücken das in der Entscheidungstheorie beschriebene generische Vorgehen speziell bei Architekturentscheidungen hat. So ist gerade zur Berücksichtigung der softwaretechnischen Details einer Architekturentscheidung eine Anpassung und Weiterentwicklung des generischen Vorgehens erforderlich. Um die identifizierten Lücken im Rahmen der Entwicklung des Entscheidungsprozesses schließen zu können, wurden verschiedene Methoden, Konzepte und Verfahren hinsichtlich ihres Beitrags zur Strukturierung der Ziele, der Vorauswahl der alternativen Lösungsansätze, der Ermittlung der Eigenschaften oder der Bewertung der Lösungsansätze untersucht. Auf dieser Grundlage erfolgt in den folgenden Abschnitten die Entwicklung des Entscheidungsprozesses, der speziell an die Spezifika von Architekturentscheidungen angepasst ist und dadurch einen wichtigen Beitrag zur Reduzierung von Risiken, Unsicherheiten und der Komplexität leistet, die im Zusammenhang mit der Entwicklung sowie der Veränderung von Softwarearchitekturen auftreten.

Im Rahmen der Analyse wurde deutlich, dass einige Methoden und Konzepte direkt in das generische Vorgehen zur Entscheidungsfindung integriert werden können. So existieren Methoden und Verfahren, um eine widerspruchsfreie Beschreibung des für die Entscheidung relevanten Teils der Architektur zu ermöglichen. Mit der ISO-Norm 9126 steht eine Möglichkeit zur Verfügung, die Qualitätsmerkmale einer Architektur einheitlich beschreiben zu können; zudem ist mit dem Klassifizierungsverfahren aus der Entscheidungstheorie eine widerspruchsfreie Strukturierung der Ziele mithilfe von Fundamen-

tal- und Instrumentalzielen möglich. Um alternative Lösungsansätze systematisch vergleichen und bewerten zu können, kann zudem das Bewertungsverfahren aus der Entscheidungstheorie angewendet werden.

Durch die Analyse der relevanten Methoden und Konzepte zur Unterstützung von Architekturentscheidungen wird jedoch deutlich, dass einige Methoden und Verfahren für die Unterstützung einer rationalen Entscheidungsfindung anzupassen und zu erweitern sind. Mit ATAM und ALMA stehen zwar Methoden zur validen Analyse von Architekturen mittels Szenarien zur Verfügung. Da sie jedoch ergänzende Aktivitäten umfassen (z. B. Diskussionen und Präsentationen), die oftmals keinen erkennbaren Nutzen bei Architekturentscheidungen erbringen, ist der Umfang der Analyseaktivitäten aus Aufwand-Nutzen-Erwägungen heraus auf ein Minimum zu reduzieren. Zudem fehlt eine systematische Unterstützung, um anhand der Analyseergebnisse und der Ziele die geeigneten Lösungsansätze auszuwählen, zu verfeinern und zu konkretisieren. Es ist daher einerseits ein systematisches Verfeinerungs- und Konkretisierungsverfahren zu entwickeln. Andererseits ist ein Katalog zu erarbeiten, in dem die wichtigsten Muster, Stile, Technologien und Softwareprodukte nach ihren Qualitätsmerkmalen gruppiert sind.

Auf der Grundlage der Ergebnisse der Analyse der relevanten Methoden und Konzepte zur Unterstützung von Architekturentscheidungen können die Ziele der Dissertation wie folgt verfeinert und ergänzt werden:

- *Reduzierung von Komplexität, Risiken und Unsicherheiten:* Das zentrale Ziel der Dissertation ist die Entwicklung eines systematischen und an rationalen Kriterien orientierten Entscheidungsprozesses, mit dem die Komplexität, die Risiken und die Unsicherheiten der Entscheidung reduziert werden können. Dafür ist das in der präskriptiven Entscheidungstheorie beschriebene generische Vorgehen zur Entscheidungsfindung um Methoden und Konzepte zu ergänzen, die eine detaillierte Betrachtung der softwaretechnischen Details von Architekturentscheidungen ermöglichen. Der Schwerpunkt der Dissertation liegt dabei auf der Entwicklung eines Verfahrens zur Auswahl, Verfeinerung und Konkretisierung geeigneter Lösungsansätze.
- *Ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis:* Bei der Anwendung des Entscheidungsprozesses ist die Frage zu beantworten, welcher Grad an Sicherheit bzw. reduziertem Risiko bei der Entscheidung im Hinblick auf den Entscheidungsgegenstand sowie den Nutzen notwendig und gerechtfertigt ist. Zur Einsetzbarkeit in der Praxis muss der mit der Entscheidungsfindung verbundene Aufwand in einem ökonomisch sinnvollen Verhältnis zum erwarteten Nutzen stehen. Es ist daher eine Notwendigkeit, dass der Detailgrad der zur Entscheidungsfindung erforderlichen Analysen und Bewertungen je nach Risiko und Kritikalität des Entscheidungsgegenstandes angepasst werden kann. Dabei muss jedoch sichergestellt sein, dass die Rationalitätsanforderungen (siehe Abschnitt 2.1, S. 22) eingehalten werden.
- *Dokumentation und Nachvollziehbarkeit:* Die Entscheidung für oder gegen einen alternativen Lösungsansatz basiert auf einem ausgewogenen und systematischen Vergleich der Eigenschaften der Lösungsansätze; die ermittelten Eigenschaftswerte sind jedoch nur Schätzwerte, die zum Zeitpunkt der Entscheidungsfindung und somit vor der eigentlichen Implementierung der Lösungsansätze ermittelt werden. Durch eine geeignete Dokumentation der einzelnen Phasen der Entscheidungsfindung soll es einerseits ermöglicht werden, die Gründe für oder gegen eine Entscheidung nachvollziehen zu können. Zum anderen sollen die dokumentierten Erwartungswerte (Grad der Zielerreichung, Implementierungsaufwand, Seiteneffekte etc.) anhand der tatsächlich eingetretenen Konsequenzen im Anschluss an die Implementierung validiert werden können. Auch bei der Dokumentation des Entscheidungsverlaufs ist ein ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis einzuhalten.
- *Eignung für architekturorientiertes Refactoring:* In der Praxis werden Architekturentscheidungen häufig erst dann getroffen, wenn die Weiterentwicklung eines Softwaresystems aufgrund von Architekturproblemen gefährdet ist, z. B. bei massiven Wartungsproblemen. Der Entscheidungsprozess soll speziell bei der Veränderung existierender Architekturen im Rahmen von architekturorientiertem Refactoring eingesetzt werden können. Im Rahmen des architekturorientierten Refactorings sind jedoch besondere Anforderungen zu beachten. So sind u. a. sowohl funktionale Einschränkungen zu vermeiden als auch komplexe und weit reichende Architekturveränderungen nicht in einem großen Schritt, sondern in mehreren kleineren Schritten durchzuführen.

Kapitel 3

Überblick über den entwickelten Entscheidungsprozess

3.1 Systematische Entscheidungsfindung in vier Phasen

Der entwickelte Entscheidungsprozess umfasst vier Phasen, die nacheinander durchlaufen werden; aufgrund der Vielzahl an gemeinsamen Schnittpunkten können die ersten beiden Phasen parallel durchgeführt werden. Werden jedoch Inkonsistenzen und Fehler erkannt, z. B. bei einer falschen Abbildung der Ziel-Mittel-Beziehungen, sind Rücksprünge zu den betreffenden Phasen und Tätigkeiten erforderlich. In Abb. 3.1 sind die Phasen des entwickelten Entscheidungsprozesses dargestellt. Mit grau hervorgehoben sind die Phasen zwei und drei; vor allem um diese Tätigkeiten wurde das generische Vorgehen aus der Entscheidungstheorie ergänzt und erweitert, um die softwaretechnischen Details von Architekturentscheidungen analysieren und bewerten zu können (siehe Abschnitt 2.1, S. 22).

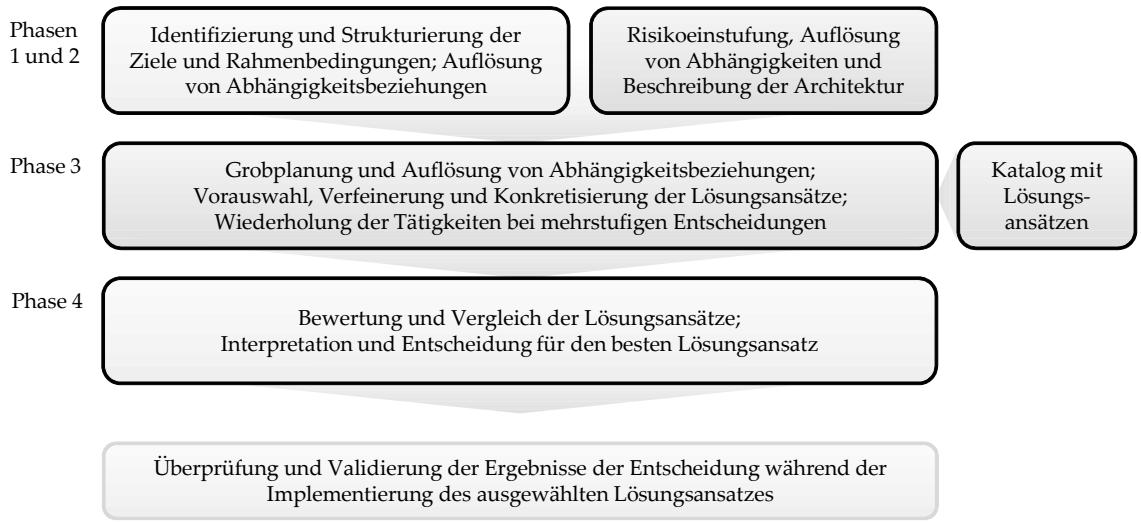


Abb. 3.1: Entwickelter Entscheidungsprozess für Architekturentscheidungen nach rationalen Gesichtspunkten

Die in Abb. 3.1 dargestellten vier Phasen des entwickelten Entscheidungsprozesses beinhalten die folgenden obligatorischen und optionalen Tätigkeiten. Während obligatorische Tätigkeiten in jedem Fall durchzuführen sind, da sie zur Einhaltung einer Entscheidungsfindung nach rationalen Gesichtspunkten erforderlich sind, kann der Entscheidungsträger bei optionalen Tätigkeiten auswählen. Bei risikoarmen und weniger komplexen Architekturentscheidungen sollten optionale Analyse- und Bewertungstätigkeiten, nur dann angewendet werden, wenn der Aufwand in einem ökonomisch sinnvollen Verhältnis zum daraus resultierenden Nutzen steht. Im darauf folgenden Abschnitt 3.3 (siehe S. 46ff.) werden die Kriterien im Detail erläutert, welche den Entscheidungsträger bei der Bestimmung des Aufwand-Nutzen-Verhältnisses von optionalen Tätigkeiten unterstützen. Eine Zusammenfassung der Tätigkeiten ist in folgender Liste aufgeführt:

- *Phase 1 – Ziele und Rahmenbedingungen:* In der ersten Phase werden die Ziele der Entscheidung identifiziert und auf der Grundlage eines Klassifikationsverfahrens aus der Entscheidungstheorie strukturiert, um Widersprüche aufzulösen, ungenaue Ziele zu verfeinern und die Priorisierung der Ziele zu erleichtern. Sind die Ziele noch nicht in ausreichender Form bekannt, erfolgt eine vereinfachte szenariobasierte Architekturanalyse. Darüber hinaus werden die organisatori-

schen (z. B. Termine, Budget) und technischen Rahmenbedingungen (z. B. einzuhaltende Standards) anhand einer Checkliste ermittelt.

- *Phase 2 – Risikoeinstufung der Entscheidung und Architekturbeschreibung:* Neben den Zielen sind die relevanten Architekturteile zu beschreiben, da diese die Basis für die Vorauswahl und die Bewertung der Lösungsansätze sind. Zu den relevanten Architekturteilen zählen jene Teile, die zur Beseitigung der Schwachstellen sowie zur Erreichung der Ziele verändert und angepasst werden müssen. Darüber hinaus sind die Teile der Architektur zu beschreiben, die die Entscheidungsfindung beeinflussen. Das sind u. a. Infrastrukturaspekte, z. B. notwendige Interpreter, oder bereits implementierte Muster und Stile, die die Eigenschaften – und damit die Eignung – der alternativen Lösungsansätze zur Architekturveränderung beeinflussen können. In der zweiten Phase erfolgt zudem die Einstufung der Entscheidung in drei Risikoklassen. Entsprechend der Risikoeinstufung kann der Aufwand zur Entscheidung bei risikoarmen Entscheidungen reduziert werden, da nicht alle optionalen Analyse- und Bewertungstätigkeiten erforderlich sind (siehe zu den Kriterien Abschnitt 3.3, S. 46ff.).
- *Phase 3 – Vorauswahl und Konkretisierung:* In der dritten Phase werden aus der Menge an möglichen Lösungsansätzen diejenigen auswählt, die unter den gegebenen Rahmenbedingungen zur Erreichung der Ziele geeignet sind. Sofern die Lösungsansätze zur Erreichung der Ziele noch nicht bekannt sind, steht ein Katalog mit Mustern, Stilen, Technologien und Softwareprodukten zur Verfügung. Zur Ermittlung der Eigenschaften der Lösungsansätze sowie der Wechselwirkungen und Seiteneffekte erfolgt im Anschluss an die Auswahl sowohl eine Verfeinerung als auch eine Konkretisierung der Lösungsansätze. Dabei finden außerdem eine detaillierte Betrachtung und Berücksichtigung der Abhängigkeitsbeziehungen statt, die u. a. zwischen den Architekturkomponenten und auch zwischen den einzelnen Schritten zur Architekturentwicklung oder -veränderung auftreten können.
- *Phase 4 – Bewertung und Entscheidung:* Sofern der Vergleich der alternativen Lösungsansätze aufgrund von Unsicherheiten über die erwarteten Eigenschaften oder der Vielzahl an Eigenschaftswerten zu komplex ist, erfolgt in der vierten Phase ein systematischer Vergleich sowie die abschließende Bewertung der Lösungsansätze.

3.2 Eingangsvoraussetzungen und Rollenverteilung

Für die Anwendung des Entscheidungsprozesses müssen bestimmte Eingangsvoraussetzungen erfüllt werden, da ansonsten das ökonomisch sinnvolle Verhältnis zwischen Aufwand und Nutzen nicht mehr gewährleistet ist. So muss sichergestellt sein, *wer* der Träger der Entscheidung ist und dass der Entscheidungsträger in allen Phasen der Entscheidungsfindung *beteiligt oder zumindest informiert* ist. In der Praxis ist häufig zu beobachten, dass Entscheidungen durch ein Gremium nur vorbereitet werden und erst zum Abschluss der Entscheidungsfindung dem eigentlichen Entscheidungsträger die Lösungsansätze und deren Eigenschaften präsentiert werden. Dies führt meist dazu, dass einzelne oder alle Phasen des Entscheidungsprozesses aufgrund von Rückfragen oder Korrekturen durch den Entscheidungsträger erneut durchlaufen werden müssen. Zur Vermeidung solcher Iterationen ist es wichtig, dass der Entscheidungsträger in allen Phasen des Entscheidungsprozesses involviert ist, um u. a. eine aussagekräftige Gewichtung der Ziele durchzuführen oder die richtigen Szenarien für die Architekturanalysen auszuwählen.

Für die Funktion des Entscheidungsträgers kommen verschiedene Personengruppen in Betracht:

- *Kunde:* Architekturentscheidungen orientieren sich maßgeblich an den Anforderungen des Kunden. Er ist derjenige, der von der hohen Architekturqualität direkt profitiert, da z. B. Anpassungen an ein verändertes Marktumfeld zeitnah umgesetzt werden können. Der Kunde ist dann als Entscheidungsträger aufzufassen, wenn ihm die technischen Details des Softwaresystems in ausreichendem Maße bekannt sind, bzw. der Kunde ohnehin die Architekturvorgaben und Architekturrichtlinien verantwortet.
- *Entwickler:* Der Entwickler ist die Person, die über detaillierte Kenntnisse über das Softwaresystem, die Architektur und die Daten- und Kontrollflüsse verfügt. Jedoch besitzt ein Entwickler allein meist nur eine eingeschränkte Entschei-

dungsbefugnis. Er kann z. B. zwar über den Aufbau seiner Klassen und Komponenten entscheiden, nicht aber über die grundsätzliche Architekturentwicklung. Aufgrund seiner Fachkenntnisse ist der Entwickler jedoch in unterstützender Funktion am Entscheidungsprozess beteiligt.

- *Architekt:* Bei Architekturentscheidungen sind die Einzelmeinungen der Entwickler zusammenzufassen und mit den Anforderungen des Kunden abzugleichen. Dem Architekten oder einer anderen Person mit entsprechender Entscheidungsbefugnis, z. B. dem Projektleiter, obliegt diese koordinierende und verwaltende Funktion. Vor allem dann, wenn dem Kunden die technischen Details des Softwaresystems nicht in ausreichendem Maße bekannt sind, ist er als Träger der Entscheidung aufzufassen.

Zudem muss die zu treffende Architekturentscheidung eine gewisse Grundkomplexität aufweisen. Zwar sollte der Entscheidungsprozess auch bei risikoarmen Entscheidungen angewendet werden, um durch die Dokumentation des Verlaufs der Entscheidungsfindung die Nachvollziehbarkeit zu gewährleisten. Jedoch ist der Aufwand für die Analysen und Bewertungen bei trivialen Entscheidungen nicht gerechtfertigt. Der Entscheidungsprozess sollte erst dann eingesetzt werden, wenn mehrere Ziele gleichzeitig zu erfüllen sind und nur unvollständige Informationen über die Architektur und die erforderlichen Schritte zur Weiterentwicklung und Veränderung der Architektur vorliegen. Nur unter diesen Voraussetzungen ist der Aufwand zur Anwendung des Entscheidungsprozesses aufgrund der Reduzierung von Komplexität, Risiken und Unsicherheit gerechtfertigt.

Eine weitere Eingangsvoraussetzung, damit der Entscheidungsprozess in einem ökonomisch sinnvollen Aufwand-Nutzen-Verhältnis durchgeführt werden kann, ist ein möglichst kleiner Kreis von unterstützenden Teilnehmern. Generell sollte nur eine kleine Gruppe von Entwicklern (oder Architekten) den Prozess der Entscheidungsfindung unterstützen, da mit der Anzahl der Personen, die an der Entscheidung beteiligt sind, die kombinatorische Komplexität ansteigt. In der Praxis werden Architekturentscheidungen häufig deswegen nicht getroffen, da in großen Gruppendiskussionen nur selten eine Einigung gefunden oder Kompromisse geschlossen werden können.

3.3 Aufwandsreduzierung durch Anpassbarkeit der Entscheidungsfindung

Ein ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis ist eines der wichtigsten Ziele, die der Entwicklung des Entscheidungsprozesses (siehe Abb. 3.1, S. 44) zugrunde liegen. Der Entscheidungsprozess umfasst zwar grundsätzlich eine Vielzahl von Analysen und Bewertungen, um eine Entscheidungsfindung nach rationalen Gesichtspunkten zu ermöglichen und damit die Risiken, die Komplexität und die Unsicherheiten reduzieren zu können. Bei Entscheidungen, die jedoch ohnehin ein geringes Risiko und eine geringe Komplexität aufweisen, kann der Analyse-, Design- und Bewertungsumfang reduziert werden. Das Ziel ist es, ein ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis bei der Entscheidungsfindung zu ermöglichen, indem um nur die Tätigkeiten gezielt ausgewählt werden, die aufgrund von Risiken oder Komplexität erforderlich sind (siehe Abb. 3.2). Nicht alle Architekturentscheidungen sind derart komplex und risikoreich, dass z. B. eine formale Architekturbeschreibung oder die Erstellung von Architekturprototypen notwendig ist, um die Risiken, die Komplexität und die Unsicherheiten bei der Entscheidungsfindung auf ein Maß zu senken, bei dem die Gefahr einer Fehlentscheidung deutlich reduziert ist. Trotzdem ist auch bei einer risikoarmen Architekturentscheidung, die eine geringe Komplexität sowie ein geringes Maß an Unsicherheiten aufweist, ein systematisches und sachlich-begründetes Vorgehen erforderlich und gerechtfertigt, da allein die Zusatzkosten sowie der Zeitverzug aufgrund einer Fehlentscheidung in der Regel höher sind und schwerer wiegen, als der Aufwand für die Durchführung des Entscheidungsprozesses im Minimalumfang. Zudem dadurch die vor allem in der Praxis sehr wichtige Dokumentation und Nachvollziehbarkeit der Entscheidungsfindung gegeben.

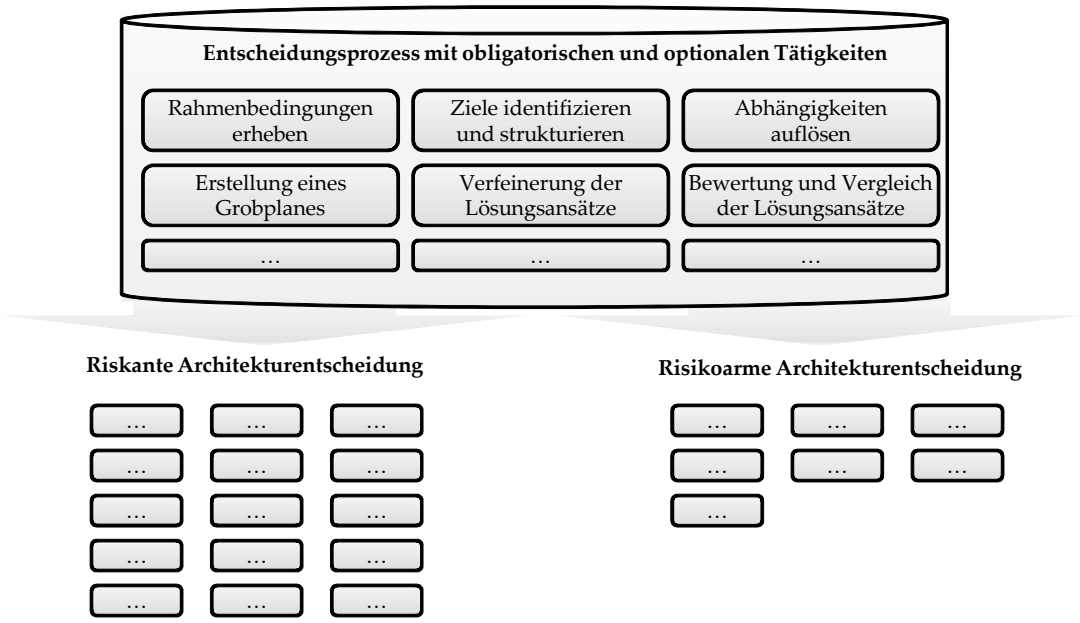


Abb. 3.2: Anpassbarkeit des Entscheidungsprozesses an verschiedene Entscheidungssituationen

Um die Analyse-, Design- und Bewertungstätigkeiten im Hinblick auf das Aufwand-Nutzen-Verhältnis bewerten und auswählen zu können, wurden die Tätigkeiten des Entscheidungsprozesses in obligatorische und optionale Analyse-, Design- und Bewertungstätigkeiten unterteilt. Die obligatorischen Tätigkeiten stellen das Mindestmaß dar, um eine systematische Entscheidung nach rationalen Gesichtspunkten zu ermöglichen und die Rationalitätskriterien (siehe Abschnitt 2.1, S. 22) zu erfüllen. So sind bei jeder Entscheidung die Entscheidungsgrundlagen *Ziele, Rahmenbedingungen* und *Architekturmodell* zumindest zu identifizieren und grob zu beschreiben. Da die alternativen Lösungsansätze in der Praxis häufig bereits aus den Zielen ableitbar sind, ist abschließend nur noch eine Verfeinerung der Maßnahmen zur Architekturentwicklung erforderlich, um die Eigenschaften der alternativen Lösungsansätze (Implementierungsaufwand, Grad der Zielerreichung) bestimmen zu können. Auf dieser Grundlage kann trotz des reduzierten Analyse- und Bewertungsumfangs eine Entscheidung nach rationalen Gesichtspunkten getroffen werden. Zudem wird die Dokumentation und Nachvollziehbarkeit der Entscheidungsfindung gewährleistet.

In Tab. 3.1 (siehe S. 48) sind ausschließlich jene obligatorische Analyse-, Design- und Bewertungstätigkeiten aufgezeigt, die die Minimalform des Entscheidungsprozesses darstellen, um eine Entscheidung nach rationalen Gesichtspunkten treffen zu können. Erste Anwendungen des Entscheidungsprozesses zeigen, dass nur die Durchführung der obligatorischen Tätigkeiten im Mittel einen Aufwand in Höhe von zwei bis drei Personentage darstellen.

Nummer	**Bezeichnung der obligatorischen Tätigkeiten**
Phase 1	
5.1.1	**Identifizierung der Ziele auf der Grundlage von Schwachstellen und Problembereichen**
5.1.3	**Erhebung der organisatorischen und technischen Rahmenbedingungen**
Phase 2	
5.2.1	**Einstufung des Risikos der Entscheidung**
5.2.2	**Beschreibung der Architektur**
Phase 3	
5.3.4	**Konkretisierung und Anpassung der Lösungsansätze**
5.3.4.1	**Aufstellung und Verfeinerung der Maßnahmenfolge**

Nummer	Bezeichnung der obligatorischen Tätigkeiten
5.3.4.2	**Auswertung der Abhängigkeitsbeziehungen**
5.3.4.3	**Ermittlung der Eigenschaften der Lösungsansätze**
Phase 4	
5.4.2	**Treffen der Entscheidung**

Tab. 3.1: Mindestumfang des Entscheidungsprozesses

Je nach Komplexität, Unsicherheit und Risiko der Entscheidung können ergänzende, optionale Tätigkeiten erforderlich werden. Im Rahmen der folgenden Abschnitte und insbesondere in der Tab. 10.2 (siehe Anhang 10.2, S. 171ff.) werden Kriterien aufgeführt und erläutert, anhand derer die Notwendigkeit der optionalen Analyse-, Design- und Bewertungstätigkeiten ermittelt werden kann. Ein Ausschnitt aus dieser Tabelle ist in Tab. 3.2 (siehe S. 48) wiedergegeben. So ist zwar bei jeder Architekturentscheidung ein Architekturmodell erforderlich, um die Schwachstellen im Detail ermitteln zu können und die möglichen Lösungsansätze zur Architekturveränderung zu vergleichen und bewerten zu können. Aus Aufwandsgründen ist hingegen der Einsatz formaler oder semi-formaler Modellierungs- und Architekturbeschreibungssprache nur bei besonders kritischen Architekturentscheidungen erforderlich und gerechtfertigt. Die Kriterien stellen Richtwerte und Empfehlungen dar, die aufgrund der Vielzahl an unterschiedlichen Entscheidungssituationen nicht verbindlich sein können.

Da der Entscheidungsträger anhand der in Tab. 10.2 aufgeführten Kriterien die zur Entscheidungsfindung nach rationalen Gesichtspunkten erforderlichen Analyse-, Design- und Bewertungstätigkeiten gezielt identifizieren und auswählen kann, ist eine Anpassbarkeit des Entscheidungsprozesses gegeben.

Kapitel	Bezeichnung der Tätigkeit	Obligatorisch / Optional	Begründung für obligatorische Tätigkeiten Kriterien für optionale Tätigkeiten
5.2.2	Beschreibung der Architektur	Obligatorisch	Eine Beschreibung der Architektur ist bei jeder Architekturentscheidung erforderlich, da auf dieser Grundlage die Auswahl, Verfeinerung und Konkretisierung der Lösungsansätze erfolgt. Je nach Risikoklasse der Entscheidung kann es erforderlich werden, den Betrachtungsraum des Modells zu erweitern, indem Um- und Drittsysteme mit in die Architekturbeschreibung aufgenommen werden.
	Einsatz formaler oder semiformaler Architektur- oder Modellbeschreibungssprachen sowie Sichten	Optional (bei 5.2.3)	Ausschließlich riskante und sehr riskante Entscheidungen rechtfertigen den Einsatz semi-formaler/formaler Beschreibungssprachen oder den Einsatz von Sichten, da dadurch die Komplexität reduziert und die Qualität der Architekturanalysen erhöht werden kann.

Tab. 3.2: Auszug aus der Übersicht der obligatorischen und optionalen Tätigkeiten im Entscheidungsprozess

3.4 Ein- und mehrstufige Architekturentscheidungen

Die Erfahrungen aus der Praxis zeigen, dass umfangreiche und weit reichende Weiterentwicklungen oder Veränderungen der Architektur oftmals nicht in einem Schritt, sondern nur in mehreren Schritten durchgeführt werden können, da die kombinatorische Komplexität zu hoch ist. Derartige Weiterentwicklungen oder Veränderungen werden oftmals als 'strategisch' bezeichnet, wobei Strategien vor allem im Bereich der Unternehmensführung, in der Politik und im Militär thematisiert werden [Horv06]. Im Rahmen der vorliegenden Dissertation werden Architekturstrategien als ein visionäres, planvolles und strukturiertes Vorgehen zur Erreichung eines oder mehrerer übergeordneter Ziele verstanden (siehe u. a. [Pos+04]). Aufgrund der kombinatorischen Komplexität strategischer Architekturentwicklungen oder -veränderungen und der begrenzten Zeit für detaillierte Analysen steigt der Grad an Unsicherheit, da nicht alle Abhängigkeitsbeziehungen, Seiteneffekte und Risiken erkannt werden können. Um die Komplexität und damit die Risiken sowie die Unsicherheiten zu reduzie-

ren, ist eine Aufteilung der Weiterentwicklungen oder Veränderungen in mehrere Maßnahmenfolgen erforderlich. Da für jede einzelne Maßnahmenfolge alternative Lösungsansätze existieren können, wird die Entscheidungsfindung auf mehrere Stufen aufgeteilt.

Zur Durchführung einer mehrstufigen Architekturentscheidung ist zunächst eine Grobplanung aufzustellen, um die erforderlichen Entwicklungs- oder Veränderungsschritte sowie die Anzahl, den Umfang und die Teilziele der Maßnahmenfolgen grob bestimmen zu können. Im Anschluss sind für jede Maßnahmenfolge geeignete alternative Lösungsansätze zu identifizieren, auszuwählen und zu verfeinern. In diesem Zusammenhang sind außerdem die Eigenschaften der einzelnen Lösungsansätze zu ermitteln, u. a. hinsichtlich des Implementierungsaufwandes oder des Grades der Erreichung der Teilziele, um die Lösungsansätze jeder Maßnahmenfolge miteinander vergleichen zu können. Im Gegensatz zu einstufigen Architekturentscheidungen, bei denen nur eine Entscheidung für eine einzelne Maßnahmenfolge zu treffen ist, sind bei mehrstufigen Entscheidungen die Abhängigkeitsbeziehungen zwischen den Maßnahmenfolgen zu berücksichtigen (siehe Absschitt 4, S. 50ff.). Die Erfahrungen aus der Praxis zeigen, dass bei aufeinander aufbauenden Maßnahmenfolgen eine bestimmte Reihenfolge einzuhalten ist oder dass bestimmte Kombinationen von Maßnahmenfolgen nicht möglich sind.

Ein Beispiel für eine Architekturstrategie ist die Verbesserung der Sicherheit eines Softwaresystems für das Internet-Banking, wobei in verschiedenen Stufen die Sicherheit schrittweise erhöht wird: (1) *Einsatz einer demilitarisierten Zone zum Schutz vor Hackern*, (2) *Einsatz von Signaturen und Fingerprints* und (3) *Ablösung des Pin/Tan-Verfahrens durch ein sichereres Verfahren.* Dabei sind nacheinander für drei Maßnahmenfolgen Einzelentscheidungen zu treffen:

(1) Welche Firewall-Technologie kann eingesetzt werden?

(2) Wo in der Architektur kann das Trustcenter zur Verifizierung der Signaturen und zur Generierung der Fingerprints implementiert werden?

(3) Ist der Einsatz des HBCI-Protokolls gerechtfertigt oder ist ein anderes Verfahren einzusetzen?

Zur Erreichung der strategischen und weit reichenden Ziele sind zunächst alternative Lösungsansätze für die einzelnen Maßnahmenfolgen auszuwählen, zu verfeinern und deren Eigenschaften zu bestimmen (siehe für ein aussagekräftigeres Beispiel zu diesem Thema Abschnitt 6.2, S. 118ff.); zudem sind die Abhängigkeitsbeziehungen zwischen den drei Maßnahmenfolgen zu berücksichtigen. In Abb. 3.3 ist ein Entscheidungsbaum dargestellt, in dem die verschiedenen Kombinationen zwischen den Lösungsansätzen für die Maßnahmenfolgen *Firewall-*, *Trustcenter-* und *Zugriffsverfahren* dargestellt sind. In der ersten Entscheidungsstufe erfolgt die Entscheidung hinsichtlich der 'Firewall', in der zweiten Stufe hinsichtlich der Implementierung des 'Trustcenters' und in der dritten Stufe hinsichtlich des 'Zugriffsverfahrens'. Jede Kombination von Lösungsansätzen führt dabei zu acht unterschiedlichen Varianten der Verbesserung des Sicherheitsniveaus und des Implementierungsaufwandes. Es ist daher jene Kombination von Lösungsalternativen zu ermitteln, welche die höchste Verbesserung der Sicherheit ermöglicht und dabei einen vergleichsweise geringen Implementierungsaufwand erfordert. Das kann in Abb. 3.3 beispielsweise die Kombination 'Firewall A', 'Trustcenter A' und 'Zugriffsverfahren B' sein.

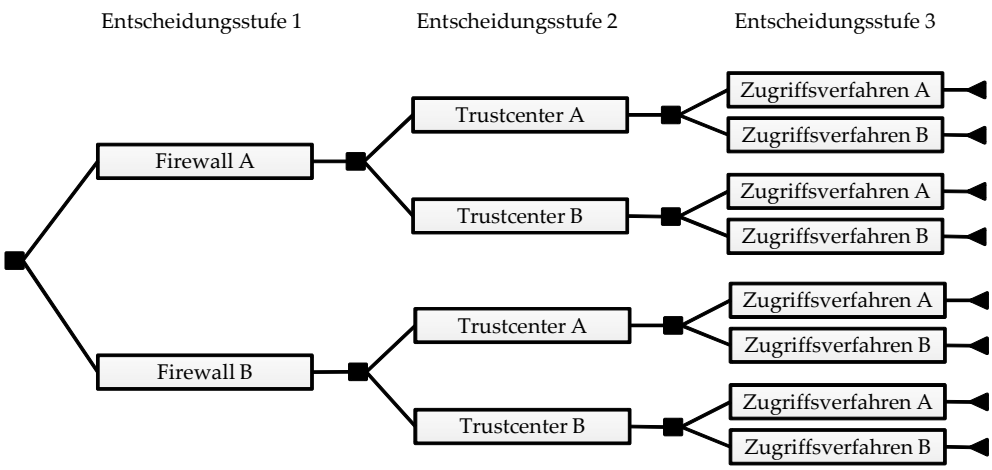


Abb. 3.3: Prototypisches Beispiel für eine mehrstufige Architekturentscheidung

Kapitel 4

Identifizierung, Beschreibung und Berücksichtigung von Abhängigkeitsbeziehungen

Bei Architekturentscheidungen existieren vielfältige Abhängigkeitsbeziehungen, die im Rahmen der Entscheidungsfindung systematisch berücksichtigt werden müssen (siehe Abb. 4.1), um Risiken und Unsicherheiten zu reduzieren. Solche Abhängigkeitsbeziehungen existieren u. a. zwischen Architekturkomponenten, die sich gegenseitig benutzen, oder zwischen konkurrierenden Zielen und Rahmenbedingungen. Neben den bereits bekannten Abhängigkeitsbeziehungen zeigen die Erfahrungen aus der Praxis, dass gerade die unbekannten oder nur in Teilen bekannten Beziehungen ein großes Risiko für die erfolgreiche Umsetzung einer Architekturentscheidung darstellen. Werden Abhängigkeiten und Konflikte erst während der Implementierung erkannt, sind Korrekturen und Nacharbeiten erforderlich, die häufig zu einem erheblichen Mehraufwand, zeitlichen Verzögerungen und überzogenen Budgets führen. Die Erkennung, Beschreibung und Berücksichtigung der Abhängigkeitsbeziehungen bei der Entscheidungsfindung sind daher eine wichtige Grundlage, um die Komplexität, die Unsicherheiten und die Risiken bei Architekturentscheidungen zu reduzieren. Wird beispielsweise erst nach der Ergänzung einer Datenbankschnittstelle erkannt, dass die neue Schnittstelle mit wichtigen Komponenten des bereits existierenden Softwaresystems inkompatibel ist, sind entsprechende Korrekturen und Nacharbeiten erforderlich. Schlimmstenfalls sind die Ergänzungen an der Datenbankschnittstelle zurückzunehmen oder es ist eine zweite Datenbankschnittstelle zu implementieren.

Eine Übersicht über die Vielfalt der Abhängigkeitsbeziehungen bei Architekturentscheidungen ist in Abb. 4.1 dargestellt. So haben u. a. Ziel-Mittel-Beziehungen eine direkte Auswirkung auf die Auswahl und Verfeinerung alternativer Lösungsansätze. Insbesondere bei der Veränderung eines bereits existierenden Softwaresystems sind bei der Entscheidungsfindung Abhängigkeitsbeziehungen zwischen Architekturkomponenten zu berücksichtigen; darüber kann bei der Implementierung eines Lösungsansatzes eine bestimmte Reihenfolge der Schritte zur Architekturveränderung einzuhalten sein.

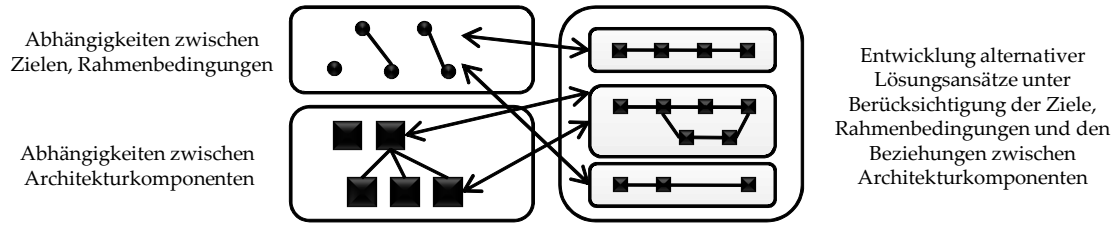


Abb. 4.1: Abhängigkeitsbeziehungen bei der Entscheidungsfindung

4.1 Reduzierung der Vielfalt von Abhängigkeitsbeziehungen

Da sich die aus der Literatur und Praxis bekannten Abhängigkeitsbeziehungen häufig überschneiden und trotz unterschiedlicher Bezeichnung identische Zusammenhänge beschreiben (siehe Abschnitt 2.8, S. 37f.), ist eine Fokussierung auf eindeutige Beziehungsarten erforderlich. Durch die Fokussierung und die damit verbundene Beschränkung der Vielfalt werden die kombinatorische Komplexität und der Aufwand zur Identifizierung, Beschreibung und Berücksichtigung der Abhängigkeitsbeziehungen reduziert; zudem werden durch die Fokussierung auf Beziehungsarten die Klarheit sowie die

Verständlichkeit der Abhängigkeitsbeziehungen erhöht. Die Fokussierung ist daher eine wichtige Voraussetzung, um die Unsicherheiten, die Komplexität und die Risiken bei Architekturentscheidungen senken zu können.

Auf der Grundlage der in Abschnitt 2.8 dargestellten Abhängigkeitsbeziehungen von Kruchten und Zimmermann ([Kru+06] und [Zim+08]) wurden die in Tab. 4.1 dargestellten Beziehungsarten entwickelt.[4] Die Beziehungsarten beschreiben dabei Beziehungen zwischen den drei Teilbereichen einer Architekturentscheidung, auf die auch der entwickelte Entscheidungsprozess mit seinen Analyse-, Design- und Bewertungsaktivitäten fokussiert: *die Architekturkomponenten, die Maßnahmen zur Architekturentwicklung/-veränderung und die Ziele/Rahmenbedingungen*. Die in Tab. 4.1 dargestellten Beziehungsarten sind nicht für eine spezifische Anwendungsdomäne angepasst, sondern müssen je nach Entscheidungssituation sowie Einsatzgebiet weiterentwickelt, verfeinert oder ergänzt werden.

Beziehungen zwischen	***Architekturkomponenten***	***Maßnahmen zur Architekturentwicklung/-veränderung***	***Zielen, Rahmenbedingungen***
Architekturkomponenten	(1.1.1) Erfordert und/oder Benutzt von (1.1.2) Aggregiert mit (1.1.3) Spezialisiert zu	(1.2.1) Verändert durch	(1.3.1) Stehen in Konflikt mit (1.3.2) Positive Auswirkung
Maßnahmen zur Architekturentwicklung/ -veränderung	*Siehe 1.2.1*	(2.2.1) Erzwingen oder Ermöglichen (2.2.2) Verhindern	(2.3.1) Stehen in Konflikt mit
Zielen, Rahmenbedingungen	*Siehe 1.3.1 und 1.3.2*	*Siehe 2.3.1*	(3.3.1) Sich ergänzen (3.3.2) Miteinander konkurrieren

Tab. 4.1: Übersicht über die grundsätzlichen Beziehungsarten zwischen den Elementen einer Architekturentscheidung

Beziehungen zwischen Architekturkomponenten

Die Abhängigkeitsbeziehungen, die einen großen Einfluss auf die Architekturentscheidung über die Weiterentwicklung oder Veränderung der Architektur besitzen, existieren zwischen den einzelnen Architekturkomponenten. Zu den Architekturkomponenten zählen im Rahmen der vorliegenden Dissertation einerseits die modellierten Komponenten eines Architekturmodells und andererseits die Softwarekomponenten als funktionale Instanzen der modellierten Komponenten. Die Erfahrungen aus der Praxis zeigen, dass die Abhängigkeiten zwischen den Architekturkomponenten meist nur lückenhaft bekannt sind, da eine vollständige und konsistente Architekturdokumentation zur Ermittlung der Abhängigkeitsbeziehungen aufgrund des gerade bei Großprojekten üblichen hohen Termindrucks und der Vielzahl an Veränderungen nicht zur Verfügung steht. Ohne eine vollständige und konsistente Architekturdokumentation ist auch die Aussagekraft werkzeugbasierter Analysen begrenzt, da zur Identifizierung von Abhängigkeiten und speziell zur Erkennung von Konflikten und Widersprüchen eine logische Referenzarchitektur modelliert werden muss. Als Werkzeuge für die Erkennung von Abhängigkeitsbeziehungen stehen z. B. Sonar-J, Java-Messplatz oder Sontograph zur Verfügung [BeBe07].

Da in der Praxis eine vollständige und konsistente Architekturbeschreibung meist nicht zur Verfügung steht, wird im Rahmen der Entscheidungsfindung ein Ausschnitt eines existierenden Architekturmodells oder ein Architekturprototyp untersucht; die Untersuchung wird mittels szenariobasierter Architekturanalysen durchgeführt. Die Untersuchung orientiert sich an den folgenden Beziehungsarten, die je nach Entscheidungssituation sowie Einsatzgebiet weiterentwickelt, verfeinert oder ergänzt werden müssen:

- *(1.1.1) Erfordert von – Required by und/oder Benutzt von – Used by:* Aufgrund technischer oder fachlicher Abhängigkeiten können zwei oder mehrere Architekturkomponenten einander 'erfordern und/oder sich gegenseitig benutzen'. Derartige Beziehungen sind zwischen allen Architekturkomponenten anzutreffen; eine Ausnahme stellen isolierte Kompo-

4 Die Richtung der Beziehungen ist dabei zeilenweise von links nach rechts dargestellt.

nenten dar, die zur Laufzeit nicht verwendet werden. Bei der Auswahl und Konkretisierung der alternativen Lösungsansätze ist zu beachten, dass Veränderungen von zentralen Komponenten, die von vielen anderen Komponenten benutzt werden, sehr riskant sind und tief greifende Untersuchungen der abhängigen Komponenten benötigen. Zudem ist aufgrund der verschiedenen 'erfordert oder benutzt von'-Beziehungen eine bestimmte Reihenfolge bei der Durchführung der Architekturentwicklung oder -veränderung einzuhalten.

- *(1.1.2) Aggregiert mit – Composed with:* Zwei oder mehrere Architekturkomponenten können miteinander 'aggregiert' sein, wenn die Komponenten in einer Teil-Ganzes-Beziehung oder in einer Parent-Child-Beziehung stehen. Zwei typische Beispiele für diese Beziehungsart sind Komponenten eines Subsystems oder eine Implementierung des Kompositum-Entwurfsmusters (siehe Abschnitt 7.1.3, S. 141f.). Im Rahmen der Auswahl sowie Konkretisierung der alternativen Lösungsansätze ist zu beachten, dass 'aggregierte' Komponenten in enger Verbindung stehen, einen hohen Kopplungsgrad aufweisen und sich oftmals wechselseitig benutzen.
- *(1.1.3) Spezialisiert zu – Specialized to:* Architekturkomponenten können durch eine oder mehrere Komponenten funktional ergänzt oder erweitert werden. Die generischen Komponenten, die oftmals nur eine grundlegende Funktionalität besitzen, werden dabei durch andere Komponente 'spezialisiert'. Ein typisches Beispiel ist eine Komponente, welche grundlegende Funktionen zur Abfrage von Datenbanken beinhaltet. Für spezielle Anwendungsfälle können spezialisierte Komponenten existieren, die ergänzende Funktionalität umfassen, z. B. für Simulationen oder mathematische Auswertungen. Da die spezialisierten Komponenten auf die grundlegende Funktionalität der generischen Komponenten zugreifen, muss bei Veränderungen an den generischen Komponenten überprüft werden, welche Anpassungen an den spezialisierten Komponenten erforderlich sind. Im Gegensatz dazu können die 'spezialisierten' Komponenten unabhängig von den generischen Komponenten verändert werden.

Beziehungen zwischen Architekturkomponenten und Maßnahmen

- *(1.2.1) Verändert durch – Changed by:* Die Architekturkomponenten oder Komponentenstrukturen, welche die Ursache von Schwachstellen und Architekturproblemen darstellen, werden durch eine oder mehrere Maßnahmen 'verändert', z. B. eine funktionale Ergänzung einer Schnittstelle zu einer Datenbank oder zu einem Drittsystem. Vor allem wenn Architekturkomponenten mehrfach verändert werden, ist diese Beziehungsart bei der Bestimmung der Reihenfolge von Veränderungen oder Weiterentwicklungen zu beachten. Gerade bei mehrfachen Veränderungen einer Komponente, z. B. im Rahmen eines architekturorientierten Refactorings, kann eine bestimmte Reihenfolge der Veränderungs- oder Weiterentwicklungsschritte zu beachten sein. Die Beziehungen zwischen Maßnahmen müssen daher vor allem bei der Reihenfolgeplanung bei mehrstufigen Architekturentscheidungen im Detail berücksichtigt werden.

Beziehungen zwischen Architekturkomponenten und Zielen sowie Rahmenbedingungen

- *(1.3.1) Stehen in Konflikt mit – In conflict with:* Der Fokus der Entscheidungsfindung liegt auf jenen Architekturkomponenten oder Komponentenstrukturen, die Ursache für die Architekturprobleme sind und 'in Konflikt' mit den Zielen oder Rahmenbedingungen stehen. Jene Architekturkomponenten oder Komponentenstrukturen bedürfen einer Veränderung oder Weiterentwicklung und sind daher der Ausgangspunkt für die Analysen sowie die Auswahl und Konkretisierung der alternativen Lösungsansätze (siehe Beziehung 1.2.1 im vorangegangenen Absatz 'verändert durch'). So kann z. B. eine proprietäre Schnittstelle zu einem Drittsystem 'in Konflikt' mit dem Ziel Verbesserung der Wartbarkeit stehen; daher ist eine Veränderung der Schnittstelle zur Erfüllung des Ziels erforderlich. Darüber hinaus kann die proprietäre Schnittstelle 'in Konflikt' mit dem Einsatz von Testautomaten, den Architekturrichtlinien oder weiteren organisatorischen und technischen Rahmenbedingungen stehen.
- *(1.3.2) Positive Auswirkung – Positive impact:* Im Gegensatz zu Architekturkomponenten, die 'in Konflikt' mit den Zielen und Rahmenbedingungen stehen, existieren Komponenten, die sich auf die Ziele oder Rahmenbedingungen bereits 'positiv auswirken'. Bei der Auswahl und Konkretisierung der alternativen Lösungsansätze ist zu vermeiden, dass eine Veränderung dieser Komponenten die positive Auswirkung auf die Ziele oder Rahmenbedingungen aufhebt, wenn z. B. die Entfernung eines Proxy die ohnehin schlechte Performance zusätzlich belastet.

Beziehungen zwischen Maßnahmen

- *(2.2.1) Erzwingen – Constrains und Ermöglichen – Facilitates:* Der Einsatz einer Maßnahme zur Architekturentwicklung oder -veränderung kann durch eine oder mehrere Maßnahmen 'erzwungen sein'. Diese Beziehungsart ist vor allem bei der Reihenfolgeplanung der Maßnahmen zu beachten, vor allem bei mehrstufigen Architekturentscheidungen. Zu dieser Beziehungsart zählt auch die 'ermöglicht'-Beziehung, wenn die Durchführung einer Maßnahme erst nach Abschluss einer anderen Maßnahme möglich ist. So 'ermöglicht' beispielsweise die Nutzung der Programmiersprache Java den Einsatz des 'Hibernate'-Frameworks für einen effizienten Datenbankzugriff oder ergänzender J2EE-Technologien.
- *(2.2.2) Verhindern – Prevents:* Der Einsatz einer oder mehrerer Maßnahmen zur Architekturentwicklung oder -veränderung kann 'verhindern', dass andere Maßnahmen durchgeführt werden können. Sofern bei der Konkretisierung und Verfeinerung der Lösungsansätze diese Beziehungsform erkannt wird, sind entsprechende Anpassungen und Veränderungen an der Maßnahmenfolge des betreffenden Lösungsansatzes erforderlich. Diese Beziehungsart tritt z. B. dann auf, wenn eine im Rahmen der Architekturveränderung entfernte Funktionalität einer Schnittstelle durch eine andere Komponente noch benötigt wird. Entweder ist die betreffende Komponente zu verändern oder es ist die Schnittstelle erneut anzupassen.

Beziehungen zwischen Maßnahmen und Zielen sowie Rahmenbedingungen

- *(2.3.1) Stehen in Konflikt mit – In conflict with:* Bestimmte Maßnahmen können 'in Konflikt mit' den Zielen und vor allem 'in Konflikt mit' den Rahmenbedingungen stehen. Werden bei der Konkretisierung und der Verfeinerung derartige Beziehungen erkannt, ist eine Anpassung der betreffenden Lösungsansätze erforderlich. Ist beispielsweise der Einsatz von OpenSource-Software nicht möglich, da die Rahmenbedingungen den Einsatz von Software von lizenzierten Herstellern vorschreiben, muss der Lösungsansatz entsprechend angepasst werden.

Beziehungen zwischen Zielen und Rahmenbedingungen

- *(3.3.1) Sich ergänzend – Supplementing:* Einzelne Ziele und Rahmenbedingungen können 'sich ergänzen' oder stehen in Ziel-Mittel-Beziehungen zueinander. Diese Beziehungen treten einerseits typischerweise zwischen den Zielen einer Architekturentscheidung auf und werden bei der Verfeinerung und Strukturierung der Ziele erkannt. So führen z. B. Verbesserungen der Analysier- und Testbarkeit häufig zu einer verbesserten Wartbarkeit. Andererseits sind diese Beziehungen bei Rahmenbedingungen zu berücksichtigen, die vom Entscheidungsträger beeinflusst werden können. So können Budgetvorgaben oftmals nicht vom Entscheidungsträger beeinflusst werden, ein großes Budget kann jedoch den Einsatz einer besonders geeigneten Werkzeugumgebung ermöglichen. Sich 'ergänzende' Beziehungen zwischen den Zielen und Rahmenbedingungen sind eine wichtige Grundlage, um dem Entscheidungsträger eine systematische Auswahl der geeigneten Lösungsansätze zu ermöglichen.
- *(3.3.2) Miteinander konkurrieren – Competing:* Die Ziele sowie die Rahmenbedingungen können 'in Konkurrenz stehen' und sich gegenseitig ausschließen. Konkurrenzbeziehungen sind vor allem bei den Rahmenbedingungen zu beachten, da nahezu alle Bedingungen um die knappen Ressourcen Zeit, Budget und Mitarbeiter konkurrieren. Im Falle der Rahmenbedingungen kann beispielsweise aufgrund eines zu geringen Budgets die gewünschte Werkzeugumgebung oder Test-Automatisierung nicht eingesetzt werden. Analog dazu können Ziele miteinander in Konkurrenz stehen und sich gegenseitig ausschließen, wenn bei Erfüllung eines Ziels ein anderes Ziel nicht mehr im erforderlichen Umfang erfüllt werden kann. Im Rahmen der Strukturierung der Ziele und Rahmenbedingungen sind 'konkurrierende'-Beziehungen zu identifizieren und aufzulösen, um Konflikte und Seiteneffekte bei der späteren Implementierung eines Lösungsansatzes zu vermeiden.

4.2 Identifizierung, Beschreibung und Berücksichtigung der Abhängigkeitsbeziehungen

Die einzelnen optionalen und obligatorischen Tätigkeiten zur Identifizierung, Beschreibung und Berücksichtigung der Abhängigkeitsbeziehungen bei der Entscheidungsfindung werden in Kapitel 5 (siehe S. 56ff.) im Detail beschrieben. Der entwickelte Entscheidungsprozess (siehe Abb. 3.1, S. 44) umfasst neben den Tätigkeiten zur Identifizierung und Berücksichtigung der Abhängigkeitsbeziehungen jedoch eine Vielzahl weiterer Analyse- und Bewertungstätigkeiten. Daher ist es notwendig, die Tätigkeiten zur Identifizierung, Beschreibung und Berücksichtigung der Abhängigkeitsbeziehungen klar herauszustellen.

Phase 1 – Abhängigkeitsbeziehungen zwischen Zielen und Rahmenbedingungen

Zur Erkennung und Strukturierung 'konkurrierender' oder 'sich ergänzender' Ziele wird in der ersten Phase ein Ziel-Wirkungsmodell erstellt. Das Vorgehen hierzu entstammt der Entscheidungstheorie (siehe Abschnitte 2.3 und 5.1.2, S. 28 und S. 57ff.). Dabei werden die Ziele nach Instrumental- und Fundamentalzielen klassifiziert und in eine baumartige Ziel-Mittel-Struktur eingeordnet. An der Spitze stehen die Fundamentalziele, welche durch die verschiedenen Instrumentalziele verfeinert werden. Ein Beispiel für ein Ziel-Wirkungsmodell ist in Abb. 4.2 dargestellt.

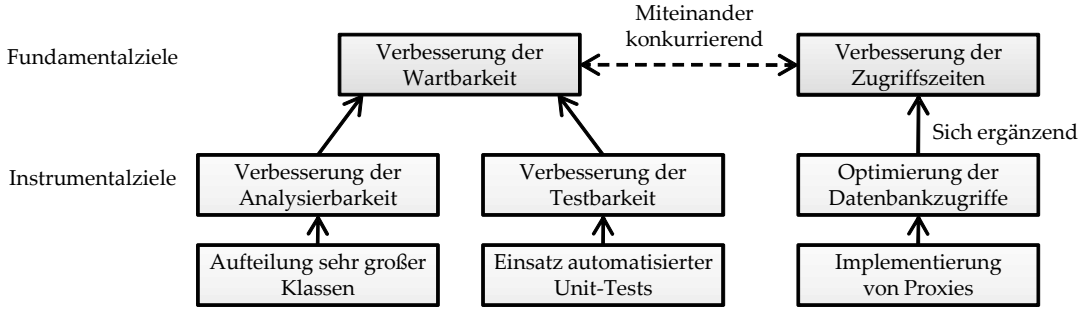


Abb. 4.2: Beispiel eines Ziel-Wirkungsmodells

Um die Rahmenbedingungen, z. B. die Höhe des Budgets oder die technischen Bedingungen der Implementierungsplattform, trotz der verschiedenen Abhängigkeitsbeziehungen systematisch identifizieren zu können, steht eine Checkliste zur Verfügung (siehe Abschnitt 5.1.3, S. 59ff.). Solch eine tabellarische Checkliste umfasst einerseits die Rahmenbedingungen in gruppierter Form und andererseits die zugehörigen absoluten Werte oder Wertebereiche. Mit der Abarbeitung der Checkliste werden die Rahmenbedingungen zunächst vollständig erhoben; 'sich ergänzende' Rahmenbedingungen werden dabei hervorgehoben. Die Auflösung von 'konkurrierenden'-Beziehungen erfolgt im zweiten Schritt durch eine Priorisierung der wichtigsten Rahmenbedingungen im Hinblick auf die Zielerreichung und die Anpassung der Rahmenbedingungen mit der geringeren Priorität. Dabei ist jedoch zu beachten, dass einige Rahmenbedingungen vom Kunden oder dem Management vorgegeben sind und nur bedingt durch den Entscheidungsträger angepasst werden können.

Phase 2 – Abhängigkeitsbeziehungen zwischen Architekturkomponenten

Um die häufig nur lückenhaft bekannten Beziehungen zwischen den Architekturkomponenten identifizieren zu können, erfolgt eine Beschreibung der relevanten Teile der Architektur mit anschließender Analyse anhand von Szenarien. Die Grundlage für die Beschreibung ist die Abgrenzung des Betrachtungsraums anhand der Ziele und mithilfe bereits bekannter Schwachstellen sowie Problembereiche. Sofern im Hinblick auf das Risiko der Entscheidung erforderlich, erfolgt im Anschluss die Beschreibung des relevanten Teils mittels formaler oder semi-formaler Beschreibungssprachen, z. B. der Unified-Modeling-Language (siehe Abschnitt 2.5, S. 31). Um die Abhängigkeitsbeziehungen zu ermitteln, ist ein Architekturreview mittels einer szenariobasierten Architekturanalyse durchzuführen. Als Grundlage kann hierfür die Architecture-Level-Modifiability-Analysis (ALMA) (siehe Abschnitt 2.4, S. 29) angewendet werden. Dabei werden Veränderungsszenarien am

Modell der Softwarearchitektur durchgespielt, um die Auswirkungen auf andere Komponenten erkennen zu können. Sofern der Betrachtungsraum zu groß ist, können zur Identifizierung der Schwachstellen zunächst Architekturmetriken ausgewertet werden, z. B. das Distanzmaß [Sim+99] oder Kopplungs-/Kohäsionsmetriken [She+01] (siehe Abschnitt 2.4, S. 28).

Phase 3 – Vorauswahl und Konkretisierung alternativer Lösungsansätze

In der dritten Phase erfolgt die Vorauswahl der alternativen Lösungsansätze (sog. Alternativen) unter Berücksichtigung der sich 'ergänzenden'- oder 'konkurrierenden'-Beziehungen zwischen Zielen und Rahmenbedingungen. Zur Vorauswahl der Lösungsansätze sind zunächst jene Best-Practices, Muster, Stile etc. auszuwählen, die zur Erfüllung der Ziele geeignet sind. Da gerade die Auswahl von Lösungsansätzen im Hinblick auf den Beitrag zur Verbesserung der Architekturqualität sehr komplex und aufwendig ist (siehe Abschnitt 2.3, S. 26), steht ein Katalog zur Verfügung (siehe Abschnitt 7, S. 138ff.). Um zu ermitteln, ob die identifizierten Beziehungen zwischen den Architekturkomponenten die Eignung der Lösungsansätze beeinflussen, sind entsprechende Untersuchungen durchzuführen, z. B. durch eine szenariobasierte Architekturanalyse (siehe vorangegangener Abschnitt).

Auf dieser Grundlage werden im Anschluss je alternativem Lösungsansatz die entsprechenden Maßnahmen identifiziert, die zur Veränderung oder Weiterentwicklung der Architektur erforderlich sind. Da sich Maßnahmen einander 'erzwingen' oder auch 'verhindern' können, ist ihre richtige Reihenfolge zu bestimmen. Sofern erforderlich kann hierfür wiederum eine szenariobasierte Architekturanalyse durchgeführt werden. Dabei wird die Architekturveränderung, bzw. die Implementierung der Maßnahmen, schrittweise durchgespielt, um den Umfang der Zielerreichung, die Einhaltung der Rahmenbedingungen sowie die Risiken und Inkompatibilität in der Maßnahmenfolge zu erkennen. Die Szenarioanalyse wird iterativ durchgeführt, um die Lösungsansätze weiter zu verfeinern und zu verbessern.

4.3 Aufwand-Nutzen-Verhältnis bei der Identifizierung von Abhängigkeiten

Die Ermittlung unbekannter Abhängigkeiten im Rahmen der Entscheidungsfindung muss in einem ökonomisch sinnvollen Verhältnis zum Analyseaufwand stehen. Es ist daher gerade bei risikoarmen Entscheidungen abzuwägen, welcher Analyseaufwand zur Identifizierung versteckter Abhängigkeiten gerechtfertigt ist. Die Erfahrungen aus der Praxis zeigen, dass nur bei riskanten oder sehr riskanten Entscheidungen mit hoher Komplexität die Identifizierung der Abhängigkeiten einen entsprechenden Nutzen in Form reduzierter Unsicherheiten erbringt. Eine Klassifizierung des Risikos der Entscheidung findet in der zweiten Phase des Entscheidungsprozesses statt (siehe Abschnitt 5.2.1, S. 62f.); anhand der Risikoklassen erfolgt eine Einordnung, welcher Analyseaufwand je Risikoklasse gerechtfertigt ist.

Kapitel 5

Vier Phasen des entwickelten Entscheidungsprozesses

Ein Überblick über die vier Phasen sowie die Analyse-, Design- und Bewertungstätigkeiten des entwickelten Entscheidungsprozesses wurde bereits in Abschnitt 3.1 (insbesondere in Abb. 3.1, siehe S. 44) dargestellt. In den folgenden Abschnitten wird im Detail erläutert, auf welche Art und Weise Architekturentscheidungen nach rationalen Gesichtspunkten getroffen werden können. Die praktische Anwendbarkeit des Entscheidungsprozesses zeigen zwei exemplarische Architekturentscheidungen in den Abschnitten 6.1 und 6.2 (siehe S. 90ff. und 118ff.).

5.1 Identifizierung und Strukturierung der Ziele und Rahmenbedingungen

5.1.1 Identifizierung der Ziele

Im ersten Schritt sind die funktionalen und die nicht-funktionalen Ziele (sog. Qualitätsziele) zu identifizieren, die für die Architekturentscheidung relevant sind. Sofern die Ziele noch nicht im Detail bekannt sind, sind geeignete Untersuchungen und Assessments durchzuführen. Während die Erfahrungen aus der Praxis zeigen, dass funktionale Ziele meist durch Zerlegung in ausreichendem Maße verfeinert werden können, sind gerade die Identifizierung sowie die Verfeinerung der Qualitätsziele eine große Herausforderung. So wird die Wartbarkeit auf Managementebene oftmals nur als Flexibilität und Erweiterbarkeit eines Softwaresystems verstanden, während die Entwickler eher auf die Aspekte Analysier- und Testbarkeit fokussieren. Zudem existiert speziell bei über Jahre gewachsenen Softwaresystemen eine Vielzahl nur teilweise bekannter 'sich ergänzender' oder 'miteinander konkurrierender' Zielbeziehungen und es ist daher unklar, welche Ziele überhaupt realistisch erfüllt werden können (siehe zu den Zielbeziehungen Abschnitt 4.1, S. 51ff.). Zur systematischen Identifizierung von Qualitätszielen steht trotz der Unklarheiten über die vielfältigen Zielbeziehungen u. a. der Quality-Attribute-Workshop (QAW) zur Verfügung [Bar+03]. Beim QAW werden die notwendigen Qualitätsziele auf der Grundlage von Unternehmenszielen ermittelt und durch Szenarien verfeinert. Eine Erweiterung des QAW stellt die Architecture-Tradeoff-Analysis-Method (ATAM) [Kaz+00] dar, mit der ebenfalls die Ziele auf der Grundlage von Unternehmenszielen identifiziert werden können. ATAM ermöglicht darüber hinaus auch eine detaillierte Architekturanalyse, um kritische Bereiche und Schwachstellen im Detail erkennen zu können. Bei der Durchführung einer szenariobasierten Architekturanalyse zur detaillierten Identifizierung der Ziele muss das Aufwand-Nutzen-Verhältnis beachtet werden; Szenarioanalysen und gerade die bei ATAM geforderte starke Einbeziehung der Stakeholder in die Analysen – maßgeblich die 'maintainers, operators, reusers, end users, testers, system administrators' – erfordern einen hohen Zeitaufwand und erbringen nur bei sehr komplexen sowie kritischen Architekturentscheidungen einen entsprechenden Nutzen.

Bei der Identifizierung der Ziele ist zu beachten, dass nur jene Ziele ausgewählt werden, die durch eine Weiterentwicklung oder Veränderung der Architektur beeinflusst werden können. Dies sind insbesondere die 'inneren Qualitätsmerkmale' eines Softwaresystems, wie z. B. die Zuverlässigkeit oder Wartbarkeit (siehe Abschnitt 2.3, S. 26f.). Sofern aus vorangegangenen Weiterentwicklungen und Veränderungen der Architektur Ziele existieren, die nicht erfüllt wurden und noch offen sind, müssen die Ziele auf Widersprüche und ihre Aktualität hin geprüft werden [Fowl05]. Gerade die Qualitätsziele können sich im Laufe der Zeit verändern und müssen entsprechend angepasst sowie eventuell ergänzt werden.

Zur Verbesserung der Klarheit und Verständlichkeit der Qualitätsziele kann die Beschreibung der Qualitätsziele auf der Grundlage der Qualitätsmerkmale der ISO-Norm 9126 erfolgen [IS9126]. Mit der ISO-Norm ist z. B. eindeutig und verständ-

lich definiert, was unter den Merkmalen Wartbarkeit oder Portabilität zu verstehen ist und welche Untermerkmale dafür existieren. Jedoch muss berücksichtigt werden, dass die ISO-Norm generische Qualitätsmerkmale umfasst, die je nach Entscheidungssituation zu ergänzen sind. So ist beispielsweise bei Architekturentscheidungen im Umfeld von Embedded-Systemen auch die Variabilität entscheidend, welche in der ISO-Norm nur am Rande thematisiert wird. Weitere Beispiele für Qualitätsmerkmale, die in der ISO-Norm 9126 nur am Rande betrachtet werden, sind die Flexibilität oder die Testbarkeit. Ein Ausschnitt der in der ISO-Norm 9126 enthaltenen Qualitätsmerkmale ist in Abb. 5.1 dargestellt. Der Ausschnitt fokussiert auf die 'inneren Qualitätsmerkmale' eines Softwaresystems, die entsprechend den Ausführungen in Abschnitt 2.3 (S. 26f.) maßgeblich von Weiterentwicklungen und Veränderungen an der Architektur beeinflusst werden können.

Abb. 5.1: Für Architekturentscheidungen relevante innere Qualitätsmerkmale der ISO-Norm 9126

5.1.2 Strukturierung der Ziele zur Erkennung und Auflösung von Abhängigkeitsbeziehungen

Im Anschluss an die Identifizierung der Ziele sind diese widerspruchsfrei zu strukturieren, um die Beziehungen zwischen den Zielen ('sich ergänzend' oder 'miteinander konkurrierend', siehe Abschnitt 4.1, S. 51ff.) erkennen, beschreiben und konkurrierende Zielbeziehungen auflösen zu können. Da mittels der Strukturierung die Beziehungen zwischen allen Zielen untersucht werden, erfolgen einerseits die Erkennung unsichtbarer Zielbeziehungen und andererseits die Reduzierung der kombinatorischen Komplexität. Zudem wird damit die Priorisierung der Ziele erleichtert. Die klare und widerspruchsfreie Strukturierung der Ziele, bei der Ziel-Mittel-Beziehungen klar und eindeutig beschrieben und Zielkonflikte aufgelöst sind, ist darüber hinaus eine Voraussetzung, um frühzeitig die *richtigen* Lösungsansätze zur Erfüllung der Ziele identifizieren zu können. Die Erfahrungen aus der Praxis zeigen, dass eine häufige Ursache von Fehlentscheidungen – neben Fehlern bei der Entscheidungsfindung selbst – widersprüchliche und falsch verstandene Ziele sowie Anforderungen sind. Widerspruchsfrei strukturierte Ziele sind darüber hinaus die Grundlage für die spätere Nachvollziehbarkeit und Validierbarkeit der Entscheidung.

Das Verfahren zur widerspruchsfreien Strukturierung der Ziele und zur Erstellung eines Ziel-Wirkungsmodells umfasst die folgenden vier Schritte:

- *Schritt 1:* Verfeinerung grober und ungenauer Ziele
- *Schritt 2:* Identifizierung sich ergänzender Ziel-Mittel-Beziehungen
- *Schritt 3:* Auflösung widersprüchlicher und konkurrierender Zielbeziehungen
- *Schritt 4:* Beschränkung auf die wichtigsten Ziele

(1) Schritt 1: Im ersten Schritt werden grobe und ungenau formulierte Ziele schrittweise durch konkretere Ziele verfeinert, da sie ansonsten nicht als Kriterien für die Architekturentscheidung eingesetzt werden können. Bei der Verfeinerung kann sich der Entscheidungsträger wiederum an der ISO-Norm 9126 orientieren, in der zu abstrakten Qualitätsmerkmalen detailliertere Untermerkmale aufgeführt sind (siehe Abb. 5.1). Ein in der Praxis häufiges Qualitätsziel ist die Verbesserung der Wartbarkeit, welche als solche zu abstrakt und ungenau ist, um alternative Lösungsansätze identifizieren und auswählen zu können. Sobald aber klar ist, bei welchen Systemteilen die Analysierbarkeit und Modifizierbarkeit (Untermerkmale von Wartbarkeit, siehe Abb. 5.1 auf S. 57) zu verbessern ist, können gezielter Lösungsansätze identifiziert werden. Da die erste Verwendung der Ziele als Auswahlkriterien erst in der dritten Phase des Entscheidungsprozesses, im Rahmen der Voraus-

wahl der Lösungsansätze, erfolgt (siehe S. 70ff.), sind die Ziele u. U. weiter zu verfeinern. Eine erneute Verfeinerung ist vor allem dann erforderlich, wenn keine oder eine sehr große Menge an alternativen Lösungsansätzen zur Erfüllung der Ziele zur Verfügung steht.

(2) Schritt 2: Im Anschluss an die Verfeinerung der Ziele sind die Ziel-Mittel-Beziehungen zwischen den Zielen zu identifizieren und zu beschreiben, um Widersprüche und Inkonsistenzen zu erkennen und aufzulösen sowie die Priorisierung der Ziele zu erleichtern. Zur Identifizierung der Wirkungszusammenhänge zwischen den Zielen erfolgt deren Klassifizierung in Instrumental- und Fundamentalziele, mittels einer aus der Entscheidungstheorie adaptierten Methode [Cle+01] (u. [EiWe03], siehe Abschnitt 2.3, S. 28). Instrumentalziele sind als Mittel zur Erreichung von Fundamentalzielen notwendig. Fundamentalziele sind die grundlegenden und für den Entscheidungsträger primären und maßgeblichen Ziele. Bei der Ziel-Mittel-Kette A → B → C sind die Ziele A und B Instrumentalziele für die Erreichung des Fundamentalziels C (siehe Abb. 5.2 mittig). Beispiele für Ziel-Wirkungsmodelle sind in Abb. 4.2 (S. 54), in Abb. 6.3 (S. 94) und in Abb. 6.20 (S. 121) dargestellt.

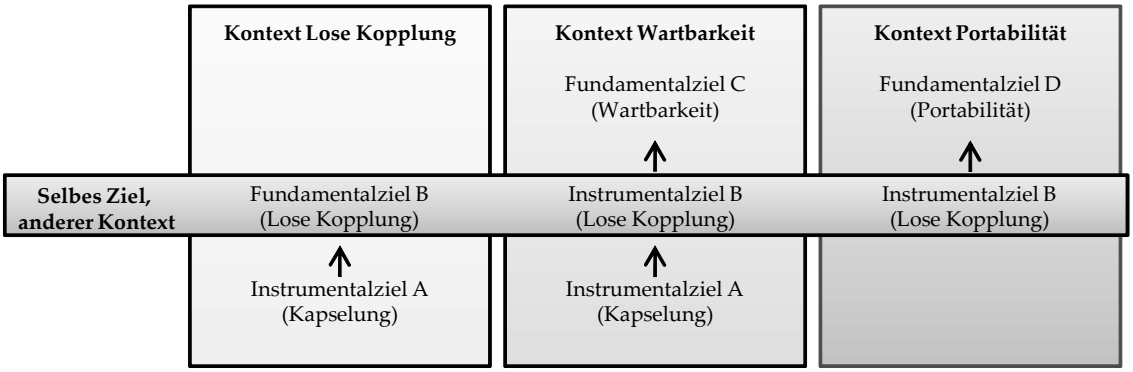


Abb. 5.2: Ziel-Mittel-Beziehungen in verschiedenen Kontexten

Eine zentrale Ursache von Widersprüchen bei der Interpretation von Ziel-Mittel-Beziehungen ist die Tatsache, dass die Unterscheidung zwischen Fundamental- und Instrumentalziel vom Betrachtungsraum bzw. vom Kontext abhängig ist. Ein Fundamental- oder Instrumentalziel kann in einem anderen Betrachtungsraum oder Kontext eine andere Bedeutung und eine andere Ziel-Mittel-Beziehung haben. Das in Abb. 5.2 dargestellte Ziel B, die lose Kopplung zwischen zwei Komponenten, kann in einem engen Betrachtungsraum ein Fundamentalziel sein. In einem größeren Betrachtungsraum, z. B. bei der Betrachtung der gesamten Komponentenstruktur, ist das Ziel B jedoch nur ein Instrumentalziel zur Verbesserung des höheren Zieles C, der Verbesserung der Wartbarkeit. Wird hingegen das Ziel B im Hinblick auf die Portabilität betrachtet, ist das Ziel wiederum nur ein Instrumentalziel (in Abb. 5.2 rechts). Die Klassifikationsmethode aus der Entscheidungstheorie ermöglicht die Betrachtung der Ziele in verschiedenen Betrachtungsräumen sowie Kontexten und leistet dadurch einen wichtigen Beitrag zur widerspruchsfreien Strukturierung der Ziele.

(3) Schritt 3: Nachdem die Ziel-Mittel-Beziehungen eindeutig beschrieben sind, erfolgt im dritten Schritt die Auflösung der konkurrierenden Zielbeziehungen. Ein Ziel X konkurriert mit einem anderen Ziel Z, wenn mit Erreichung von Ziel X das Ziel Z nicht in vollem Umfang erreicht werden kann (und umgekehrt). Zur Auflösung der Konkurrenzbeziehungen ist zunächst eine Priorisierung der Ziele notwendig. Im Anschluss wird das Ziel mit der niedrigeren Priorisierung entsprechend eingeschränkt. Ein häufiges Beispiel für konkurrierende Zielbeziehungen sind Verbesserungen an der Wartbarkeit bei gleichzeitiger Verbesserung der Performance. Wird z. B. die Wartbarkeit durch den Entscheidungsträger höher priorisiert, ist eine Einschränkung bei der Verbesserung der Performance notwendig. Dieser Zusammenhang ist in Abb. 5.3 dargestellt.

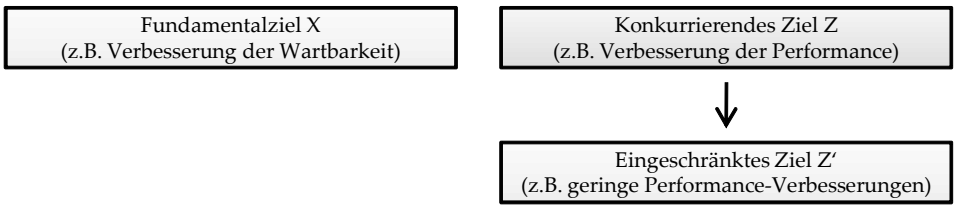


Abb. 5.3: Auflösung widersprüchlicher und konkurrierender Zielbeziehungen

(4) Schritt 4: Bei einer Vielzahl gleichzeitig zu erfüllender Ziele erfolgt im vierten Schritt eine Einschränkung auf die wichtigsten Ziele. Erste Anwendungen des Entscheidungsprozesses zeigten, dass mit mehr als zwei bis drei Fundamentalzielen die kombinatorische Komplexität stark ansteigt. Vor allem bei der Konkretisierung, Verfeinerung und Bewertung der Lösungsansätze führt die höhere kombinatorische Komplexität zu zusätzlichen Unsicherheiten sowie zu einem erhöhten Risiko einer Fehlentscheidung. So ist aufgrund der Vielzahl an Zielen trotz einer Priorisierung kaum noch eine Unterscheidung zwischen wichtigen und unwichtigen Zielen möglich; außerdem sind derart viele Einzelanalysen erforderlich, dass der dafür benötigte Aufwand in keinem ökonomisch sinnvollen Verhältnis zum Nutzen aus der systematischen Entscheidungsfindung steht. Sind beispielsweise fünf Fundamentalziele im Rahmen einer Architekturentscheidung zu betrachten, zu denen jeweils fünf Instrumentalziele gehören, sind für die Bewertung und den Vergleich *jedes* Lösungsansatzes 25 Kriterien zu überprüfen. Bei fünf Lösungsansätzen, die miteinander zu vergleichen sind, müssten 125 Kriterien systematisch überprüft werden. Eine hohe Zahl an Zielen ist in vielen Fällen vor allem ein Hinweis darauf, dass das Architekturproblem noch nicht vollständig verstanden wurde und zusätzliche Untersuchungen erforderlich sind.

Die Eingrenzung der Ziele ist speziell bei architekturorientiertem Refactoring ein erfolgskritischer Faktor. Grundsätzlich sind Veränderungen an der Architektur einem hohen Risiko ausgesetzt, da vor allem bei gewachsenen Legacy-Systemen große Unsicherheiten hinsichtlich der Abhängigkeitsbeziehungen existieren sowie jegliche Architekturveränderungen in einem hohen Test- und Analyseaufwand resultieren. Daher wird empfohlen, in kleinen Schritten vorzugehen, um bei einer Architekturveränderung nicht das gesamte Softwaresystem zu verändern, sondern nur einzelne, möglichst isolierte Teile davon [RoLi04].

5.1.3 Identifizierung und Erhebung organisatorischer sowie technischer Rahmenbedingungen

Jede Weiterentwicklung und Veränderung der Architektur unterliegt einer Vielzahl von organisatorischen und technologischen Rahmenbedingungen, die bei der Entscheidungsfindung berücksichtigt werden müssen. Zu den Rahmenbedingungen zählen z. B. zeitliche Vorgaben zu Rollout- und Releaseterminen, Budgetvorgaben oder die Einhaltung bestimmter Testabdeckungsgrade. In der Praxis existiert meist eine Vielzahl an unterschiedlichen Rahmenbedingungen, die 'sich ergänzen' oder 'miteinander konkurrieren' (siehe Abschnitt 4.1, S. 51ff.). So 'ermöglicht' beispielsweise ein ausreichend hohes Budget, dass die Qualität der Weiterentwicklung und die Veränderung der Architektur durch den Einsatz automatisierter Tests oder eines zuverlässigen Konfigurationsmanagements verbessert werden. Die Abhängigkeitsbeziehungen zwischen den Rahmenbedingungen müssen bei der Erhebung sowie Strukturierung beachtet und im Falle von Widersprüchen und Konflikten aufgelöst werden, um die kombinatorische Komplexität, die Risiken und die Unsicherheiten bei der Entscheidungsfindung reduzieren zu können (siehe zu den Rationalitätskriterien Abschnitt 2.1, S. 22).

Aufgrund der Vielzahl an unterschiedlichen Rahmenbedingungen stellen die Identifizierung und die Erhebung der für die Entscheidungsfindung relevanten Rahmenbedingungen ein Risiko dar. Da die Rahmenbedingungen neben den Zielen sehr wichtige Entscheidungsfaktoren für die Auswahl, den Vergleich und die Bewertung alternativer Lösungsansätze darstellen, können nicht berücksichtigte oder falsch erhobene Bedingungen eine erneute Durchführung des Entscheidungsprozesses erfordern oder – sofern der Fehler erst bei der Implementierung erkannt wird – zu einer Fehlentscheidung führen. Um dieses Risiko zu vermeiden, erfolgt die Identifizierung und Erhebung der relevanten Rahmenbedingungen strukturiert und systematisch anhand einer Checkliste. Die Checkliste umfasst dabei jene organisatorischen und technischen Rahmenbe-

dingungen, die bei Architekturentscheidungen typischerweise beachtet werden müssen (siehe u. a. [Pos+04] und [Hof+00]). Die Checkliste ist je nach Entscheidungssituation anzupassen und zu erweitern, um projektspezifische Bedingungen berücksichtigen zu können, z. B. bei der Steuerung räumlich verteilter Teams und Teilprojekte oder bei der Koordination paralleler Erweiterungsaktivitäten.

Bei der Erhebung der technischen Rahmenbedingungen ist zu unterscheiden, ob ein neues Softwaresystem entwickelt oder ein bereits existierendes System verändert wird. Bei der Neuentwicklung eines Softwaresystems sind die technischen Rahmenbedingungen maßgeblich auf der Grundlage der Spezifikation zu erheben. So ist beispielsweise im Pflichtenheft festgelegt, welche Hardwareressourcen zur Verfügung stehen werden und welches Betriebssystem einzusetzen ist; u. U. steht ein Prototyp zur Analyse zur Verfügung. Darüber hinaus sind Rahmenbedingungen aufgrund des Einsatzes von Drittsystemen, z. B. einzuhaltende Standards oder benötigte Netzwerkprotokolle, zu beachten. Sofern ein existierendes Softwaresystem verändert und restrukturiert wird, steht zur Erhebung eine breitere Informationsgrundlage zur Verfügung. Hierbei können der existierende Code, die eingesetzten Testfälle und – sofern verfügbar – die Architektur- und Entwicklungs-Dokumentation zur Erhebung der technischen Rahmenbedingungen herangezogen werden.

Organisatorische Rahmenbedingungen

Die organisatorischen Rahmenbedingungen umfassen alle nicht-technischen Bedingungen bezüglich der Terminvorgaben, der personellen Kapazität und des Budgets. Die Mehrheit der organisatorischen Bedingungen kann vom Entscheidungsträger nicht oder nur in geringem Umfang verändert werden. Daher haben die organisatorischen Bedingungen große Auswirkungen auf die technischen Bedingungen, die vom Entscheidungsträger viel stärker beeinflusst werden können bzw. zur Auflösung von Konflikten angepasst werden müssen (siehe Abschnitt 5.1.4 auf S. 61). Zu den organisatorischen Bedingungen zählen nach [Pos+04] und [Hof+00] vor allem die Folgenden:

- *Managementvorgaben:* Das sind einerseits strategische und politische Vorgaben, z. B. eine besondere Präferenz für offene Schnittstellen (im Gegensatz zu sog. Insellösungen), 'Make-or-Buy'-Entscheidungen oder der selektive Einsatz externer Dienstleister. Andererseits zählen zeitliche Vorgaben bzgl. der Terminierung des Entwicklungsprojektes dazu. Im Gegensatz zu den anderen Bedingungen kann der Entscheidungsträger diese Vorgaben meist nicht beeinflussen.
- *Personelle Kapazität:* Hierzu zählen die Mitarbeiter, die für das Entwicklungsprojekt verfügbar sind. Die Anzahl und deren Auslastung können im Verlauf des Projektes stark schwanken.
- *Finanzieller Spielraum:* Für die Weiterentwicklung und Veränderung der Architektur steht nur ein begrenztes Budget zur Verfügung. Es fehlt zudem häufig an der Bereitschaft der Kunden, die Kosten für Architekturveränderungen mitzutragen.

Technische Rahmenbedingungen

Neben den organisatorischen Aspekten sind die technischen Rahmenbedingungen wichtige Faktoren im Rahmen der Entscheidungsfindung. Zu den technischen Rahmenbedingungen zählen u. a. die Hardware- und Softwarebedingungen eines existierenden Softwaresystems und der zugehörigen Umsysteme oder bereits implementierte Architekturtechnologien, wie Muster oder Stile. Zudem sind bei der Weiterentwicklung sowie Veränderung der Architektur Standards- und Protokolle zu beachten und es steht nur eine bestimmte Werkzeugumgebung zur Verfügung. Im Gegensatz zu den organisatorischen Rahmenbedingungen können die technischen Bedingungen zum Teil sehr flexibel verändert und ergänzt werden, um den Anforderungen des Entwicklungsprojektes Rechnung zu tragen oder Konflikte mit den organisatorischen Bedingungen auszugleichen (siehe Abschnitt 5.1.4 auf S. 61). Die Checkliste umfasst neben den organisatorischen die folgenden technischen Rahmenbedingungen:

- *Hardware:* Für das zu entwickelnde oder zu verändernde Softwaresystem stehen nur begrenzte Kapazitäten an Rechenleistung, Netzwerk- oder Speicherkapazität zur Verfügung.

- *Softwaretechnologien:* Hierzu zählen u. a. Betriebssysteme, Programmiersprachen oder Komponentenmodelle des Systems, die bei der Weiterentwicklung und Veränderung der Architektur zu beachten sind.
- *Architekturtechnologien:* Die Eigenschaften der Lösungsansätze (z. B. der Grad der Verbesserung der Qualität, der Aufwand) können von bereits implementierten Architekturstilen und -mustern beeinträchtigt werden. So kann eine Schichtenarchitektur zwar ein Lösungsansatz zur Verbesserung der Wartbarkeit sein; jedoch kann ein bereits implementiertes Architekturmuster, z. B. ein Broker, dazu führen, dass der betreffende Teil der Komponentenstruktur des Softwaresystems nicht in Schichten aufgeteilt werden kann.

  Zur automatischen Erkennung bereits implementierter Muster und Stile können bestimmte Algorithmen angewendet werden [Balz00]. In diesem Zusammenhang existieren außerdem Verfahren, wie die Bedingungen und Einschränkungen, die aus Mustern oder auch Stilen resultieren, erkannt und klassifiziert werden können [Gie+06].
- *Standards:* Im Rahmen der Architekturentwicklung können standardisierte Kommunikationsprotokolle, Daten- oder Verschlüsselungsformate zu berücksichtigen sein.
- *Werkzeugumgebung:* Um die Weiterentwicklung und Veränderung der Architektur in einer hohen Qualität durchführen zu können, müssen die notwendigen Entwicklungswerkzeuge, wie CASE-Tools, spezielle Compiler oder eine automatisierte Testumgebung zur Verfügung stehen. Gerade bei architekturorientiertem Refactoring ist aufgrund der weit reichenden Restrukturierungen eine hohe Testabdeckung zu gewährleisten, die wegen der in der Praxis üblichen engen Rollout- und Releasetermine ohne automatisierte Tests bzw. Testautomaten nicht erreicht werden kann.

5.1.4 Auflösung von Konflikten zwischen den Rahmenbedingungen

Die organisatorischen und technischen Rahmenbedingungen können sich einander 'ergänzen' oder 'miteinander konkurrieren' (sehe zu den Beziehungen Abschnitt 4.1, S. 51ff.). Während 'sich ergänzende' Rahmenbedingungen nur bei der Identifizierung und Erhebung der Rahmenbedingungen berücksichtigt werden müssen, sind Konkurrenz-Beziehungen aufzulösen, da die Widersprüche eine zusätzliche kombinatorische Komplexität und damit ein Risiko bei der Auswahl, dem Vergleich und der Bewertung der alternativen Lösungsansätze darstellen. So droht die Gefahr, dass die falschen Lösungsansätze ausgewählt werden und letztlich eine Fehlentscheidung getroffen wird. Ein Beispiel für sich ergänzende Rahmenbedingungen ist beispielsweise dann gegeben, wenn genau die Entwicklungswerkzeuge, wie Compiler oder Frameworks, ausgewählt werden können, mit denen die Entwickler vertraut sind und die im Rahmen des Budgets liegen. Sind jedoch aus Kostengründen andere, kostengünstigere Werkzeuge einzusetzen, mit denen die Entwickler nur bedingt vertraut sind, ist der Entscheidungsträger mit konkurrierenden Rahmenbedingungen konfrontiert.

Aufgelöst werden die erkannten Konflikte durch eine Priorisierung und Anpassung der variablen Rahmenbedingungen, die der Entscheidungsträger zum Teil oder vollständig beeinflussen kann. In der Praxis sind das meist die technischen Bedingungen, u. a. die Werkzeugauswahl, die Hardwareausstattung oder die einzusetzenden Architekturtechnologien. Dabei werden zunächst die Rahmenbedingungen angepasst, die eine geringe Priorität besitzen. Demgegenüber stehen jene Bedingungen, die der Entscheidungsträger nicht oder nur in Ausnahmefällen beeinflussen kann, u. a. das Budget, der Zeitplan oder die bereits implementierten Software- und Architekturtechnologien. Welche Rahmenbedingungen der Checkliste variabel oder fix sind, differiert je nach Entscheidungssituation. Können die konkurrierenden Beziehungen nicht durch Anpassung der variablen Rahmenbedingungen aufgelöst werden, ist zu überprüfen, ob und inwieweit die fixen Rahmenbedingungen angepasst werden können oder ob die Ziele eingeschränkt werden müssen.

5.2 Architekturbeschreibung und Risikoeinstufung

In der zweiten Phase des Entscheidungsprozesses (siehe Abb. 3.1, S. 44) erfolgt die Beschreibung der für die Weiterentwicklung und Veränderung der Architektur relevanten Teile der Architektur. Das Architekturmodell – auch wenn es nur einzelne Teile der Gesamtarchitektur umfasst – stellt die Analysegrundlage dar, um systematisch alternative Lösungsansätze zur Erfüllung der Ziele und Rahmenbedingungen identifizieren, vergleichen und bewerten zu können. Darüber hinaus ist das Architekturmodell eine wichtige Referenz zur Erkennung von Abhängigkeitsbeziehungen, die im Rahmen der Entscheidungsfindung berücksichtigt oder bei Konflikten aufgelöst werden müssen. Um Risiken, die Komplexität und Unsicherheiten reduzieren zu können sowie eine Entscheidungsfindung nach rationalen Gesichtspunkten zu ermöglichen, muss das Architekturmodell aktuell, widerspruchsfrei und eindeutig interpretierbar sein (siehe Abschnitte 2.1 und 2.5 auf S. 22 und 30f.). Dabei hängt es vom Risiko der Architekturentscheidung ab, welcher Grad an formaler Genauigkeit im Hinblick auf den damit verbundenen Aufwand ökonomisch sinnvoll und gerechtfertigt ist. In den folgenden Abschnitten wird detailliert erläutert, anhand welcher Kriterien das Risiko einer Architekturentscheidung ermittelt werden kann und welche Schritte zur Architekturbeschreibung im Hinblick auf den Aufwand und den Nutzen durchgeführt werden sollten.

5.2.1 Einstufung des Risikos einer Architekturentscheidung

Um den Umfang der Architekturbeschreibung sowie den Grad an formaler Genauigkeit auf ein ökonomisch sinnvolles Maß zu reduzieren, ist zu ermitteln, wie groß das mit der Entscheidung verbundene Risiko ist. Für das Risiko einer Architekturentscheidung ist ausschlaggebend, welche negativen Konsequenzen bei einer Fehlentscheidung zu erwarten sind, z. B. ein verzögerter Rollout einer Software und die damit einhergehenden Umsatzeinbußen, und welche Handlungsoptionen zur Minimierung negativer Konsequenzen verbleiben [Pos+04]. Eine umfassende und formale Architekturbeschreibung ist im Hinblick auf das Aufwand-Nutzen-Verhältnis erst dann gerechtfertigt, wenn der geschätzte Nacharbeitungsaufwand zur Korrektur einer Fehlentscheidung höher ist als der Aufwand zur Beschreibung.

Das Risiko einer Architekturentscheidung ist über die Bestimmung des Umfangs und die formale Genauigkeit der Architekturentscheidung hinaus eines der wichtigsten Kriterien für die Auswahl der obligatorischen Tätigkeiten des gesamten Entscheidungsprozesses (siehe zu den obligatorischen und optionalen Tätigkeiten Tab. 10.2 ab S. 171ff.). So wird u. a. anhand des Risikos entschieden, in welchem Umfang szenariobasierte Architekturanalysen für die Ermittlung der Eigenschaften der alternativen Lösungsansätze, u. a. der Implementierungsaufwand oder Grad der Zielerfüllung, erforderlich sind (siehe Abschnitt 5.3.4.3, S. 76ff.).

Da die Erfahrungen in der Praxis zeigen, dass die Stakeholder das mit einer Architekturentscheidung verbundene Risiko unterschiedlich interpretieren, sind Kriterien erforderlich, anhand derer das Risiko nachvollziehbar und systematisch eingestuft werden kann. Für die Risikoermittlung ist zunächst zu bestimmen, ob es sich um ein geschäftskritisches Softwaresystem handelt. Im Gegensatz zu Softwaresystemen für rein administrative Zwecke, z. B. Lohnbuchhaltung oder Fuhrparkverwaltung, führen Fehlentscheidungen bei geschäftskritischen Softwaresystemen in der Praxis zeitnah zu massiven negativen Konsequenzen, z. B. in Form von Umsatzeinbrüchen, da die Geschäftstätigkeit nicht in vollem Umfang möglich ist. Vergleichbare negative Konsequenzen bei Softwaresysteme für administrative Zwecke treten in der Regel nur bei langfristigen Ausfällen auf. Erst dann haben auch Softwaresystemen für administrative Zwecke direkte Auswirkungen auf die Geschäftstätigkeit. Es ist jedoch von Unternehmen zu Unternehmen unterschiedlich zu bewerten, was ein geschäftskritisches Softwaresystem ist. Während z. B. für Supermarktketten und Discounter eine effiziente Logistikkette geschäftskritisch ist, benötigen Betreiber von Internet-Suchmaschinen eine zuverlässige und performante Indizierung der gefundenen Metadaten über Webseiten, um den Suchanfragen die relevanten Werbedaten zuzuordnen. Sobald die Lieferkette falsch koordiniert oder der Index-Katalog korrumpiert ist, können die Unternehmen nicht mehr das volle Sortiment an Produkten und Dienstleistungen anbieten. Der kurzfristige Ausfall des Lohnbuchhaltungssystems hat bei beiden Unternehmen jedoch nur marginale Auswirkungen auf die Geschäftstätigkeit.

Neben der Einstufung, ob es sich bei der Weiterentwicklung und Veränderung der Architektur um ein geschäftskritisches Softwaresystem handelt, ist der Aufwand für Rücksprünge (Undo) oder der Wechsel zu einer lauffähigen Version des Softwaresystems ein Kriterium für das Risiko einer Entscheidung (sog. Prinzip der Wechselkosten). Dieser Aufwand ist dann gering, wenn der von der Entscheidung betroffene Teil der Architektur mit geringem Aufwand ersetzt werden kann (geringe Wechselkosten). Ist ein Ersetzen nur mit hohem Aufwand möglich, z. B. aufgrund einer Vielzahl von benötigten Anpassungen, ist die Entscheidung als riskanter einzuschätzen (hohe Wechselkosten). Die Wechselkosten werden zu diesem Zeitpunkt der Entscheidung nur grob geschätzt. Sofern bei der Auswahl, Konkretisierung und Verfeinerung der Lösungsansätze erkannt wird, dass im Falle einer Fehlentscheidung mit erhöhten Wechselkosten zu rechnen ist, ist eine Anpassung der Risikoeinstufung der Entscheidung erforderlich.

Mithilfe der beiden Kriterien 'Kritikalität für Geschäftsentwicklung' und 'Wechselkosten' erfolgt eine Klassifizierung der Entscheidung in drei Risikoklassen (siehe Tab. 5.1). Die 'Klasse 0' beschreibt eine risikoarme, die 'Klasse 1' eine riskante und die 'Klasse 2' eine sehr riskante Architekturentscheidung.

Risikoklasse	Kriterium
Klasse 0	Architekturentscheidung bei einem wichtigen Softwaresystem, das jedoch nicht geschäftskritisch ist
Klasse 1	Architekturentscheidung bei einem geschäftskritischen Softwaresystem; ein Wechsel oder Rücksprung zu einer lauffähigen Version ist mit geringem Aufwand möglich
Klasse 2	Architekturentscheidung bei einem geschäftskritischen Softwaresystem; ein Wechsel oder Rücksprung zu einer lauffähigen Version ist mit erheblichem Aufwand verbunden

Tab. 5.1: Risikoklassen von Architekturentscheidungen

5.2.2 Beschreibung der relevanten Teile der Architektur

Die Architekturbeschreibung ist vor allem bei der Veränderung eines existierenden Softwaresystems notwendig, aber auch zur Beschreibung des Grundgerüsts und der Schnittstellen zu Drittsystemen bei einem neu zu entwickelnden Softwaresystem. Zur Beschreibung der Architektur sind, unter Berücksichtigung der Rationalitätsanforderungen sowie der Vermeidung von unnötigem Aufwand, die folgenden vier Schritte durchzuführen:

- *Schritt 1:* Analyse und Bewertung der Informationen (Dokumentation, Quelltext etc.)
- *Schritt 2:* Abgrenzung der relevanten Teile der Architektur
- *Schritt 3:* Auswahl der geeigneten Sichten sowie der geeigneten Beschreibungssprache
- *Schritt 4:* Beschreibung der Architektur und der Beziehungen zwischen den Komponenten

(1) Analyse und Bewertung der Informationen: Der erste Schritt zur Architekturbeschreibung ist die Analyse der zur Verfügung stehenden Informationen und Dokumentationen über die Architektur sowie das Softwaresystem, um Klarheit über die Bestandteile der Architektur und deren Beziehungen zu erhalten. Geeignete Informationsquellen sind u. a. die technische Dokumentation des Softwaresystems oder von Drittsystemen, bereits existierende Architekturmodelle oder eine Quelltext-Analyse, beispielsweise mittels einer Architektur-Rekonstruktion [Yan+99]. Um Risiken und Unsicherheiten zu reduzieren, müssen alle Informationen auf ihre Vollständigkeit und Aktualität analysiert, bewertet sowie strukturiert werden. Speziell bei existierenden Architekturmodellen ist im Detail zu überprüfen, in wieweit das Modell tatsächlich mit der Architektur des vorliegenden Softwaresystems übereinstimmt.

(2) Abgrenzung der relevanten Teile der Architektur: Im Anschluss an die Analyse und Bewertung der Informationen sind jene Teile der Architektur zu identifizieren, die für die Architekturentscheidung von Relevanz sind. Aus Aufwandsgründen werden nur jene Teile der Architektur beschrieben, die von den geplanten Weiterentwicklungs- und Veränderungsmaßnah-

men betroffen sind. Relevant sind die direkt von der Entscheidung betroffenen Komponenten, Schnittstellen und Beziehungen des Softwaresystems, welche z. B. die Schwachstellen für die mangelhafte Wartbarkeit des Softwaresystems darstellen. Das sind alle Komponenten, die 'in Konflikt mit' den Zielen stehen (siehe S. 51). Zudem sind bei Architekturentscheidungen der Klasse 1 und 2 auch die Teile der Architektur zu beschreiben, die nur in indirektem Zusammenhang mit der Weiterentwicklung oder Veränderung der Architektur stehen, aber bei der Auswahl alternativer Lösungsansätze oder deren Bewertung berücksichtigt werden müssen, z. B. Schnittstellen zu Datenbanken oder Drittsystemen. Dazu gehören vor allem jene Komponenten, die bereits jetzt eine 'positive Auswirkung' auf die Ziele haben (siehe S. 51). Gerade bei sehr riskanten Architekturentscheidungen der Klasse 2 sollte ein großer Betrachtungsraum für die Architekturbeschreibung gewählt werden. Dieser Mehraufwand ist gerechtfertigt, da bei einer Fehlentscheidung ein Rücksprung zu einer lauffähigen Version des Softwaresystems mit einem erheblichen Aufwand verbunden ist. Da die Auswahl der alternativen Lösungsansätze erst in der folgenden Phase des Entscheidungsprozesses stattfindet (siehe Abschnitt 5.3, S. 66ff.), ist das erstellte Architekturmodell u. U. anzupassen, zu ergänzen oder zu erweitern.

(3) Auswahl der geeigneten Sichten sowie der geeigneten Beschreibungssprache: Gerade bei riskanten oder sehr riskanten Entscheidungen der Klasse 1 und 2 sollte die Beschreibung der Architektur mittels einer Architekturbeschreibungssprache erfolgen, damit die Qualität der Architekturanalysen erhöht und Widersprüche bei der Interpretation der Analyseergebnisse vermieden werden können. Entsprechend der Argumentation in Abschnitt 2.5 (siehe S. 32) eignen sich für die Architekturbeschreibung vor allem ACME, UML oder die relativ moderne Beschreibungssprache xADL. Sofern im vorangegangenen Schritt ein großer Betrachtungsraum ausgewählt wurde, kann das Architekturmodell zur Reduzierung der kombinatorischen Komplexität in verschiedene Sichten aufgeteilt werden. Dafür stehen verschiedene Sichten-Konzepte zur Verfügung, die je nach Entscheidungssituation ausgewählt werden müssen (siehe die Ausführungen auf S. 32f.).

(4) Beschreibung der Architektur und der Beziehungen zwischen den Komponenten: Im vierten Schritt werden die relevanten Teile der Architektur mit der ausgewählten Beschreibungssprache und – je nach Umfang des Betrachtungsraumes – dem ausgewählten Sichten-Konzept beschrieben. Erfolgt die Architekturbeschreibung auf der Grundlage eines bereits existierenden Modells, ist es entsprechend der Ergebnisse der Informationsanalyse (erster Schritt der Architekturbeschreibung, S. 63) u. U. anzupassen und zu ergänzen.

5.2.3 Ermittlung unbekannter Abhängigkeitsbeziehungen zwischen den Komponenten

Ein wichtiges Ziel bei der Beschreibung der Architektur ist die Erkennung bisher unbekannter oder undokumentierter Abhängigkeitsbeziehungen zwischen den Architekturkomponenten. Dadurch kann einerseits das Risiko reduziert werden, dass bei der Implementierung eines Lösungsansatzes Nacharbeiten und Korrekturen aufgrund plötzlich auftretender Abhängigkeits- oder Konfliktbeziehungen erforderlich werden. Andererseits kann dadurch die Qualität der Architekturentscheidung verbessert werden, indem frühzeitig genau die Lösungsansätze identifiziert werden können, mit denen die Ziele in hohem Maße erfüllt werden.

Da die Architekturkomponenten in einer Vielzahl unterschiedlicher Abhängigkeitsbeziehungen zueinander stehen können, wurden in Abschnitt 4.1 (siehe S. 51ff.) verschiedene Beziehungsarten identifiziert. Durch die Fokussierung auf Beziehungsarten wird die Anzahl an unterschiedlichen Beziehungen – die sich mitunter nur in ihrer Bezeichnung unterscheiden – reduziert und die Beziehungen werden klarer, verständlicher sowie eindeutiger zu interpretieren. Zwischen den Architekturkomponenten werden im Rahmen des Entscheidungsprozesses die folgenden Beziehungsarten unterschieden:

- Architekturkomponenten können von anderen Komponenten 'erfordert' oder 'benutzt' werden
- Zusammengehörige Komponenten können miteinander 'aggregiert' sein
- Komponenten mit generischem Funktionsumfang können durch Komponenten mit erweitertem Funktionsumfang 'spezialisiert' werden

Der im Rahmen der Architekturbeschreibung (siehe vorangegangener Absatz) betrachtete Ausschnitt der Architektur wird durch szenariobasierte Architekturanalysen hinsichtlich der Beziehungsarten untersucht. Dabei wird u. a. ermittelt, welche Komponenten einander 'erfordern', einander 'benutzen' oder miteinander 'aggregiert' sind. Sofern bislang unentdeckte und nicht beschriebene Abhängigkeitsbeziehungen identifiziert werden, ist das Architekturmodell entsprechend zu ergänzen. Quelle für die Erkennung der Abhängigkeitsbeziehungen zwischen den Komponenten sind einerseits die Ergebnisse der Architekturanalyse, mit der die Schwachstellen und Problembereiche der Architektur im Rahmen der Identifizierung der Ziele ermittelt wurden (siehe Abschnitt 5.1.1, S. 56f.). Wurde für den betroffenen Architekturteil keine oder keine ausreichende Architekturanalyse durchgeführt, ist dies durch ein Architekturreview mittels einer szenariobasierten Architekturanalyse nachzuholen. Als Grundlage kann hierfür die Architecture-Tradeoff-Analysis-Method (ATAM) oder die Architecture-Level-Modifiability-Analysis (ALMA, siehe für beide Abschnitt 2.4, S. 29) angewendet werden. Dabei werden verschiedene Szenarien am Modell der Softwarearchitektur durchgespielt, um die Auswirkungen auf andere Komponenten erkennen zu können.

Sofern die kritischen Komponenten noch nicht in ausreichendem Maße bekannt sind, können quantitative Architekturanalysen (siehe Abschnitt 2.4 ab S. 28) zur Erhebung sowie Auswertung von Architekturmetriken durchgeführt werden, u. a. das Distanzmaß [Sim+99] oder die Kopplungs- und Kohäsionsmetriken [She+01]. Mit den Architekturmetriken können speziell jene zentralen Komponenten identifiziert werden, die von der Mehrheit der anderen Komponenten 'erfordert' oder 'benutzt' werden.

Nicht in jedem Fall steht der Aufwand zur Erkennung und Dokumentation der Abhängigkeitsbeziehungen in einem ökonomisch sinnvollen Verhältnis zum erwarteten Nutzen. So ist bei risikoarmen und leicht zu korrigierenden Architekturentscheidungen der Klassen 0 und 1 (siehe vorangegangener Abschnitt 5.2.1, S. 63) abzuwägen, in welchem Ausmaß und Detailgrad die Abhängigkeitsbeziehungen zwischen den für die Entscheidung relevanten Komponenten identifiziert und dokumentiert werden müssen. Speziell bei den szenariobasierten Architekturanalysen ist die Anzahl der Analyseschritte aus Aufwandsgründen auf ein Minimum zu reduzieren. Dabei kann insbesondere die Vielzahl an Präsentationen und Diskussionen reduziert werden, da die Ziele der Entscheidung bereits in der ersten Phase des Entscheidungsprozesses identifiziert wurden.

5.3 Auswahl, Verfeinerung und Konkretisierung alternativer Lösungsansätze

In der dritten Phase des Entscheidungsprozesses (siehe Abb. 3.1, S. 44) werden anhand eines systematischen Verfahrens jene alternativen Lösungsansätze ausgewählt, verfeinert und konkretisiert, die zur Erfüllung der Ziele geeignet sind. Die Schritte sind erforderlich, um im Rahmen der Entscheidungsfindung mehrere alternative Lösungsansätze miteinander zu vergleichen und zur bewerten; entsprechend der Rationalitätskriterien muss der Vergleich und die Bewertung nach objektiven Kriterien erfolgen (siehe Abschnitt 2.1, S. 22). Für die Entscheidung über die Lösungsansätze sind daher eine Vielzahl an Kriterien und Entscheidungsfaktoren zu berücksichtigen, u. a. Ziele, Rahmenbedingungen, Eigenschaften der Lösungsansätze. Zur Reduzierung der kombinatorischen Komplexität – sowie der damit verbundenen Unsicherheiten und Risiken – ist ein systematisches Verfahren zur Auswahl, Konkretisierung und Verfeinerung der Lösungsansätze erforderlich. Das Vorgehen muss es zudem ermöglichen, das die Lösungsansätze anhand ihrer softwaretechnischen Details analysiert werden können, um die Lücken des generischen Vorgehens zur Entscheidungsfindung zu schließen (siehe die Argumentation in Abschnitt 2.1 (siehe S. 23f.). Umfasst eine Architekturentscheidung beispielsweise fünf Fundamentalziele, zu denen jeweils fünf Instrumentalziele gehören, sind für die Bewertung und den Vergleich jedes Lösungsansatzes mindestens 25 Kriterien zu überprüfen. Zusätzlich kann es erforderlich werden, Seiteneffekte, Varianten und Wechselwirkungen zu betrachten. Ohne ein systematisches Verfahren zur Auswahl, Verfeinerung und Konkretisierung der Lösungsansätze führt die kombinatorische Komplexität zu großer Unsicherheit und zu hohen Risiken für die Architekturentscheidung.

Das entwickelte Verfahren zur Auswahl, Verfeinerung und Konkretisierung alternativer Lösungsansätze umfasst die folgenden Tätigkeiten (siehe die zusammenfassende Darstellung in Abb. 5.4):

- *Aufstellung eines Grobplanes bei mehrstufigen Architekturentscheidungen:* Insbesondere bei mehrstufigen Architekturentscheidungen sind verschiedene Einzelentscheidungen (für jede Stufe) in Kombination zu betrachten. Gerade bei riskanten und sehr riskanten Entscheidungen der Risikoklasse 1 und 2 (siehe S. 63) ist es zur Reduzierung der Komplexität, der Risiken und der Unsicherheit erforderlich, einen Grobplan der Architekturentwicklung aufzustellen.
- *Vorauswahl auf Kategorieebene:* Sofern im Anschluss an die Strukturierung der Ziele noch kein ausreichender Überblick über die alternativen Lösungsansätze besteht, erfolgt zunächst eine Vorauswahl auf einer Kategorieebene (siehe Kapitel 7, S. 138ff.). In diesem Schritt werden jene Kategorien ausgewählt, die zur Erreichung der Ziele prinzipiell geeignet sind.
- *Analyse und Ranking alternativer Lösungsansätze:* Um sich bei der anschließenden Konkretisierung auf die besten Lösungsansätze zu fokussieren zu können, werden alternative Lösungsansätze mittels einer Architekturanalyse untersucht. Auf der Basis der Analyseergebnisse erfolgen ein Ranking und die Vorauswahl der besten zwei bis drei Lösungsansätze.
- *Verfeinerung und Konkretisierung alternativer Lösungsansätze:* Im letzten Schritt werden die vorausgewählten Lösungsansätze konkretisiert und verfeinert, um die Eigenschaften der Lösungsansätze, u. a. den Implementierungsaufwand sowie den Grad der Zielerreichung, im Detail ermitteln zu können. Dabei wird eine möglichst exakte Maßnahmenfolge erarbeitet, die den Prozess der Implementierung des Lösungsansatzes beschreibt.

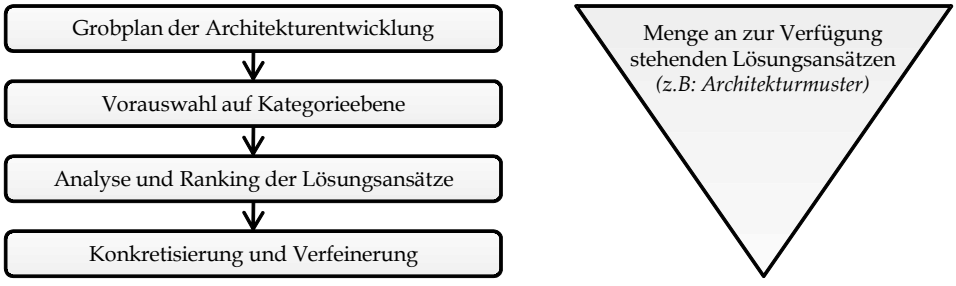


Abb. 5.4: Verfahren zur Vorauswahl und Konkretisierung alternativer Lösungsansätze

5.3.1 Aufstellung eines Grobplanes bei mehrstufigen Architekturentscheidungen

Da bei mehrstufigen Architekturentscheidungen verschiedene Kombinationen von Einzelentscheidungen im Hinblick auf die Zielerreichung betrachtet werden müssen, ist es gerade bei riskanten und sehr riskanten Entscheidungen der Risikoklasse 1 und 2 (siehe S. 63) zur Reduzierung der Komplexität, der Risiken und der Unsicherheit erforderlich, einen Grobplan der Architekturentwicklung aufzustellen. Da zunächst grob geplant wird, mit welchen Maßnahmen die Ziele erfüllt werden können, steht der ungefähre Ablauf der Architekturentwicklung zur Auswahl der alternativen Lösungsansätze bereits fest. Eine wichtige Voraussetzung ist jedoch die Überprüfung, ob mit dem Grobplan auch die Ziele im erforderlichen Maß erfüllt und die Rahmenbedingungen eingehalten werden können. Ist beispielsweise die starre Anbindung eines Softwaresystems an eine Datenbank zu flexibilisieren, kann der Grobplan beispielsweise die Schritte 'Einführung einer Datenbank-Abstraktionsschicht' sowie 'Anpassungen bei den Datenbankzugriffen' umfassen, um die Ziele im gewünschten Umfang zu erreichen. Für jeden einzelnen Schritt können im Anschluss geeignete Lösungsansätze ausgewählt werden, wie beispielsweise die Frameworks Hibernate oder der Java-Persistance-Layer [Beeg07] für den Schritt der 'Einführung einer Datenbank-Abstraktionsschicht'.

Entwicklung von Lösungsszenarien zur Konzeption der Weiterentwicklung der Architektur

Um einen Grobplan der Architekturentwicklung erstellen zu können, werden zunächst Lösungsszenarien skizziert. Ein Lösungsszenario ist die informale Beschreibung eines Lösungsansatzes, bzw. einer geeigneten Komponentenstruktur, mit dem die in der ersten Phase identifizierten Ziele, z. B. die grundlegende Verbesserung der Wartbarkeit oder der Sicherheit, erfüllt werden können. Der Grad der Zielerreichung ist durch geeignete Verfahren, die in den folgenden Absätzen näher beschrieben werden, zu überprüfen. Die Lösungsszenarien sind einerseits zur Verdeutlichung notwendig, in welche Richtung das Modell der existierenden Architektur zur Erreichung der Ziele weiterentwickelt werden kann. Damit wird das Risiko gesenkt, bei der Aufstellung der Kombination von Lösungsansätzen in eine Sackgasse zu geraten. Andererseits stellt das Lösungsszenario eine wichtige Grundlage dar, um durch den Vergleich mit dem Modell der existierenden Architektur die notwendigen Maßnahmen zur Architekturentwicklung und -veränderung ableiten zu können.

Ein Beispiel für die Erstellung von Lösungsszenarien ist in Abb. 5.5 dargestellt. Auf der linken Seite ist das mit Schwachpunkten (hervorgehoben durch Fettdruck) behaftete existierende Modell der Architektur dargestellt. Auf der rechten Seite sind die skizzierten Lösungsszenarien der zukünftigen Architektur abgebildet, mit denen die Schwachstellen des existierenden Modells ausgeglichen werden. Aus den Unterschieden zwischen den Modellen werden die notwendigen Maßnahmen zur Architekturentwicklung abgeleitet und es kann eine systematische Auswahl der Lösungsansätze durchgeführt werden.

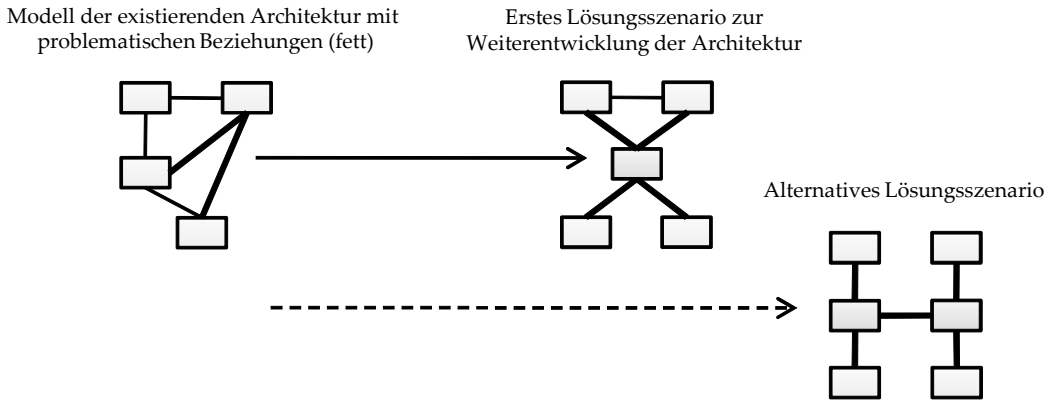


Abb. 5.5: Skizzierung von Lösungsszenarien zur Konzeption der Architekturveränderung

Um die Lösungsszenarien entwickeln zu können, muss bekannt sein, auf welche Art und Weise die Architektur verändert oder weiterentwickelt werden kann, um die Ziele im erforderlichen Umfang zu erreichen. Hilfreich sind dabei einerseits die Erfahrungen der an der Entscheidung beteiligten Entwickler, Projektleiter, Architekten und Kunden mit vergleichbaren Problem- und Fragestellungen aus früheren Projekten. Andererseits können Referenzarchitekturen ausgewertet werden, die

Rollen, Struktur und Organisation von Komponenten für eine Klasse von Architekturen beschreiben, wie z. B. bei [Gur+03] für Referenzarchitekturen für Unternehmensportale. Die Beschreibung eines Lösungsszenarios erfolgt mittels grafischer Beschreibungselemente einer Architekturbeschreibungs- oder Modellierungssprachen, z. B. ADL, UML oder xADL (siehe Abschnitt 2.5, S. 31f.), um das Modell eindeutig interpretieren zu können.

Überprüfung der Lösungsszenarien und Ermittlung der Differenzen

Für die Überprüfung, ob mit den skizzierten Lösungsszenarien die Ziele im erforderlichen Umfang erfüllt werden können, sind je nach Risikoeinstufung, Zielstellung spezielle Analyseverfahren notwendig. Nicht in jedem Fall sind die Lösungsszenarien so übersichtlich, aussagekräftig und klar strukturiert, dass direkt aus dem Szenario erkannt werden kann, in welchem Umfang die Ziele erreicht werden können. Zur Überprüfung der Zielerreichung sind gemäß der Argumentation in Abschnitt 2.4 (siehe S. 29ff.) vor allem szenariobasierte Verfahren geeignet, z. B. die Architecture Tradeoff Analysis Method (ATAM). Hierfür ist zunächst aus dem Lösungsszenario selbst ein prototypisches Modell der zukünftigen Architektur zu erstellen (einen sog. Architekturprototypen). Bei der Durchführung einer szenariobasierten Architekturanalyse werden zunächst Szenarien ausgewählt, mit denen die Architekturprototypen auf die Erfüllung ihrer Ziele hin untersucht werden können. Diese Szenarien werden anhand von Walkthroughs angewendet, um die in den Szenarien beschriebenen Situationen durchzuspielen sowie Rückschlüsse auf den Grad der Erfüllung der Ziele zu erhalten. Je nach Risiko einer Architekturentscheidung kann es zudem erforderlich werden, die Einhaltung der organisatorischen oder technischen Rahmenbedingungen zu überprüfen.

Ist die Szenarioanalyse dafür nicht ausreichend, können auch ausgewählte Teile der der Lösungsszenarien prototypisch implementiert werden. Dabei ist bei risikoarmen Architekturentscheidungen (Klasse 0, siehe S. 63) abzuwägen, ob der Aufwand zur prototypischen Implementierung im Hinblick auf das Aufwand-Nutzen-Verhältnis gerechtfertigt ist. Alternativ oder ergänzend zur Szenarioanalyse können Simulationen durchgeführt werden, um z. B. die Performance des Lösungsszenarios zu überprüfen. Da hierfür jedoch die betreffenden Teile der Lösungsszenarien mit einer speziellen Modellierungssprache zu beschreiben sind, wie z. B. Rapide (siehe Abschnitt 2.5, S. 31), ist der Aufwand nur bei sehr riskanten Architekturentscheidungen der Klasse 2 gerechtfertigt, da im Falle einer Fehlentscheidung ein Rücksprung zu einer lauffähigen Version mit erheblichem Aufwand verbunden ist.

Die Grundlage für die Aufstellung des Grobplans ist der Vergleich des Modells der existierenden Architektur mit den Architekturprototypen, um aus den Differenzen zwischen den Modellen Rückschlüsse auf die notwendigen Maßnahmen zur Weiterentwicklung und Veränderung der Architektur ziehen zu können. Dabei wird einerseits ermittelt, welche Komponenten, Komponentengruppen und Beziehungen unverändert auch in den Architekturprototypen enthalten sind. Andererseits werden die Unterschiede zwischen den Modellen ermittelt: Welche Komponenten und Beziehungen sind neu hinzugekommen, welche sind anzupassen und zu verändern? Anhand der ermittelten Unterschiede werden die Maßnahmen zur Weiterentwicklung der Architektur in Richtung der Lösungsszenarien ermittelt. Die Maßnahmen sind zu diesem Zeitpunkt der Entscheidungsfindung meist noch grob, ungenau und umfassen im Detail nicht alle Entwicklungsaktivitäten. Sie werden bei der Verfeinerung und Konkretisierung der Lösungsansätze korrigiert, angepasst und ergänzt (siehe Abschnitt 5.3.4, S. 73ff.).

Berücksichtigung und Auflösung von Abhängigkeiten

Die groben Maßnahmen, für die im weiteren Verlauf der Entscheidungsfindung alternative Lösungsansätze ausgewählt und konkretisiert werden, können in Abhängigkeitsbeziehungen zueinander stehen. Diese müssen bei der Aufstellung des Grobplanes einerseits berücksichtigt und andererseits im Falle von Konflikten aufgelöst werden. Entsprechend den Ausführungen in Abschnitt 4.1 (siehe Tab. 4.1 auf S. 51) können sich die Maßnahmen gegenseitig 'erzwingen' oder 'ermöglichen'; zudem können bestimmte Entwicklungsmaßnahmen den Einsatz anderer Maßnahmen 'verhindern'. Derartige Konflikte sind aufzulösen, indem die betroffenen Maßnahmen in einer anderen Reihenfolge angeordnet bzw. aufgeteilt und entsprechend verändert werden. Sofern die Konflikte dadurch nicht aufgelöst werden können ist eine Anpassung des Lösungsszenarios erforderlich; je nach Umfang der Anpassungen ist eine erneute Überprüfung der Zielerreichung und der Einhaltung

der Rahmenbedingungen notwendig. Die Erfahrungen aus der Praxis zeigen, dass die Maßnahmen zu diesem Zeitpunkt der Entscheidungsfindung meist so grob sind, dass zur Reihenfolgeplanung keine methodische Unterstützung erforderlich ist. Ein Beispiel für Maßnahmen, die sich gegenseitig 'erzwingen', sind Restrukturierungen an den Elementen der Benutzeroberfläche, die sehr eng an einen Controller angebunden sind. Bei der Reihenfolgeplanung muss berücksichtigt werden, dass aufgrund der engen Bindung jegliche Veränderungen an den Elementen der Benutzeroberfläche entsprechende Veränderungen am Controller erfordern. Im Hinblick auf die Reihenfolgeplanung kann es je nach Entscheidungssituation erforderlich sein, entweder zuerst den Controller oder zuerst die Elemente der Benutzeroberfläche anzupassen.

Um die Abhängigkeitsbeziehungen zwischen den Maßnahmen identifizieren zu können, sind die Abhängigkeitsbeziehungen zwischen den Architekturkomponenten im Modell der existierenden Architektur zu untersuchen. So können die Komponenten in Benutzungs- oder Spezialisierungsbeziehungen zueinander stehen (siehe für weitere Abhängigkeitsbeziehungen Abschnitt 4.1, S. 50ff.), die u. a. Auswirkungen auf die Reihenfolgeplanung der Maßnahmen haben.

Priorisierung und Aufstellung des Grobplanes

Im Anschluss an die Strukturierung und die Reihenfolgeplanung erfolgt die Gruppierung von Maßnahmen und die Aufstellung des Grobplans der Architekturentwicklung. Der Grobplan umfasst dabei jene Maßnahmen, mit denen die existierende Architektur in Richtung der Lösungsszenarien weiterentwickelt werden kann, um die erforderlichen Ziele zu erfüllen. Zur Aufstellung sind zunächst zusammengehörige Maßnahmen zu Maßnahmenfolgen zu gruppieren. Da für jede Maßnahmenfolge alternative Lösungsansätze ausgewählt, konkretisiert und verfeinert werden, entspricht eine Maßnahmenfolge einer Entscheidungsstufe. Ein wichtiger Punkt ist dabei die Formulierung von Zwischenzielen für jede Stufe der Entscheidung. Zwischenziele sind die Entscheidungskriterien für die Einzelentscheidungen auf jeder Stufe (siehe hierzu Abschnitt 3.4, S. 48ff.). Die Zwischenziele leiten sich aus den Fundamental- und vor allem aus den Instrumentalzielen des in der ersten Phase erstellten Ziel-Wirkungsmodells ab (siehe Abschnitt 5.1.2, S. 57ff.).

Die Aufstellung eines Grobplans zur Architekturentwicklung ist in Abb. 5.6 schematisch dargestellt. Die problematischen Komponentenbeziehungen im Modell der existierenden Architektur (links) sind durch Fettdruck hervorgehoben; zur Lösung dieses Problems sind auf der rechten Seite zwei Lösungsszenarien mit ihren Architekturprototypen abgebildet. Um die Architektur in Richtung des ersten Lösungsszenarios weiterzuentwickeln, sind die Maßnahmenfolgen eins bis drei in aufsteigender Reihenfolge erforderlich; für die Weiterentwicklung der Architektur in Richtung des zweiten Lösungsszenarios sind die Maßnahmenfolgen eins, vier und fünf erforderlich. Die Architekturentscheidung ist demnach auf drei Stufen verteilt; auf jeder der drei Stufen der Entscheidung werden für fünf Maßnahmenfolgen unterschiedliche Lösungsansätze ausgewählt, verfeinert und bewertet.

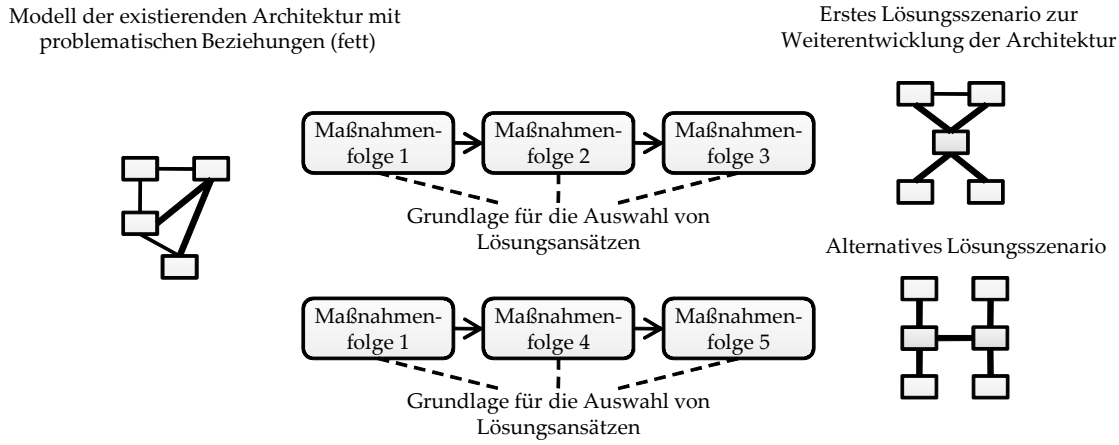


Abb. 5.6: Erstellung eines Grobplans der Architekturentwicklung

5.3.2 Vorauswahl auf Kategorieebene

Sofern nach der widerspruchsfreien Strukturierung der Ziele oder der Aufstellung des Grobplanes keine ausreichenden Anhaltspunkte für mögliche Lösungsansätze vorliegen, erfolgt zunächst eine Vorauswahl der alternativen Lösungsansätze auf der Ebene der Kategorien von Mustern, Stilen, Softwareprodukten oder Technologien. Insbesondere bei der Verbesserung der Softwarequalität bei über Jahre gewachsenen Softwaresystemen ist häufig unklar, welche alternativen Lösungsansätze zur Erreichung der Ziele geeignet sind. In diesen Fällen ist eine Vorauswahl auf der Kategorieebene geboten, da aufgrund der Vielzahl theoretisch möglicher Lösungsansätze die Verfeinerung und Konkretisierung (siehe Abschnitt 5.3.4 ab S. 73ff.) jedes einzelnen Lösungsansatzes aus ökonomischen Gründen nicht sinnvoll ist. Der Aufwand zur Durchführung des Entscheidungsprozesses übersteigt sonst dessen Nutzen und das Ziel einer effizienten Entscheidungsfindung wird nicht erreicht (siehe Abschnitt 2.11, S. 42f.). Zur Vorauswahl auf der Ebene der Kategorien von Lösungsansätzen stehen die in Abschnitt 7 erarbeiteten Kategorien und deren generische Qualitätseigenschaften zur Verfügung (siehe S. 138ff.).

Da die Vorauswahl auf der abstrakten Ebene der Kategorien von Mustern, Stilen, Softwareprodukten oder Technologien erfolgt, können auch individuelle Erfahrungslösungen berücksichtigt werden. Das sind bereits erfolgreich implementierte Lösungsansätze, die aufgrund ihres individuellen Charakters jedoch keine allgemein akzeptierten Muster oder Stile darstellen. Um die Erfahrungslösungen bei der Vorauswahl zu berücksichtigen, müssen sie in die Kategorien aus Abschnitt 7 eingeordnet werden können. In Anbetracht der Individualität der Erfahrungslösungen muss jedoch außerdem sichergestellt werden, dass die gewünschten Eigenschaften der Lösungsansätze, wie z. B. die Verbesserungen der Testbarkeit in einer verteilten Umgebung, auch beim aktuell vorliegenden Softwaresystem erzielt werden können.

Zur Vorauswahl auf Kategorieebene sind die folgenden zwei Schritte notwendig:

- *Schritt 1 – Auswahl der Kategorien anhand des Beitrags zur Zielerreichung:* Im ersten Schritt werden jene Kategorien ausgewählt, deren Eigenschaften, z. B. die Qualitätseigenschaften, mit den Zielen aus Abschnitt 5.1 (siehe S. 56ff.) grundsätzlich übereinstimmen. Ein Beispiel für eine Kategorie von Lösungsansätzen zur Verbesserung der Wartbarkeit sind die Entwurfsmuster der Kategorie 'Wartbare Subsysteme' (siehe S. 141f.).
- *Schritt 2 – Berücksichtigung negativer Auswirkungen:* Im anschließenden zweiten Schritt sind die negativen Auswirkungen der Lösungsalternativen der ausgewählten Kategorien auf die Ziele zu berücksichtigen. Die Architekturmuster der Kategorie 'Portable Softwaresysteme' können sich z. B. negativ auf die Wartbarkeit auswirken (siehe S. 146). Jedoch muss gerade bei negativen Auswirkungen berücksichtigt werden, dass die Eigenschaften der Kategorien von Lösungsansätzen generisch und allgemein sind. Im konkreten Einzelfall können einerseits weitere negative Qualitätseigenschaften zu berücksichtigen sein; andererseits können die in Abschnitt 7 dokumentierten Eigenschaften u. a. durch Optimierungen bei der Implementierung wieder ausgeglichen werden.
- *Rahmenbedingungen:* Da die Vorauswahl von Lösungsansätzen auf Kategorieebene sehr abstrakt ist, kann die Einhaltung der Rahmenbedingungen, z. B. die Einhaltung von Terminplänen und Budgetvorgaben, nur eingeschränkt durchgeführt werden. Die detaillierte Berücksichtigung der Rahmenbedingungen erfolgt daher erst im Rahmen des anschließenden Rankings oder bei der Konkretisierung und Verfeinerung der alternativen Lösungsansätze (siehe Abschnitt 5.3.4, S. 73ff.).

5.3.3 Analyse und Ranking alternativer Lösungsansätze

Die Erfahrungen aus der Praxis zeigen, dass oftmals auch eine Vielzahl alternativer Lösungsansätze zur Erfüllung der Ziele geeignet ist. Das ist u. a. häufig dann der Fall, wenn eine Vorauswahl von alternativen Lösungsansätzen auf Kategorieebene durchgeführt wurde (siehe vorangegangener Abschnitt 5.3.2). In diesen Fällen ist ein Ranking der Lösungsansätze erforderlich, bei dem das Potenzial der Lösungsansätze zur Erreichung der Ziele unter Berücksichtigung der Rahmenbedingungen ermittelt wird. Das ist notwendig, da aus Aufwandsgründen nur die besten zwei bis drei Lösungsansätze bei der Konkretisierung und Verfeinerung berücksichtigt werden können, die das größte Potenzial zur Erreichung der Ziele haben.

Grundlage des Rankings ist eine kurze Szenarioanalyse der alternativen Lösungsansätze, in welchem Umfang sie die Ziele und die Rahmenbedingungen erfüllen. Dadurch können die Lösungsansätze miteinander verglichen werden, um die besten zwei bis drei Lösungsansätze für die Konkretisierung und Verfeinerung auszuwählen. Für die Analyse der Eigenschaften der Lösungsansätze wird eine reduzierte szenariobasierte Architekturanalyse durchgeführt, beispielsweise mittels der Architecture-Tradeoff-Analysis-Method (ATAM). Speziell zur Untersuchung der Veränderbarkeit und Modifizierbarkeit von Architekturen kann die Architecture-Level Modifiability-Analysis eingesetzt werden (ALMA, siehe für beide Abschnitt 2.4 auf S. 29). Zur Reduzierung des Aufwands kann im Falle von ALMA u. a. auf die ersten beiden der fünf Analyseschritte, die Identifizierung der Ziele und die Beschreibung der Architektur, verzichtet werden, da dies bereits in den ersten beiden Phasen des Entscheidungsprozesses erfolgte (siehe Abschnitt 5.1 u. 5.2, S. 56ff. und 62ff.). Die entsprechenden Schritte können auch bei ATAM entfallen. Zusätzlich dazu können im Hinblick auf ein ökonomisch sinnvolles Verhältnis zwischen Aufwand und Nutzen die vielfältigen Diskussionen sowie Präsentationen der Ergebnisse der Szenarioanalysen auf ein Minimum reduziert werden, da es sich hierbei nur um ein Ranking und noch nicht um die letztliche Entscheidung für einen bestimmten Lösungsansatz handelt.

Für das Ranking der Lösungsansätze sind die folgenden Analyseschritte durchzuführen:

- *Schritt 1:* Ermittlung der Szenarien
- *Schritt 2:* Prototypische Modellveränderung anhand der Lösungsansätze
- *Schritt 3:* Anwendung der Szenarien und Berücksichtigung der Rahmenbedingungen
- *Schritt 4:* Ranking und Auswahl der besten zwei bis drei Lösungsansätze

(1) Ermittlung der Szenarien: Im ersten Schritt werden die Szenarien ermittelt, anhand derer die Lösungsansätze auf ihre Eigenschaften hin untersucht werden können. Die Szenarien sollten typische Anwendungsfälle sein, um die Erfüllung der Ziele zu überprüfen. Speziell für Veränderungsszenarien, wie z. B. beim Refactoring eines existierenden Softwaresystems, unterscheiden Bengtsson et al. die folgenden drei Kategorien: 'Veränderungen an der funktionalen Spezifikation', 'Veränderungen bei den Anforderungen' und 'Umgebungsveränderungen' [Ben+04]. Diesen Kategorien sind verschiedene Szenarien zugeordnet, die entweder direkt übernommen oder auf deren Grundlage individuellen Szenarien ermittelt werden können (siehe hierzu Tab. 10.3 im Anhang 10.3, S. 175f.). Beispiele für Szenarien sind das Auftreten eines kritischen Fehlers in einer Komponente für das Ziel Zuverlässigkeit oder Veränderungen am Datenbankzugriff für das Ziel Wartbarkeit. Sofern bereits früher eine Architekturanalyse mittels Szenarien durchgeführt wurde und die Szenarien noch aktuell sind, können sie wiederverwendet werden.

(2) Prototypische Modellveränderung anhand der Lösungsansätze: Im anschließenden zweiten Schritt erfolgt eine prototypische Modellveränderung, um die geplanten Veränderungen durchzuspielen und eine Grundlage für die Abschätzung der Eigenschaften der Lösungsansätze zu erhalten. Auf der Grundlage des in der zweiten Phase erarbeiteten Architekturmodells (siehe Abschnitt 5.2, S. 62ff.) wird ein Prototyp der zukünftigen Architektur entwickelt. Im Rahmen der Modellveränderung (siehe Abb. 5.7, S. 71) wird das Modell der existierenden Architektur so verändert, dass die Implementierung des betrachteten Lösungsansatzes weitestgehend abgebildet wird. Sofern sich die Lösungsansätze strukturell unterscheiden, kann es erforderlich sein, die Modellveränderung für alle Lösungsansätze separat vorzunehmen. Gerade bei risikoarmen Entscheidungen der Klasse 0 (siehe S. 63) ist jedoch darauf zu achten, den Modellierungs- und Analyseaufwand auf ein Minimum zu reduzieren, da es sich hierbei nur um ein Ranking und eine Vorauswahl handelt.

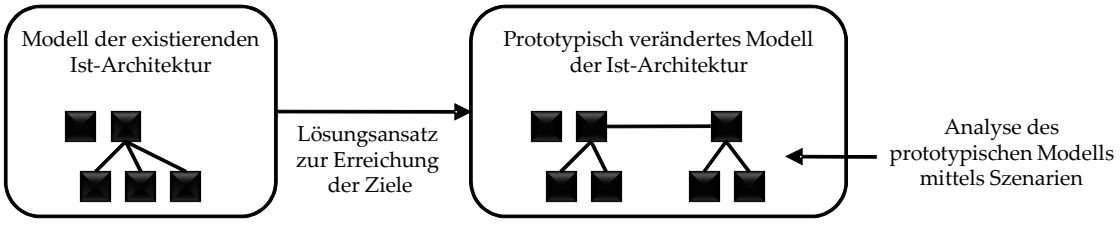


Abb. 5.7: Szenarioanalyse zur Ermittlung der Eigenschaften der Lösungsansätze

Die Schritte, die zur Modellveränderung erforderlich sind, werden dokumentiert, um die Maßnahmen zur Implementierung der Lösungsansätze ableiten zu können. Diese grob formulierte Maßnahmenfolge wird im Rahmen der Verfeinerung und Konkretisierung der Lösungsansätze verfeinert und ergänzt (siehe Abschnitt 5.3.4, S. 73ff.). Soll die zukünftige Architektur beispielsweise das Entwurfsmuster 'Fassade' (siehe S. 141) umfassen, werden die zur Modellveränderung notwendigen Aktivitäten, wie z. B. die Hinzufügung von Schnittstellen sowie die Anpassung der Zugriffe, notiert.

Speziell bei architekturorientiertem Refactoring ist die Forderung nach kleinen Schritten mit möglichst isolierten Veränderungen am Softwaresystem zu beachten [RoLi04]. Wird bei der Erarbeitung der Prototypen der zukünftigen Architektur erkannt, dass zur Zielerreichung umfangreiche Restrukturierungen notwendig werden, ist ein Rücksprung zur ersten Phase des Entscheidungsprozesses erforderlich, um die Architekturentscheidung in mehrere Einzelentscheidungen aufzuteilen (siehe Abschnitt 5.1.2, S. 59). Eine umfangreiche Architekturveränderung in einem Schritt ist zwar prinzipiell möglich; jedoch ist das mit einem erheblichen Risiko verbunden, da ein Rücksprung auf eine unveränderte und lauffähige Version des Softwaresystems oftmals nur bedingt möglich bzw. mit großem Aufwand verbunden ist.

(3) Anwendung der Szenarien und Berücksichtigung der Rahmenbedingungen: Im dritten Schritt werden die Eigenschaften der Lösungsansätze auf der Grundlage der Untersuchung des prototypischen Architekturmodells ermittelt. Die Anwendung der Szenarien erfolgt durch Walkthroughs, um zu ermitteln, in welchem Umfang die geforderten Ziele erreicht werden können. Ein Walkthrough ist ein Peer-Review-Analyseverfahren, bei dem die Schritte eines Prozesses oder eines Algorithmus theoretisch und konzeptionell durchgespielt werden [Wha+94]. Für das Ziel Zuverlässigkeit wird durch einen Walkthrough z. B. das Auftreten eines kritischen Fehlers in einer Komponente durchgespielt und es wird überprüft, in welchem Grad der betrachtete Lösungsansatz zu einer Verbesserung der Zuverlässigkeit führt.

Im Rahmen der Szenarioanalyse wird darüber hinaus überprüft, ob die Rahmenbedingungen für die Implementierung der Lösungsansätze eingehalten werden. Durch die Modellveränderung erhält der Entscheidungsträger einen ersten Überblick über die notwendigen Maßnahmen zur Implementierung der Lösungsansätze. Er kann dadurch auf grober Ebene bestimmen, wie aufwendig und riskant die Implementierung ist und ob die organisatorischen und technischen Rahmenbedingungen eingehalten werden, wie z. B. die Verfügbarkeit der benötigten Anzahl an Entwicklern oder die Einhaltung von Architekturrichtlinien. Die Grundlage für die Überprüfung der Rahmenbedingungen ist die in der ersten Phase des Entscheidungsprozesses erarbeitete Checkliste (siehe Abschnitt 5.1.3, S. 59ff.). Sofern die Rahmenbedingungen nicht eingehalten werden können, ist abzuwägen, ob der betreffende Lösungsansatz angepasst oder durch eine Alternative ersetzt werden kann. Ansonsten kann der betreffende Lösungsansatz im Rahmen der Entscheidungsfindung nicht mehr berücksichtigt werden.

(4) Ranking und Auswahl der besten zwei bis drei Lösungsansätze: Anhand der Ergebnisse der verfeinerten Architekturanalyse erfolgen im vierten Schritt das Ranking sowie die Auswahl der zwei bis drei Lösungsansätze, die das größte Potenzial zur Erreichung der Ziele haben. Die Lösungsansätze, die neben den Zielen alle geforderten Rahmenbedingungen erfüllen, werden dabei in der Rangfolge an oberster Stelle eingeordnet. Im Anschluss erfolgt die Auswahl der zwei bis drei besten Lösungsansätze, um unnötigen Aufwand bei der anschließenden Verfeinerung und Konkretisierung der alternativen Lösungsansätze zu vermeiden.

5.3.4 Verfeinerung und Konkretisierung alternativer Lösungsansätze

Nach Abschluss der Vorauswahl der zwei bis drei Lösungsansätze, die zur Erreichung der Ziele am besten geeignet sind, werden diese verfeinert, konkretisiert und angepasst. Dieser Schritt ist obligatorisch und notwendig, da für den systematischen Vergleich sowie die systematische Bewertung der alternativen Lösungsansätze deren Eigenschaften, u. a. der Grad der Zielerreichung, im Detail bekannt sein müssen. Zudem schreiben die Rationalitätskriterien ein sachlich-begründetes Vorgehen zur Einhaltung der rationalen Prozeduralität vor (siehe Abschnitt 2.1, S. 22). Zur detaillierten Ermittlung der Eigenschaften sind verschiedene Verfeinerungs- und Konkretisierungsschritte erforderlich. So sind u. a. die konkreten Maßnahmen zu bestimmen, die zur Implementierung der alternativen Lösungsansätze erforderlich sind, sowie der Umfang der Zielerreichung und die Einhaltung der Rahmenbedingungen zu untersuchen. Ohne die Detailinformationen über die Lösungsansätze verbleiben erhebliche Unsicherheiten darüber, mit welcher der beiden Lösungsansätze die Ziele am besten erfüllt werden können, ob die Rahmenbedingungen dafür ausreichend sind und bei welchem Lösungsansatz Veränderungen am funktionalen Verhalten des Softwaresystems zu erwarten sind. Um beispielsweise den Java-Persistance-Layer und das Hibernate-Framework [Beeg07] als alternative Lösungsansätze zur Verbesserung der Portabilität vergleichen zu können, müssen deren Eigenschaften im Detail ermittelt werden. Für den Vergleich der Lösungsansätze ist u. a. zu ermitteln welche Veränderungen an den Komponenten, die auf die Datenbank zugreifen, jeweils notwendig sind, in welchem Umfang die Portabilität jeweils verbessert werden kann und welche Risiken und Seiteneffekte zu berücksichtigen sind.

Das Verfahren zur Konkretisierung und Verfeinerung umfasst die folgenden vier Schritte (siehe Abb. 5.8):

- *Schritt 1 – Aufstellung der Maßnahmenfolge:* Bei diesem Schritt erfolgt die Aufstellung und Verfeinerung der Maßnahmenfolge, die den Prozess zur Implementierung der Lösungsansätze beschreibt.
- *Schritt 2 – Auswertung der Abhängigkeitsbeziehungen:* Die Maßnahmenfolge kann aufgrund von Abhängigkeiten zu ergänzen, anzupassen und zu korrigieren sein. Zu den Abhängigkeiten zählen einerseits Abhängigkeitsbeziehungen zwischen den Maßnahmen und andererseits Beziehungen zwischen den Architekturkomponenten.
- *Schritt 3 – Überprüfung der Eigenschaften:* Im Anschluss werden die Eigenschaften der Lösungsansätze ermittelt und überprüft. Dazu zählt der Grad der Zielerreichung oder auch die Einhaltung organisatorischer und technischer Rahmenbedingungen.
- *Schritt 4 – Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Speziell bei architekturorientiertem Refactoring ist die Überprüfung erforderlich, ob durch die Implementierung der alternativen Lösungsansätze das funktionale Verhalten verändert wird und welche Anpassungen erforderlich sind.

In Abb. 5.8 (siehe S. 74) ist dargestellt, in welcher Reihenfolge die einzelnen Schritte durchgeführt werden und wann ein Rücksprung erforderlich ist. Die Ergebnisse der Verfeinerung und Konkretisierung sind die Grundlage für den Vergleich der Lösungsansätze und die abschließende Entscheidung. Die vier Schritte werden für jeden Lösungsansatz nacheinander durchlaufen.

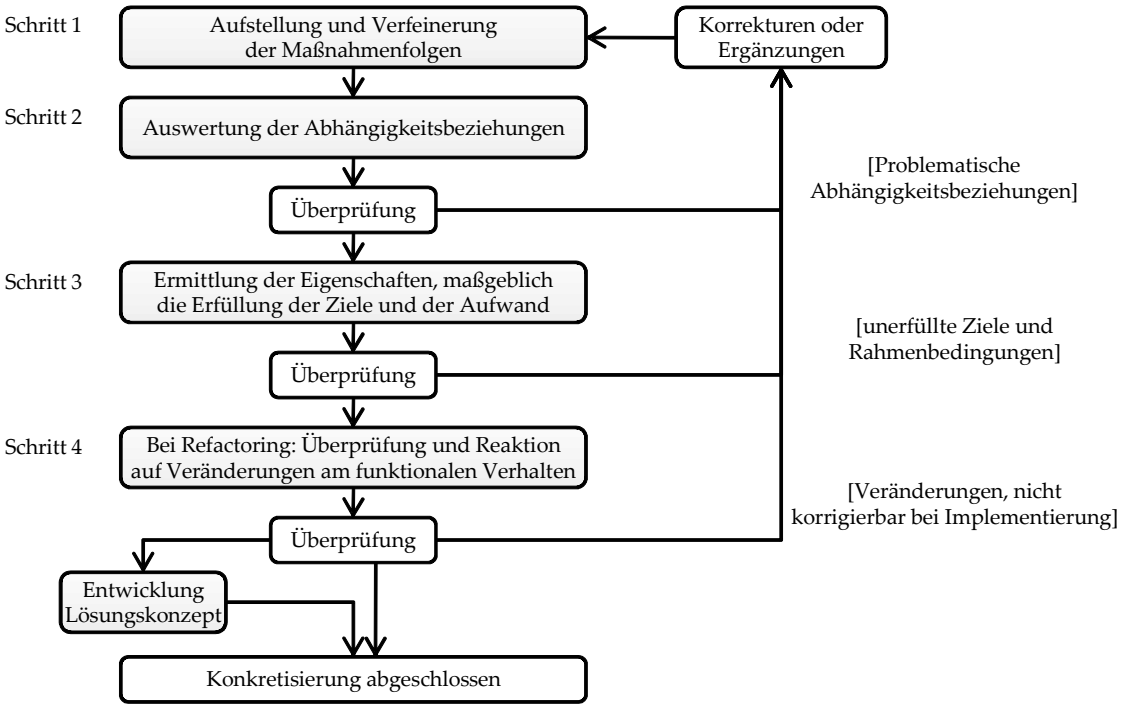


Abb. 5.8: Schritte zur Verfeinerung und Konkretisierung der Lösungsansätze

5.3.4.1 Aufstellung und Verfeinerung der Maßnahmenfolgen

Die Grundlage für die Ermittlung der Eigenschaften der Lösungsansätze ist die Aufstellung einer detaillierten Maßnahmenfolge für jeden vorausgewählten Lösungsansatz (siehe Abb. 5.8). Die Maßnahmenfolge beschreibt dabei im Detail die Schritte zur Implementierung der Lösungsansätze. Ein Hilfsmittel bei der Aufstellung ist die prototypische Modellveränderung, bei der für jeden Lösungsansatz durchgespielt wird, wie das Modell der existierenden Architektur zur Erreichung der Ziele verändert und weiterentwickelt wird. Sofern bereits für das Ranking der Lösungsansätze eine prototypische Modellveränderung (siehe S. 71f.) vorgenommen wurde, wird anhand der dokumentierten Schritte der – bereits durchgeführten – prototypischen Modellveränderung eine detaillierte Maßnahmenfolge aufgestellt. Die identifizierten Maßnahmen werden sukzessive verfeinert, um eine möglichst detaillierte Maßnahmenfolge zu beschreiben. Dabei werden zu grobe Aktivitäten, die eine Vielzahl von Einzelaktivitäten umfassen, aufgeteilt; im Gegensatz dazu werden elementare und zu feingranulare Aktivitäten aggregiert, um die kombinatorische Komplexität zu reduzieren. Sofern noch kein Architekturprototyp erstellt wurde, ist dies nachzuholen (siehe für die Schritte Abschnitt 5.3.3, S. 71ff.). So wird beispielsweise zur Aufstellung der Maßnahmenfolge zur Implementierung des Frameworks Hibernate [Beeg07] zur Abstraktion der Datenbankzugriffe durchgespielt, welche Veränderungen an der Datenbank, der Datenbankschnittstelle und den betreffenden Komponenten notwendig werden.

Bei riskanten und sehr riskanten Architekturentscheidungen der Risikoklasse 2 und höher (siehe S. 63) ist zur Wahrung der Verständlichkeit, Klarheit und Übersichtlichkeit eine grafische Darstellung der Maßnahmenfolge vorzunehmen. Aufgrund des erhöhten Risikos muss die Entscheidungsfindung dokumentiert und zu einem späteren Zeitpunkt nachvollzogen werden können. Die Nachvollziehbarkeit ist in der Praxis ein wichtiger Grund, um auch eine risikoarme Architekturentscheidung mithilfe des Entscheidungsprozesses durchzuführen. Zur Visualisierung der Maßnahmen und möglichen Varianten existieren verschiedene Methoden und Verfahren. Entsprechend der Argumentation in Kapitel 2.10 (S. 42) erfolgt die Visualisierung anhand von Aktivitätsdiagrammen der UML, da Rücksprünge und zyklische Abhängigkeiten abgebildet werden können.

Da für alle zwei bis drei Lösungsansätze unterschiedliche Maßnahmenfolgen aufgestellt werden, werden die Maßnahmenfolgen bzw. die jeweiligen Aktivitätendiagramme als erste Stufe eines Entscheidungsbaums (siehe zu den Entscheidungsbäumen Abschnitt 2.10, S. 40) dargestellt. Es erfolgt somit eine Kombination von Entscheidungsbäumen und Aktivitätsdiagramme, um die Verständlichkeit, Klarheit und Übersichtlichkeit zu erhöhen und die Entscheidungsfindung nachvollziehbar zu gestalten. Eine exemplarische Darstellung der Maßnahmenfolgen von zwei Lösungsansätzen ist in Abb. 5.9 abgebildet.

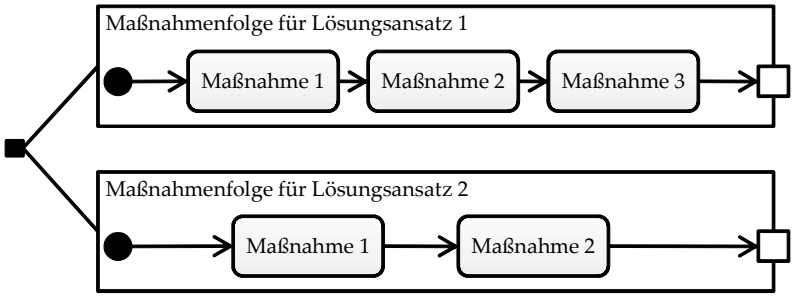


Abb. 5.9: Beispiel für Kombination von Aktivitätsdiagrammen und Entscheidungsbäumen

5.3.4.2 Auswertung der Abhängigkeitsbeziehungen

Bei der Aufstellung und Verfeinerung der Maßnahmenfolge sind insbesondere jene Architekturkomponenten zu berücksichtigen, die 'in Konflikt mit' den Zielen stehen, oder bereits jetzt eine 'positive Auswirkung' auf die Ziele haben. Nach Abschluss der Aufstellung und Verfeinerung der Maßnahmenfolgen sollten einerseits keine der für die Entscheidungsfindung relevanten Komponenten mehr 'in Konflikt mit' den Zielen stehen. In der Praxis ist oft der Fall zu beobachten, dass zwar eine Weiterentwicklung und Veränderung der Architektur durchgeführt wird, die Schwachstellen und Problembereiche jedoch nur zum Teil behoben werden. Zur vollständigen Behebung der Problembereiche und damit zur vollständigen Erfüllung der Ziele sind entsprechende Maßnahmen zur Weiterentwicklung und Veränderung der Architektur zu konzipieren. Andererseits ist bei der Aufstellung und Verfeinerung der Maßnahmenfolgen zu berücksichtigten, dass sich einzelne Architekturkomponenten des existierenden Softwaresystems bereits 'positiv' auf die Ziele auswirken. Sofern die betroffenen Komponenten durch die aktuelle Weiterentwicklung und Veränderung der Architektur verändert werden, ist zu überprüfen, dass die 'positive Auswirkung' erhalten bleibt. Es sollte beispielsweise vermieden werden, dass mit der Entfernung eines Proxy die ohnehin schlechte Performance zusätzlich belastet wird.

Ein wichtiger Schritt bei der Aufstellung und Verfeinerung der Maßnahmenfolgen ist außerdem die Berücksichtigung und Auswertung der Abhängigkeitsbeziehungen zwischen den Architekturkomponenten selbst. Die folgenden Beziehungsarten können zwischen den Komponenten auftreten und müssen im Rahmen der Verfeinerung und Konkretisierung der Lösungsansätze berücksichtigt und ausgewertet werden (siehe Tab. 4.1, S. 51):

- *Erfordert von* und/oder *Benutzt von:* Aufgrund technischer oder fachlicher Abhängigkeiten, können zwei oder mehr Komponenten einander 'erfordern und/oder sich gegenseitig benutzen'. Bei der Aufstellung und Verfeinerung der Maßnahmenfolgen ist zu beachten, dass Veränderungen von zentralen Komponenten, die von vielen anderen Komponenten benutzt werden, sehr riskante und tiefgreifende Untersuchungen der abhängigen Komponenten erfordern und dass bei der Veränderung eine bestimmte Reihenfolge der Veränderungsschritte (siehe nachfolgenden Absatz) einzuhalten ist.
- *Aggregiert mit:* Zwei oder mehr Komponenten können miteinander 'aggregiert' sein, wenn die Komponenten in einer Teil-Ganzes-Beziehung oder in einer Parent-Child-Beziehung stehen, z. B. bei Komponenten eines Subsystems. Bei der Aufstellung und Verfeinerung der Maßnahmenfolgen gilt es zu beachten, dass 'aggregierte' Komponenten in enger Verbindung und hoher Kohärenz zueinander stehen und sich oftmals wechselseitig benutzen.

- *Spezialisiert zu:* Architekturkomponenten können durch eine oder mehrere Komponenten funktional ergänzt oder erweitert werden. Die generischen Komponenten, die oftmals nur eine grundlegende Funktionalität besitzen, werden dabei durch andere Komponente 'spezialisiert'. Da die spezialisierten Komponenten auf die grundlegende Funktionalität der generischen Komponenten zugreifen, muss bei der Aufstellung und Verfeinerung der Maßnahmenfolge überprüft werden, welche Veränderungen an den generischen Komponenten Anpassungen an den spezialisierten Komponenten erfordern. Im Gegensatz dazu können die 'spezialisierten' Komponenten unabhängig von den generischen Komponenten verändert werden.

Zudem ist bei der Reihenfolgeplanung der Maßnahmen zu überprüfen, ob Abhängigkeitsbeziehungen zwischen den Maßnahmen selbst existieren. Entsprechend den Ausführungen in Abschnitt 4.1 (siehe Tab. 4.1 auf S. 51) können sich die Maßnahmen gegenseitig 'erzwingen', 'ermöglichen' oder miteinander in Konflikt stehen und sich gegenseitig 'verhindern'. Die Konfliktbeziehungen müssen aufgelöst werden, indem die betroffenen Maßnahmen in einer anderen Reihenfolge angeordnet bzw. aufgeteilt und entsprechend verändert werden. Sofern die Konflikte dadurch nicht aufgelöst werden können, ist eine Anpassung des betreffenden Lösungsansatzes durchzuführen; sofern auch dies nicht zu einer Auflösung der Konflikte führt, muss ein alternativer Lösungsansatz ausgewählt werden, z. B. auf der Grundlage der Tabelle mit den Lösungsansätzen und derem Ranking (siehe Abschnitt 5.3.3, S. 70ff.). Die Maßnahmen, welche sich gegenseitig 'erzwingen' oder 'ermöglichen' sind in einer entsprechenden Reihenfolge einzuordnen. Ein Beispiel für Maßnahmen, die sich gegenseitig 'erzwingen' oder 'ermöglichen', sind Restrukturierungen an den Elementen der Benutzeroberfläche, die in enger Bindung mit einem Controller stehen. Bei der Reihenfolgeplanung ist zu berücksichtigen, dass aufgrund der engen Bindung jegliche Veränderungen an den Elementen der Benutzeroberfläche entsprechende Veränderungen am Controller erfordern. Im Hinblick auf die Reihenfolgeplanung kann es erforderlich sein, entweder zuerst den Controller oder zuerst die Elemente der Benutzeroberfläche anzupassen.

5.3.4.3 Ermittlung der Eigenschaften der Lösungsansätze

Grad der Zielerreichung

Im dritten Schritt der Konkretisierung und Verfeinerung der Lösungsansätze (siehe Abb. 5.8, S. 74) werden die Eigenschaften ermittelt, die mit der Implementierung des Lösungsansatzes erwartet werden können. Es wird dabei u. a. im Detail ermittelt, in welchem Umfang die Ziele erreicht werden und welcher Implementierungsaufwand dafür erforderlich ist. Die detaillierte Ermittlung der Werte ist notwendig, um die Lösungsansätze im Detail miteinander vergleichen zu können und weitere Unsicherheiten bei der Entscheidungsfindung zu reduzieren. Betrachtet wird hierbei die erste Gabelung des Entscheidungsbaumes (siehe die Darstellung in Abb. 5.10). So muss beispielsweise für den Vergleich des Hibernate-Frameworks und des Java-Persistance-Layers [Beeg07] als alternative Lösungsansätze zur Abstraktion von Datenbankzugriffen im Detail ermittelt werden, welche Vor- und Nachteile die Lösungsansätze jeweils bieten und welcher Implementierungsaufwand erforderlich ist.

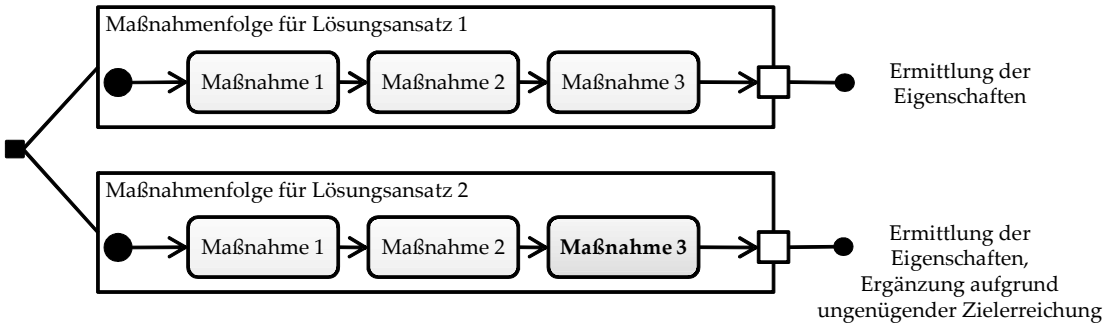


Abb. 5.10: Ermittlung der Eigenschaften der Lösungsansätze

Wie auch beim Ranking der Lösungsansätze können deren Eigenschaften, vor allem der Grad der Erfüllung der Ziele, aufgrund der Komplexität und der Größe des Betrachtungsraums oftmals nur durch zusätzliche Analysen ermittelt werden. In diesen Fällen wird erneut eine angepasste und deutlich reduzierte szenariobasierte Architekturanalyse (siehe S. 71) durchgeführt. So kann im Falle von ALMA u. a. auf die ersten beiden der fünf Analyseschritte, die Identifizierung der Ziele und die Beschreibung der Architektur, verzichtet werden, da dies bereits in den ersten beiden Phasen des Entscheidungsprozesses erfolgte (siehe Abschnitt 5.1 u. 5.2, S. 56ff. u. 62ff.). Vor allem bei weniger riskanten Entscheidungen (Klasse 0, siehe S. 63) können die vielfältigen Diskussionen und Präsentationen der Ergebnisse der Szenarioanalyse auf ein Minimum reduziert werden. Zudem können Szenarien wiederverwendet werden, wenn bereits im Rahmen des Rankings der Lösungsansätze eine szenariobasierte Architekturanalyse durchgeführt wurde. Bei riskanten und sehr riskanten Entscheidungen (Klasse 1 und 2) ist darauf zu achten, dass die bereits existierenden Szenarien alle relevanten Ziele abdecken; bei identifizierten Lücken sind ergänzende Szenarien zu entwerfen. Die Szenarien werden im Anschluss auf das prototypisch veränderte Architekturmodell angewendet, welches mit der Implementierung des betreffenden Lösungsansatzes erwartet werden kann. Sofern ein bereits existierendes prototypisches Architekturmodell wiederverwendet wird, ist darauf zu achten, dass es entsprechend der Verfeinerungen und Ergänzungen an der Maßnahmenfolge (siehe den ersten, vorangegangenen Schritt der Verfeinerung und Konkretisierung) angepasst wird. Wie bereits beim Ranking der Lösungsansätze erfolgt die Anwendung der Szenarien ebenfalls durch Walkthroughs (siehe S. 72).

Alternativ oder ergänzend zur Szenarioanalyse können Simulationen durchgeführt werden, um z. B. die Performance eines Lösungsansätze zu überprüfen. Hierfür sind die betreffenden Teile der Lösungsszenarien mit einer speziellen Modellierungssprache zu beschreiben, wie z. B. Rapide (siehe Abschnitt 2.5, S. 31). Gerade bei risikoarmen Architekturentscheidungen der Klasse 0 steht dieser Aufwand jedoch nur selten in einem ökonomisch sinnvollen Verhältnis zum Nutzen.

Weichen die ermittelten Eigenschaften der Lösungsansätze von den Zielvorgaben ab, wird die Maßnahmenfolge entweder um zusätzliche Maßnahmen ergänzt oder entsprechend korrigiert (siehe durch Fettdruck hervorgehobene Maßnahme in Abb. 5.10, S. 76). Bei besonders starken Abweichungen von den Vorgaben, die vom Entscheidungsträger nicht mehr toleriert werden können, wird der Lösungsansatz nicht weiter betrachtet, da der Aufwand zur weiteren Konkretisierung nicht lohnt. In dem Fall ist abzuwägen, ob der nicht weiter betrachtete Lösungsansatz durch eine Alternative ersetzt werden kann, die aufgrund des Rankings (siehe Abschnitt 5.3.3, S. 70ff.) bisher nicht betrachtet wurde.

Einhaltung der Rahmenbedingungen

Neben der Zielerreichung ist die Einhaltung der Rahmenbedingungen zu überprüfen. Zu den Rahmenbedingungen zählen die organisatorischen und die technischen Bedingungen, die in der ersten Phase im Rahmen einer Checkliste erhoben wurden (siehe Abschnitt 5.1.3, ab S. 59ff.).

Bei der Berücksichtigung der Rahmenbedingungen werden die bereits verfeinerten Maßnahmenfolgen und die prototypischen Modelle der zukünftigen Architektur anhand der Checkliste der Rahmenbedingungen überprüft. Der Entscheidungsträger betrachtet dabei die einzelnen Punkte auf der Checkliste und überprüft, ob die Rahmenbedingungen eingehalten oder überschritten werden oder ob die Bedingungen negative Einflüsse auf die Eigenschaften der Lösungsansätze haben. So ist u. a. zu überprüfen, ob mit der Implementierung der Lösungsansätze die vorgegebenen Terminpläne eingehalten werden können oder das zur Verfügung stehende Budget ausreichend ist. Andererseits ist gerade im Hinblick auf die technischen Rahmenbedingungen außerdem zu überprüfen, in welcher Art und Weise die Rahmenbedingungen die Eigenschaften der Lösungsansätze beeinflussen. So kann z. B. ein Lösungsansatz zur Verbesserung der Zugriffszeiten durch ungenügende Hardware-Ressourcen, Beschränkungen aufgrund proprietärer Nachrichtenformate oder Wechselwirkungen mit bereits implementierten Mustern und Stilen negativ beeinflusst werden. Um die negativen Einflüsse auszugleichen, ist der betreffende Lösungsansatz entsprechend anzupassen. Dabei können entweder die Maßnahmenfolge angepasst und verändert oder ergänzende Maßnahmen definiert werden.

Skalierungsvarianten zur Quantifizierung der Eigenschaftswerte

Um eine Bewertung der Lösungsansätze nach dem rationalen Bewertungsverfahren durchführen zu können, sind die Eigenschaftswerte, maßgeblich die Zielerreichungsgrade und der Implementierungsaufwand, in einer einheitlichen sowie vergleichbaren Form zu quantifizieren. Zur Quantifizierung stehen die folgenden beiden Skalierungsvarianten zur Verfügung [EiWe03]:

- *Ordinalskala:* Die Werte dieser Skala lassen sich in einer Rangfolge anordnen, z. B. in Form von Noten. Die Werte der Ordinalskala müssen transitiv sein. Das ist dann der Fall, wenn der Wert *a* größer als *b* und *b* größer als *c* ist und der Wert *a* auch im direkten Vergleich größer als *c* ist. Bei der Erhebung einer Ordinalskala ist auf eine gerade Anzahl der Ausprägungen zu achten, da damit der typische Bewertungsfehler 'Tendenz zur Mitte' vermieden wird. Bei einer ungeraden Anzahl, z. B. die fünf Ausprägungen 'sehr gut', 'gut', 'mittel', 'negativ' und 'sehr negativ', vermeidet der Bewerter unbewusst die positiven und negativen Extremwerte und tendiert bei der Bewertung zu 'mittel'.
- *Kardinalskala:* Die Anordnung der Werte erfolgt bei der Kardinalskala ebenfalls in einer Rangfolge. Im Gegensatz zur Ordinalskala existiert jedoch ein willkürlicher festgelegter Nullpunkt, z. B. wenn eine Zielerreichung von unter 30 % keiner Qualitätsverbesserung (dem Nullpunkt) entspricht. Durch den Nullpunkt können die Intervalle zwischen den Rangfolgen exakter quantifiziert werden, als dies bei einer Ordinalskala möglich ist. So ist z. B. die Aussage, Lösungsansatz *A* hat eine um 35 % höhere Zielerreichung als Lösungsansatz *B*, exakter als die Aussage, *A* ist eine Notenstufe besser als *B*. In der Praxis werden für kardinal skalierte Werte typischerweise Prozentangaben verwendet.

Berücksichtigung von wahrscheinlichen und unwahrscheinlichen Varianten

Oft können nicht alle Eigenschaften eines Lösungsansatzes mit Sicherheit bestimmt werden, da die Zusammenhänge zu komplex sind oder der betrachtete Ausschnitt des prototypisch veränderten Architekturmodells nicht ausreichend groß genug ist. Im Falle von Unsicherheiten werden wahrscheinliche und unwahrscheinliche Varianten für die Eigenschaften eines Lösungsansatzes unterschieden. Dabei ist zu berücksichtigen, dass es mehrere Varianten für eine Eigenschaft eines Lösungsansatzes geben kann; die Summe der Wahrscheinlichkeiten aller Varianten muss 100 % ergeben. Ein Beispiel ist eine Kapselung von Komponenten, die mit einer Wahrscheinlichkeit von 80 % zu einer verbesserten Verständlichkeit und Testbarkeit führt, sofern die Zugriffe auf die gekapselten Komponenten nur über Schnittstellen möglich sind. Zum Zeitpunkt der Entscheidungsfindung kann jedoch bereits abzusehen sein, dass mit einer Wahrscheinlichkeit von 20 % der Termindruck zu hoch und nur eine prototypische Schnittstellenimplementation mit eingeschränkter Funktionalität zur Verfügung stehen wird. In dieser zwar unwahrscheinlichen Variante wird die Testbarkeit nicht im erforderlichen Umfang erfüllt; trotzdem muss diese Variante in der Entscheidungsfindung berücksichtigt werden, da sie in 20 von 100 Fällen und Situationen eintritt.

Für die Messung von Wahrscheinlichkeiten steht in der Entscheidungstheorie eine Vielzahl von Methoden und Konzepten zur Verfügung (siehe Abschnitt 2.2, S. 24ff. und insbesondere Anhang 10.1, S. 168ff.). Da die Messung von Wahrscheinlichkeiten einen erheblichen Aufwand für die Entscheidungsfindung darstellt, ist abzuwägen, ob auf die Berücksichtigung von Wahrscheinlichkeiten bei weniger riskanten Entscheidungen der Klasse 0 und 1 (siehe S. 63) verzichtet werden kann. Bei Entscheidungen der Klasse 0 und 1 ist ein Rücksprung zu einer lauffähigen Version mit geringem Aufwand möglich.

Sofern die Eigenschaften in unterschiedlichen Varianten auftreten, können für die jeweiligen Varianten ergänzende Maßnahmen definiert werden, um negative Auswirkungen kompensieren zu können. In Abb. 5.11 ist dieser Zusammenhang grob skizziert. Bei 'Lösungsansatz 1' sind zwei Varianten zu unterscheiden; mit einer Wahrscheinlichkeit von 60% werden alle Ziele erfüllt, mit einer Wahrscheinlichkeit in Höhe von 40% wird ein Ziel nicht erreicht. Für diese Variante wird eine ergänzende Maßnahme (durch Fettdruck hervorgehoben) definiert, um das Ziel trotzdem erfüllen zu können.

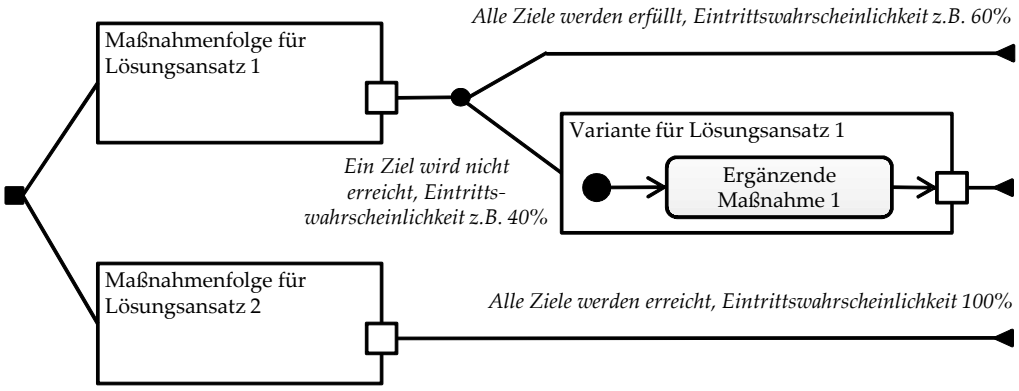


Abb. 5.11: Wahrscheinliche und unwahrscheinliche Varianten für die Eigenschaften von Lösungsätzen

5.3.4.4 Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften

Der vierte Schritt zur Konkretisierung (siehe Abb. 5.8, S. 74) ist die Untersuchung, ob durch die Weiterentwicklung und Veränderung der Architektur anhand der Lösungsansätze Veränderungen am den funktionalen Systemeigenschaften auftreten und entsprechende Korrekturen notwendig sind. Dies ist speziell beim Refactoring existierender Softwaresysteme zu beachten, da die Systeme bei identischen Eingangswerten (vor und nach erfolgtem Refactoring) dieselben Ausgangswerte liefern müssen [Fowl05] (oder [Kol+03]). Aber auch bei der Entwicklung neuer Softwaresysteme sind Um- und Drittsysteme zu berücksichtigen, deren funktionale Eigenschaften durch die Architekturentscheidungen nicht beeinträchtigt werden dürfen. Wird beispielsweise eine existierende Web-Anwendung zur Verbesserung der Portabilität auf ein Java-Applet migriert, ist zu berücksichtigen, dass funktionale Einschränkungen aufgrund des Sandbox-Prinzips existieren; so kann das Hauptfenster eines Java-Applets beispielsweise nicht maximiert werden. Wird diese Funktion jedoch benötigt, sind Korrekturen am betreffenden Lösungsansatz bzw. an der Maßnahmenfolge erforderlich.

Zur Überprüfung sind zuerst die relevanten funktionalen Anforderungen zu identifizieren, die auch nach erfolgter Weiterentwicklung und Veränderung der Architektur in unveränderter Art und Weise erfüllt werden müssen. Zur Reduzierung des Aufwands werden nur die Anforderungen überprüft, die mit den direkt von der Entscheidung betroffenen Komponenten, Schnittstellen und Beziehungen des Softwaresystems in Verbindung stehen. Da bei der Beschreibung der Architektur in der zweiten Phase des Entscheidungsprozesses bereits eine Analyse sowie eine Strukturierung der Informationen über das Softwaresystem erfolgte (siehe Abschnitt 5.2, S. 56ff.), kann diese Informationsgrundlage zur Auswahl der relevanten Anforderungen wiederverwendet werden.

Im Anschluss wird anhand von Walkthroughs (siehe S.72) oder Tests einer prototypischen Implementierung überprüft, ob die funktionalen Anforderungen erfüllt werden können. Die Grundlage für die Überprüfung ist einerseits das prototypische Architekturmodell des Lösungsansatzes, welches einen Ausschnitt der zukünftigen Architektur des Softwaresystems abbildet. Dieser Ausschnitt ist zu vergrößern, sofern er für die Überprüfung der funktionalen Anforderungen nicht ausreichend ist. Können durch die Walkthroughs durch das veränderte Architekturmodell keine verwertbaren Aussagen über die Erfüllung der funktionalen Anforderungen getroffen werden, ist ein Prototyp zu entwickeln und zu testen. Dieser Mehraufwand ist vor allem bei riskanten Architekturentscheidungen der Klasse 2 (siehe S. 63) notwendig und gerechtfertigt, da eine Fehlentscheidung bei einem geschäftskritischen Softwaresystem in erheblichen negativen Konsequenzen resultiert und ein Rücksprung zu einer lauffähigen Version des Softwaresystems mit einem erheblichen Aufwand und entsprechend hohen Risiken verbunden ist.

Wird erkannt, dass einzelne funktionale Anforderungen nicht erfüllt werden, sind angemessene Gegenmaßnahmen festzulegen. Dabei werden die folgenden beiden Fälle unterschieden:

- *Veränderung funktionaler Systemeigenschaften kann verhindert werden:* Sofern die erkannten Probleme bei der nachgelagerten Implementierung des Lösungsansatzes gelöst werden können, bzw. eine Veränderung der funktionalen Systemeigenschaften verhindert werden kann, sind keine Korrekturen an der Maßnahmenfolge notwendig. Es erfolgt lediglich eine Dokumentation der betreffenden Systemeigenschaften und die Erstellung eines Lösungskonzeptes mit genauer Spezifikation und den entsprechenden Testfällen
- *Korrektur der Maßnahmenfolge notwendig:* Ist abzusehen, dass die Veränderung funktionaler Systemeigenschaften im Rahmen der Implementierung des Lösungsansatzes nicht verhindert werden kann, ist eine Korrektur der Maßnahmenfolge notwendig.

5.3.5 Konkretisierung und Anpassung bei mehrstufigen Entscheidungen

5.3.5.1 Aufstellung kombinierter Lösungsansätze

Das in den vorangegangenen Abschnitten beschriebene Verfahren zur Verfeinerung und Konkretisierung fokussiert auf die alternativen Lösungsansätze *einer* Entscheidungsstufe. Nicht immer jedoch können die Ziele mit einer Entscheidungsstufe im erforderlichen Umfang erfüllt werden. In der Praxis sind oft mehrstufige Architekturentscheidungen notwendig, um weitreichende Architekturveränderungen zu planen und um damit übergeordnete bzw. mitunter strategische Ziele erfüllen zu können (siehe Abschnitt 3.4, S. 48ff.). Dabei wird die Architekturentscheidung anhand eines Grobplans (siehe Abschnitt 5.3.1, S. 67ff.) in mehrere Stufen aufgeteilt. Für jede Stufe wird dabei separat und individuell ermittelt, welche alternativen Lösungsansätze zur Erreichung der im Grobplan festgelegten Zwischenziele geeignet sind, und es erfolgt eine separate Konkretisierung sowie Verfeinerung der Lösungsansätze.

In Abb. 5.12 ist ein Beispiel zur Verbesserung der Sicherheit eines E-Banking-Systems mittels einer mehrstufigen Architekturentscheidung dargestellt. In der ersten Entscheidungsstufe (in Abbildung links) werden zwei 'Firewall'-Lösungsansätze ausgewählt und konkretisiert. Daran schließt sich die zweite Stufe an, in der zwei 'Trustcenter'-Lösungsansätze ausgewählt und konkretisiert werden; da die 'Firewall A' bereits ein Trustcenter beinhaltet, werden nur im Anschluss an die 'Firewall B' die zwei Trustcenter-Varianten überprüft. In der dritten Entscheidungsstufe werden zwei alternative Zugriffsverfahren in den unterschiedlichen Kombinationen zu 'Firewall A und B' sowie in Kombination zu den beiden 'Trustcentern A und B' ausgewählt und konkretisiert. Jede Kombination von Lösungsansätzen hat dabei unterschiedliche Eigenschaften hinsichtlich der Verbesserung der Sicherheit des E-Banking-Systems und des Implementierungsaufwands. So können beispielsweise 'Firewall A', 'Trustcenter B' und 'Zugriffsverfahren A' die maximale Sicherheit bieten, aber auch den höchsten Implementierungsaufwand erfordern. Die anderen Kombinationen von Lösungsansätzen besitzen hingegen andere Eigenschaften. Das Ziel der Entscheidungsfindung ist es, die unterschiedlichen Eigenschaften der Lösungsansätze zu ermitteln und die verschiedenen Kombinationen ausgewogen und systematisch zu vergleichen, um eine Entscheidung nach rationalen Gesichtspunkten treffen zu können.

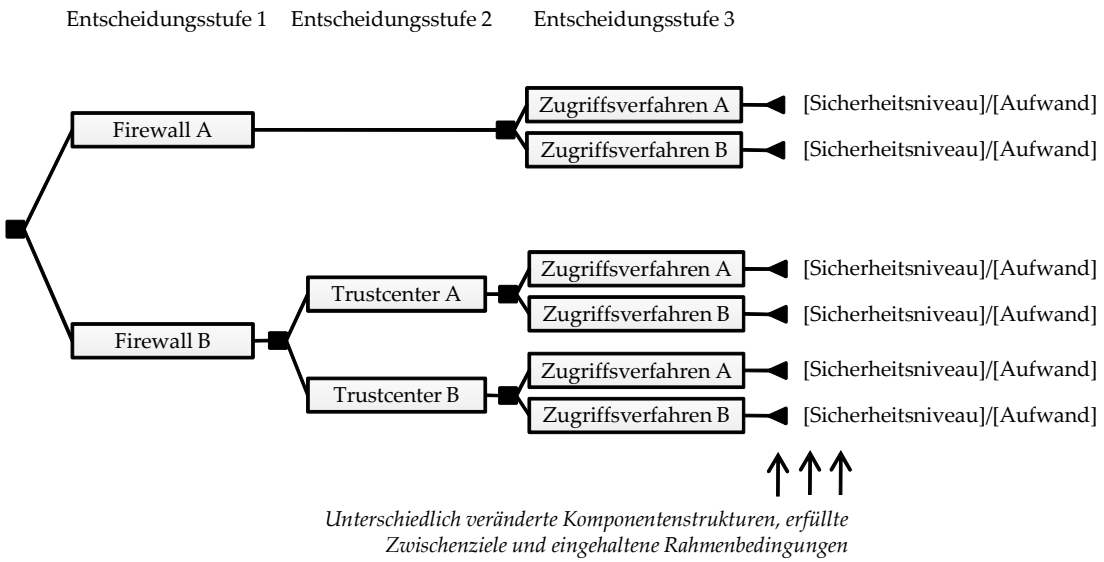


Abb. 5.12: Aufstellung der Kombinationen von Lösungsansätzen

Sofern im Grobplan (siehe Abschnitt 5.3.1, S. 67) zwei oder mehr Lösungsszenarien zur Erfüllung der Ziele zur Verfügung stehen, die sich beispielsweise in grundlegenden Kontroll- und Datenflussverfahren stark unterscheiden, sind entweder zwei getrennte Entscheidungsbäume aufzubauen oder es wird ein Entscheidungsbaum verwendet, der eine sog. Verfah-

rensentscheidung beinhaltet. Die Verfahrensentscheidung dient ausschließlich zur Kennzeichnung und Verdeutlichung der Unterschiede zwischen den Lösungsszenarien. In Abb. 5.13 ist ein Beispiel für eine Verfahrensentscheidung dargestellt, bei der es neben dem in Abb. 5.12 aufgezeigten Verfahren zur Verbesserung der Sicherheit eines E-Banking-Systems noch ein weiteres 'Verfahren B' gibt. Um beide Lösungsszenarien in einem Entscheidungsbaum darstellen zu können, wird eine Verfahrensentscheidung vorangestellt.

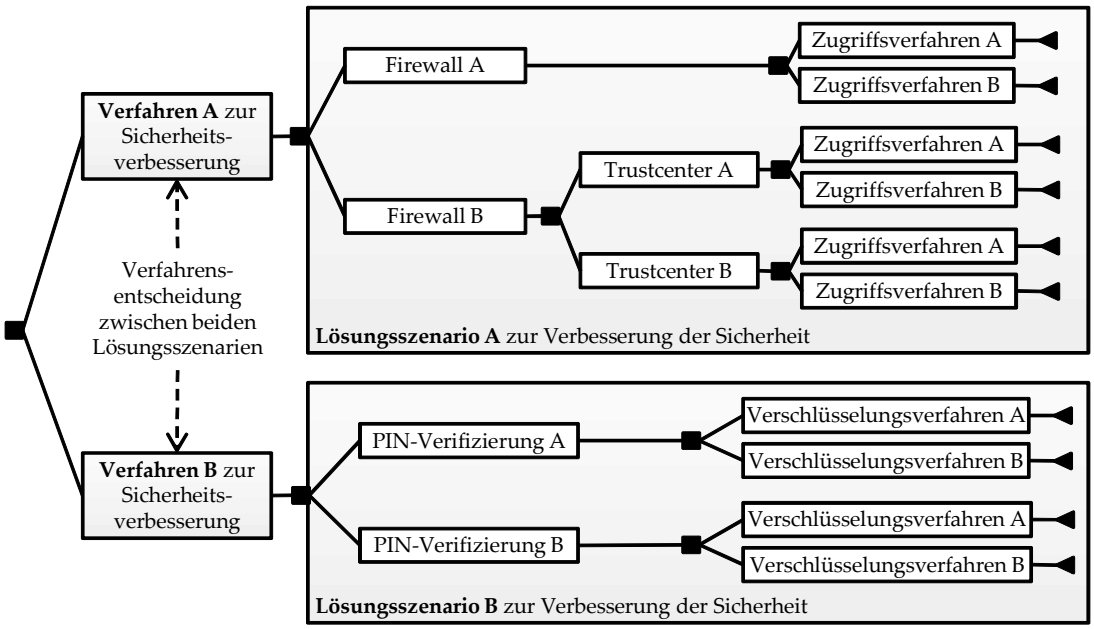


Abb. 5.13: Verfahrensentscheidung bei zwei Lösungsszenarien für ein Entscheidungsproblem

5.3.5.2 Berücksichtigung von Abhängigkeitsbeziehungen

Gerade bei mehrstufigen Architekturentscheidungen können Abhängigkeitsbeziehungen zwischen den aufeinander folgenden Entscheidungsstufen existieren. In der Praxis ist häufig zu beobachten, dass eine bestimmte Reihenfolge zwischen einzelnen Lösungsansätzen 'erzwungen' ist oder erst der Einsatz eines Lösungsansatzes A den Einsatz eines Lösungsansatzes B erst 'ermöglicht' (siehe zu den Beziehungsarten Abschnitt 4.1, S. 51ff.). Zudem können die Lösungsansätze der verschiedenen Entscheidungsstufen in Konflikt miteinander stehen und sich gegenseitig 'verhindern'. Es ist daher bei der Auswahl von Lösungsansätzen für jede Entscheidungsstufe erforderlich, die Abhängigkeitsbeziehungen zu berücksichtigen und im Falle von Konflikten aufzulösen. Sofern die Konflikte nicht durch den Einsatz eines alternativen Lösungsansatzes aufgelöst werden können, ist zu analysieren, in wie weit die Reihenfolgeplanung der Maßnahmenfolgen im Grobplan (siehe Abschnitt 5.3.1, S. 67ff.) angepasst werden kann.

Neben den Beziehungen zwischen den Entscheidungsstufen ist bei mehrstufigen Architekturentscheidungen zu berücksichtigen, dass die Architektur schrittweise weiterentwickelt und verändert wird. Jeder Lösungsansatz einer vorangegangenen Entscheidungsstufe verändert die Architektur in einer bestimmten Art und Weise. Daher ist bei der Auswahl und der Konkretisierung in einer höheren Entscheidungsstufe immer zu berücksichtigen, welche Veränderungen an der Architektur bereits vorgenommen wurden. Die unterschiedlichen Architekturveränderungen können durch einen direkten Vergleich der prototypisch veränderten Architekturmodelle ermittelt werden, die für jeden Lösungsansatz einer vorangegangenen Entscheidungsstufe erstellt wurden.

In Abb. 5.14 (siehe S. 83) ist für das E-Banking-Beispiel aus Abb. 5.12 (S. 81) dargestellt, dass ein angepasstes Vorgehen bei der Implementierung des Zugriffsverfahrens erforderlich ist (dritte Stufe der Entscheidung). Die Ursache liegt darin, dass

mit den Lösungsansätzen 'Firewall A und B' sowie den 'Trustcentern A und B' die Architektur in unterschiedlicher Art und Weise verändert wird:

- So werden die 'Komponenten A und B' (dargestellt als Viereck mit den jeweiligen Buchstaben) durch die Lösungsansätze 'Firewall A und B' verändert.
- Die 'Komponente C' wird außerdem durch die 'Firewall A' und zusammen mit der 'Komponente D' durch die Implementierung des 'Trustcenter A' verändert.
- Die 'Komponenten E und F' werden ausschließlich durch die Implementierung des 'Trustcenter B' verändert.

Mithilfe dieser Abhängigkeitsbeziehungen ('verändert durch', siehe S. 51ff.) kann ein individuelles Vorgehen für die Auswahl, Verfeinerung und Konkretisierung der Lösungsansätze in der dritten Stufe der Entscheidung erarbeitet werden. So setzen die 'Zugriffsverfahren A und B' auf einen unterschiedlichen Stand der Weiterentwicklung und Veränderung der Architektur auf. Bei der Verfeinerung und Konkretisierung der Lösungsansätze können u. a. unterschiedliche Szenarien erstellt und überprüft oder ergänzende Maßnahmen definiert werden, die aufgrund der unterschiedlichen Architekturveränderungen erforderlich sind.

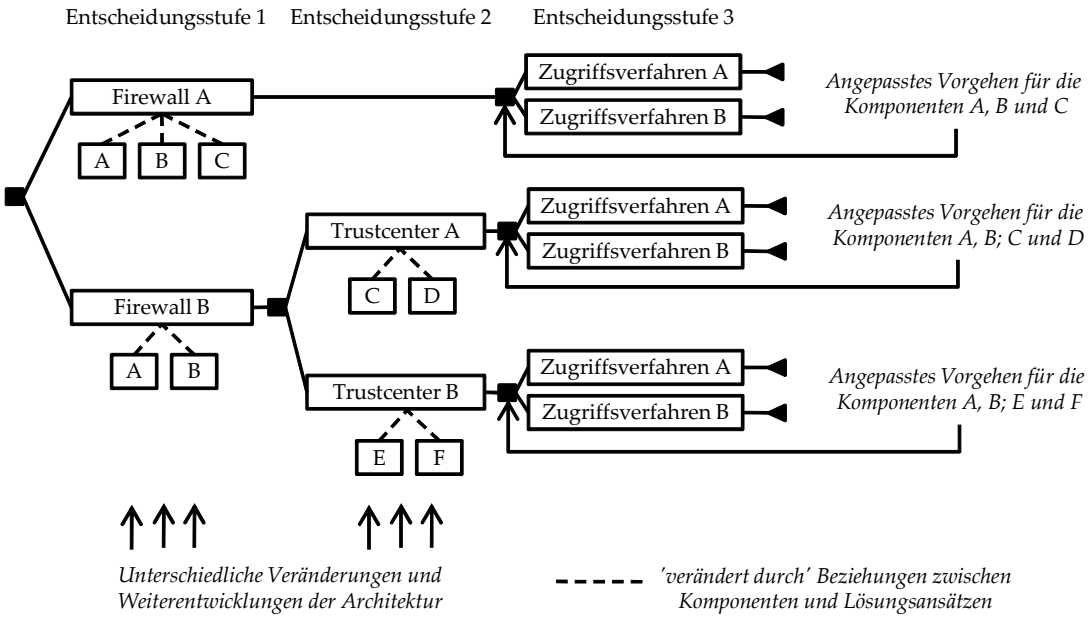


Abb. 5.14: Angepasstes Vorgehen bei der Auswahl und Konkretisierung in der dritten Stufe

5.3.5.3 Dokumentation des Entscheidungsverlaufes

Da mit jeder Stufe der Entscheidungsfindung verschiedene Architekturveränderungen vorgenommen werden, die in unterschiedlicher Art und Weise die Ziele und Rahmenbedingungen erfüllen, kann es erforderlich werden, den Entscheidungsverlauf zu dokumentieren. Das ist insbesondere bei riskanten und sehr riskanten Entscheidungen der Klassen 2 und 3 in Erwägung zu ziehen (siehe Abschnitt 5.2.1, S. 63), da aus dem Entscheidungsbaum heraus allein nicht genügend Informationen über den Fortschritt der Weiterentwicklung sowie Veränderung der Architektur und insbesondere über die Abhängigkeitsbeziehungen zwischen den Lösungsansätzen erkennbar sind. Da bei diesen riskanten und sehr riskanten Entscheidungen ein Rücksprung zu einer lauffähigen Version des Softwaresystems nur mit erheblichem Aufwand möglich ist, werden höhere Anforderungen an die Dokumentation und die Nachvollziehbarkeit der Entscheidung gestellt. Mit der Dokumentation des Entscheidungsverlaufs können beispielsweise die Unsicherheiten reduziert werden, ob und in welchem Umfang bereits erstellte Szenarien wiederverwendet werden können oder welche Architekturveränderungen im Detail mit jeder einzelnen Entscheidungsstufe verbunden sind.

Die Dokumentation erfolgt anhand von speziellen Formularen (sog. Datenblättern, siehe Abb. 5.15 und Abb. 5.16, S. 85) Alle Eigenschaften eines Lösungsansatzes werden auf einem Formular eingetragen. Darauf sind neben der obligatorischen Bezeichnung des Lösungsansatzes die folgenden Informationen enthalten:

- *Hintergrund:* Im Entscheidungsverlauf wird ein Entscheidungsbaum erstellt, welcher sich hinter jedem einzelnen Lösungsansatz neu verzweigt. Hierbei wird notiert, warum der Lösungsansatz eingesetzt wird, z. B. da ein Lösungsansatz aus einer vorangegangenen Entscheidungsstufe dies 'erfordert' (siehe zu den Abhängigkeitsbeziehungen S. 51).
- *Maßnahmenfolge:* Einer der wichtigsten Punkte des Datenblattes ist die Beschreibung der Maßnahmen, die zur Implementierung des Lösungsansatzes erforderlich sind. Analog zu den Ausführungen im Abschnitt 5.3.4 (siehe S. 73ff.) wird die Maßnahmenfolge als Aktivitätsdiagramm dargestellt.
- *Eigenschaften*: Anhand der Maßnahmenfolge oder eines prototypischen Architekturmodells wird ermittelt, welche Eigenschaften die Lösungsansätze hinsichtlich des Grads der Zielerreichung und des Implementierungsaufwandes haben. Dabei wird einerseits überprüft, ob die organisatorischen und technischen Rahmenbedingungen eingehalten werden und andererseits, ob die Rahmenbedingungen die Eigenschaften der Lösungsansätze beeinflussen. Zudem wird dokumentiert, ob Veränderungen funktionaler Systemeigenschaften eintreten und welche Gegenmaßnahmen im Rahmen der Implementierung durchzuführen sind.

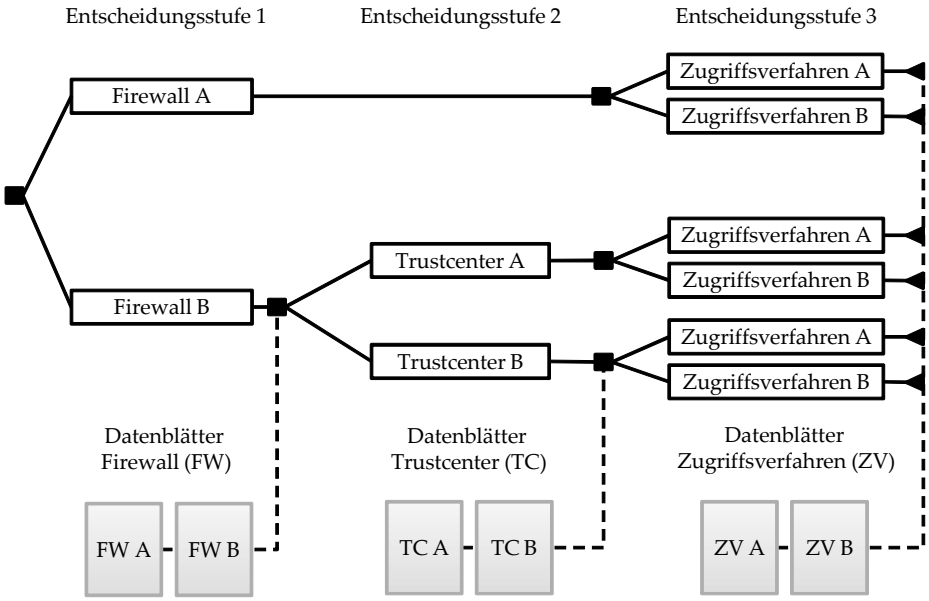
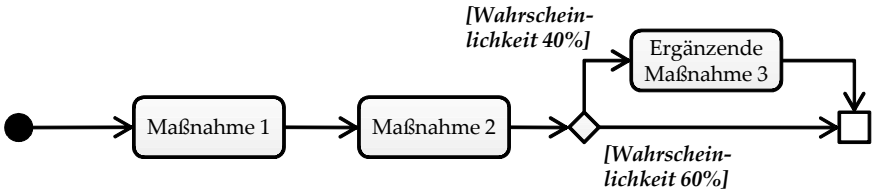


Abb. 5.15: Dokumentation des Entscheidungsverlaufes

Lösungsansatz: Trustcenter A

Hintergrund: Erfordert von Firewall B

Maßnahmenfolge:



Eigenschaften des Lösungsansatzes

Ziel: Verbesserung der Sicherheit	*Erfüllt zu 80%*
Implementierungs-aufwand	*30 Personentage*
Rahmenbedingungen	*Erfüllt; Aufgrund technischer Einschränkungen sind ergänzende Maßnahmen erforderlich*
Veränderungen am funktionalen Verhalten	*Mögliche Verhaltensänderungen, Korrektur bei Implementierung möglich*

Abb. 5.16: Beispiel eines Datenblattes eines Lösungsansatzes

5.4 Rationale Bewertung und Entscheidung über alternative Lösungsansätze

5.4.1 Bewertung der Lösungsansätze

Notwendigkeit einer systematischen Gesamtbewertung der Lösungsansätze

In der vierten und letzten Phase des Entscheidungsprozesses (siehe Abb. 3.1, S. 44) erfolgt eine systematische Gesamtbewertung der alternativen Lösungsansätze, wenn die Lösungsansätze aufgrund einer Vielzahl an Eigenschaften, Varianten und Wahrscheinlichkeiten nicht direkt miteinander verglichen werden können. Um eine Entscheidung nach rationalen Gesichtspunkten treffen zu können, sind die Lösungsansätze anhand ihrer Eigenschaftswerte systematisch zu vergleichen und zu bewerten. Der Entscheidungsträger muss klar und nachvollziehbar ermitteln können, welcher Lösungsansatz im Hinblick auf die Zielerreichung und die Einhaltung der Rahmenbedingungen am besten geeignet ist. Ein Vergleich der Lösungsansätze ist jedoch komplex und sehr riskant, wenn der Wert der Lösungsansätze nicht allein durch Addition der einzelnen Eigenschaftswerte ermittelt werden kann. Die Eigenschaftswerte können u. a. dann nicht zu einem Wert aufsummiert werden, wenn die Werte unterschiedlich skaliert, sowie Prioritäten bzw. wahrscheinliche oder unwahrscheinliche Varianten einer einzelnen Eigenschaft zu berücksichtigen sind (siehe Abschnitt 5.3.4.3, S. 76ff.). So kann z. B. die Bewertung eines Lösungsansatzes nicht durch Aufsummieren der Einzelwerte ermittelt werden, wenn der Grad der Verbesserung der Testbarkeit in Form von Prozentwerten angegeben ist, der Grad der Verbesserung der Verständlichkeit hingegen in Form von Schulnoten. Zudem kann der Fall eintreten, dass Varianten mit Wahrscheinlichkeit zu berücksichtigen sind und daher eine systematische Auswertung der Eigenschaftswerte erforderlich ist.

Um eine systematische, transparente und nachvollziehbare Bewertung der Lösungsansätze zu gewährleisten, stellt die Entscheidungstheorie ein Bewertungsverfahren zur Verfügung (siehe Abschnitt 2.9, S. 38ff.). Mit diesem Bewertungsverfahren können alternative Lösungsansätze im Hinblick auf ein oder mehrere Ziele – mit oder ohne die Einbeziehung von wahrscheinlichen Varianten – systematisch bewertet werden. Das Verfahren umfasst drei Schritte, die in den folgenden Abschnitten detailliert beschrieben werden:

- *Schritt 1:* Normalisierung der Eigenschaftswerte
- *Schritt 2:* Gewichtung nach Relevanz
- *Schritt 3:* Berücksichtigung wahrscheinlicher und unwahrscheinlicher Varianten

Im Hinblick auf ein ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis ist die Durchführung der abschließenden Gesamtbewertung der alternativen Lösungsansätze nur dann gerechtfertigt, wenn im Rahmen der Konkretisierung und Verfeinerung der Lösungsansätze (siehe Abschnitt 5.3.4, ab S. 73ff.) eine Vielzahl unterschiedlicher Eigenschaften berücksichtigt wurde. Zudem muss es sich um eine riskante oder sehr riskante Architekturentscheidung der Klasse 1 und 2 handeln (siehe S. 63), da ein Rücksprung zu einer lauffähigen Version des Softwaresystems mit erheblichem Aufwand verbunden ist.

Das Bewertungsverfahren fokussiert auf die beiden dominierenden Eigenschaften der Lösungsansätze: der Implementierungsaufwand und der Grad der Zielerreichung. Im tatsächlichen Entscheidungsfall können weitere Eigenschaften zu berücksichtigen sein, z. B. das Risiko eines Fehlschlages oder der Beitrag zur Komplexitätsminderung. Das Bewertungsverfahren kann bei diesen Eigenschaften analog angewendet werden.

Normalisierung der Eigenschaftswerte

Im ersten Schritt erfolgt die Normalisierung der unterschiedlich skalierten Eigenschaftswerte, u. a. die unterschiedlichen Ziel- oder Teilzielerreichungsgrade oder der Implementierungsaufwand. Das ist notwendig, damit den Lösungsansätzen ein *einheitlicher* Gesamtwert zugewiesen werden kann. Der Implementierungsaufwand wird meist in kardinaler Skalierung

ausgewiesen (z. B. in Stunden, Personentagen oder Kosten). Die erwarteten Ziel- oder Teilzielerreichungsgrade werden in der Praxis meist auf einer Ordinalskala angegeben, z. B. in Form von Noten. Zur Vereinheitlichung der unterschiedlichen Skalen erfolgt die Normalisierung der Eigenschaftswerte auf einer einheitlichen Skala im Intervall von null bis eins.

Die Normalisierung beginnt mit der Ermittlung der positiven und negativen Extremwerte. Dabei wird über alle im Rahmen der Entscheidungsfindung berücksichtigten Lösungsansätze ermittelt, welche positiven und negativen Extremwerte erreicht werden. Stehen z. B. drei Lösungsansätze zur Verfügung, werden über alle der höchste Zielerreichungsgrad (im Extremfall 100 % oder die Note 1) und der geringste Implementierungsaufwand bestimmt. Diesen positiven Extremwerten wird der Wert eins zugewiesen. Auf der anderen Seite werden der geringste Zielerreichungsgrad und der höchste Implementierungsaufwand ermittelt. Diesen negativen Extremwerten wird der Wert null zugewiesen.

Im Anschluss werden die Werte zwischen den Extremwerten auf das Intervall [0;1] anhand einer Wertfunktion normalisiert. Der Verlauf der Wertfunktion hängt davon ab, wie stark der Grad der Zielerfüllung ist bzw. welche Werte der Entscheidungsträger bevorzugt. Die Eigenschaftswerte, die kaum zur Zielerreichung beitragen, werden mit einem Wert nahe null umgeformt, z. B. ein sehr hoher Implementierungsaufwand. Im Gegensatz dazu werden jene Eigenschaftswerte, die eine hohe Zielerreichung repräsentieren, mit einem Wert nahe eins umgeformt, z. B. eine sehr hohe Qualitätsverbesserung. Der Verlauf der Wertfunktion bzw. die Art und Weise der Normalisierung der Werte muss nicht linear erfolgen, sondern kann auch progressiv oder degressiv sein:

- *Progressive Normalisierung:* Bevorzugt der Entscheidungsträger ausschließlich die sehr positiven Eigenschaftswerte, erfolgt die Normalisierung anhand einer progressiven Funktion. In Abb. 5.17 ist eine solche Normalisierungsfunktion an einem Beispiel für die Verbesserung der Wartbarkeit abgebildet. Der höchste mögliche Zielerreichungsgrad ist 95 %, der naturgemäß stark präferiert wird. Mögliche weitere Zielerreichungsgrade sind 60 % und 80 %. Diese präferiert der Entscheidungsträger in diesem Fall nur in geringem Maße. Beide Werte werden daher *unterproportional* mit 0.1 und 0.2 normalisiert.
- *Degressive Normalisierung:* Bevorzugt der Entscheidungsträger gerade die geringen Verbesserungen, erfolgt die Normalisierung mittels einer degressiven Funktion. In Abb. 5.17 ist sichtbar, dass auch die Zielerreichungsgrade 60 % und 80 % im Vergleich zur progressiven Normalisierung mit *überproportional* hohen Werten (0.5 und 0.7) umgeformt werden.
- *Lineare Normalisierung:* Besteht durch den Entscheidungsträger keine Bevorzugung bei den Eigenschaftswerten, werden die Werte über eine lineare Funktion normalisiert. In Abb. 5.17 ist dargestellt, dass die Zielerreichungsgrade 60 % und 80 % im *proportionalen* Verhältnis zueinander normalisiert sind (0.3 und 0.6).

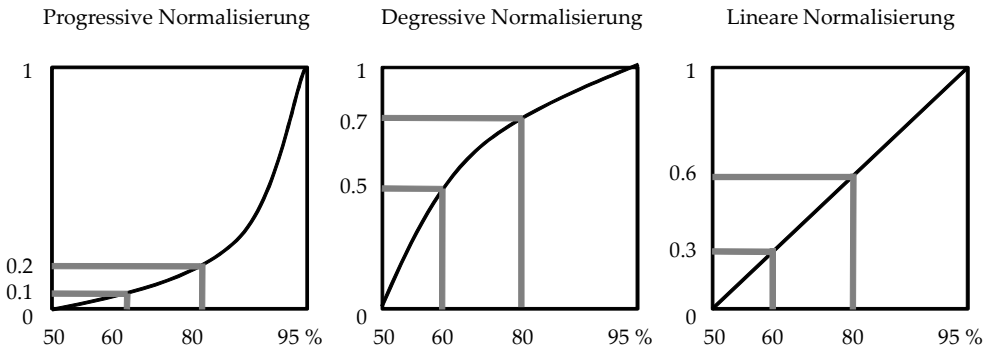


Abb. 5.17: Unterschiedliche Wertfunktionen für zwei Eigenschaftswerte

In der Entscheidungstheorie existieren verschiedene Verfahren, um die Wertfunktionen für den Entscheidungsträger zu ermitteln. Die beiden bekanntesten sind die 'Methode gleicher Wertdifferenzen' und das 'Direct-Ratio-Verfahren' [Fish67] (und [EiWe03]).

Gewichtung nach Relevanz

Im zweiten Schritt erfolgt die Gewichtung der Eigenschaftswerte der Lösungsansätze, die bereits in Werte zwischen null und eins umgeformt wurden. Das ist notwendig, da der Gesamtwert der Lösungsansätze mit zunehmender Anzahl zu berücksichtigender Eigenschaften *stetig* ansteigt. Je mehr Ziele und damit Eigenschaftswerte betrachtet werden, desto höher ist der rechnerische Wert der Lösungsansätze. Für eine Bewertung nach rationalen Gesichtspunkten ist jedoch ein diskreter Wert zwischen null und eins erforderlich [EiWe03]. Die normalisierten Eigenschaftswerte werden daher mit einem Gewichtungsfaktor multipliziert, welcher der Priorisierung der Ziele entspricht. Alle Gewichtungsfaktoren müssen aus Berechnungsgründen in der Summe den Wert eins ergeben. Für den Entscheidungsträger kann z. B. ein möglichst geringer Implementierungsaufwand eine doppelt so hohe Priorität wie die Verbesserung der Wartbarkeit besitzen. In diesem Fall haben die Gewichtungsfaktoren den Wert 0.67 und 0.33.

Sofern keine Varianten der Eigenschaftswerte zu berücksichtigen sind, kann der Gesamtwert der Lösungsansätze durch Addition der einzelnen Eigenschaftswerte ermittelt werden. Dies entspricht im Rahmen der Entscheidungstheorie als Bewertung unter Sicherheit [EiWe03]. Als Berechnungsmethode wird im Rahmen der vorliegenden Dissertation das 'Additive-Modell' eingesetzt (siehe Abschnitt 2.9, S. 38ff.). In Abb. 5.18 ist die entsprechende Bewertungsfunktion dargestellt. Es wird der Wert (Value) eines Lösungsansatzes a ermittelt, repräsentiert durch $v(a)$. Zur Berechnung werden die einzelnen, normalisierten Eigenschaftswerte $v(a_r)$ mit dem Gewichtungsfaktor w_r multipliziert und addiert. Die einzelnen Eigenschaftswerte sind indiziert mit r im Intervall von eins bis m.

$$v(a)=\sum_{r=1}^{m} w_r v(a_r) \quad .$$

Abb. 5.18: Bewertung unter Sicherheit

Berücksichtigung von wahrscheinlichen und unwahrscheinlichen Varianten

Im dritten Schritt der Bewertung werden jene Lösungsansätze einbezogen, bei denen bei einzelnen Eigenschaftswerten wahrscheinliche und unwahrscheinliche Varianten berücksichtigt und ausgewertet werden müssen. Dies wird im Rahmen der Entscheidungstheorie als Bewertung unter Unsicherheit verstanden [EiWe03]. Bei der Bewertung unter Unsicherheit wird der Mittelwert ermittelt, der mit der Implementierung des Lösungsansatzes erwartet werden kann (sog. Erwartungswert). Zur Berechnung werden die Eigenschaftswerte der Varianten mit den entsprechenden Wahrscheinlichkeiten multipliziert und addiert. Für einen Lösungsansatz mit z. B. drei Varianten A, B und C ist jeder Eigenschaftswert der Variante A, B oder C mit der entsprechenden Wahrscheinlichkeit zu multiplizieren und zum Erwartungswert zu addieren. Die Formel zur Ermittlung des Erwartungswertes (Expected-Value – EV) ist in Abb. 5.19 dargestellt (siehe Abschnitt 2.9, S. 22). Die normalisierten Eigenschaftswerte des Lösungsansatzes a, indiziert durch r im Intervall von eins bis m, werden mit den entsprechenden Gewichtungsfaktoren w_r multipliziert. Zusätzlich werden die verschiedenen Varianten, charakterisiert durch i im Intervall von eins bis n, mit den zugehörigen Wahrscheinlichkeiten p_i multipliziert.

$$EV(a)=\sum_{i=1}^{n} p_i[\sum_{r=1}^{m} w_r v(a_{ir})] \quad .$$

Abb. 5.19: Bewertung unter Unsicherheit

5.4.2 Interpretation des Bewertungsergebnisses und Entscheidung

Im Anschluss an die Entscheidungsfindung erfolgt die Entscheidung für den besten Lösungsansatz. Die Grundlage der Entscheidung sind die – erwarteten – Gesamtwerte der Lösungsansätze. Nach rationalen Gesichtspunkten erfolgt die Entscheidung für den besten bzw. für den am höchsten bewerteten Lösungsansatz.

Gerade bei minimalen Abweichungen zwischen den Lösungsansätzen ist zu prüfen, ob weitere Ziele, Faktoren oder Eigenschaften existieren, die bislang noch nicht in ausreichendem Maße in die Entscheidungsfindung mit einbezogen worden sind. Das Ergebnis spiegelt zwar die Eignung der Lösungsansätze und die Bevorzugung des Entscheidungsträgers nach rationalen Gesichtspunkten wieder. Jedoch sollte insbesondere bei minimalen Abweichungen zwischen den Eigenschaftswerten die Tatsache Berücksichtigung finden, dass aus ökonomischen Gründen nur die wichtigsten Ziele, Rahmenbedingungen, Einflussfaktoren und Eigenschaftswerte im Rahmen der Entscheidungsfindung betrachtet werden.

Kapitel 6

Exemplarische Architekturentscheidungen

6.1 Verbesserung der Portabilität des Web-Content-Management-Systems Typo3

In den Abschnitten 6.1 und 6.2 (siehe S. 118ff.) wird der entwickelte Entscheidungsprozess an zwei Praxisbeispielen prototypisch angewendet. Das erste Beispiel ist eine Architekturveränderung des Softwareproduktes Typo3. Typo3 ist ein Web-Content-Management-System (WCMS), das eingesetzt wird, um die Inhalte von Webseiten möglichst flexibel gestalten und administrieren zu können. In der Praxis wird Typo3 insbesondere dann eingesetzt, wenn viele Autoren an einer Webseite mit einheitlichem Layout arbeiten. Die Gestaltung der Webseiten sowie die Administration und Konfiguration erfolgen ausschließlich über eine Weboberfläche. Typo3 ist eine PHP-Anwendung mit modularer Architektur und kann durch ca. 3000 sog. 'Extensions' erweitert sowie ergänzt werden. Die freie Verfügbarkeit des Open-Source-Softwaresystems Typo3 führte zu einer hohen Akzeptanz und Nutzung in der Praxis. Als WCMS wird Typo3 derzeit bei ca. 4700 Webseiten eingesetzt; beispielsweise bei der TU-Ilmenau [Spr+04] oder bei der internationalen Hotelkette Maritim [Mrtm07].

In Abb. 6.1 ist ein Screenshot der Weboberfläche von Typo3 dargestellt. Über die Weboberfläche werden die 'Inhalte einer Webseite' eingegeben (rechts unten), formatiert und nach Aufruf des Befehls 'Speichern' (mittig oben) in eine MySQL-Datenbank geschrieben.

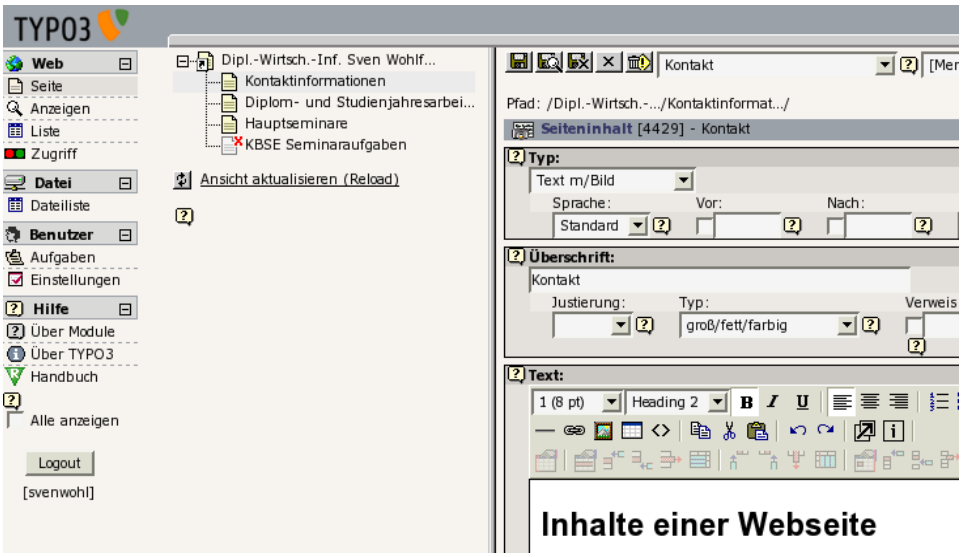


Abb. 6.1: Verwaltung von Inhalten einer Webseite mittels Typo3

Ein zentrales Problem von Typo3 in der Version 3.8.1 ist die eingeschränkte Portabilität, da ausschließlich MySQL-Datenbanken als zentrales Speichermedium eingesetzt werden können. Gerade im Unternehmensumfeld ist jedoch der Einsatz von MySQL-Datenbanken aus Sicherheits- oder Lizenzgründen häufig nicht möglich. Damit Typo3 mit anderen Datenbanksystemen, z. B. Oracle oder DB2, zusammenarbeiten kann, sind strukturelle Veränderungen am Datenbankzugriff notwendig. Da diese Veränderungen einen Eingriff in die Kern-Funktionalität von Typo3 darstellen, können Fehlentscheidungen gravierende negative Auswirkungen auf die Benutzbarkeit von Typo3 und die Zugriffszeiten auf die Datenbank haben. Eine Beeinträchtigung der Benutzbarkeit aufgrund massiver Verschlechterungen an den Zugriffszeiten stellt gerade für

Unternehmen ein großes Risiko dar, da ein Wechsel des WCMS mit einem hohen Migrationsaufwand verbunden ist. Zudem existieren Unsicherheiten hinsichtlich unbekannter Abhängigkeitsbeziehungen zwischen den vielzähligen 'Extensions' und der MySQL-Datenbank.

Zur Reduzierung der mit der Architekturveränderung verbundenen Risiken und Unsicherheiten wird mithilfe des entwickelten Entscheidungsprozesses nach rationalen Gesichtspunkten ermittelt, welche alternativen Lösungsansätze zur Restrukturierung des Datenbankzugriffs am besten geeignet sind. Die strukturierte und systematische Entscheidungsfindung ist zwar mit einem zusätzlichen Analyse- und Bewertungsaufwand verbunden. Der ist jedoch gerechtfertigt, da u. a. die Abhängigkeitsbeziehungen systematisch identifiziert und berücksichtigt werden können sowie die Überprüfung der Eignung der alternativen Lösungsansätze nach objektiven Kriterien erfolgt. Somit kann in frühen Phasen ermittelt werden, in welchem Grad die Lösungsansätze die Zugriffszeiten beeinträchtigen sowie welche Risiken und Einschränkungen mit den Lösungsansätzen jeweils verbunden sind. In jeder Phase des Entscheidungsprozesses ist jedoch kritisch zu hinterfragen, welche Analyse- und Bewertungsschritte im Hinblick auf die Risiken, die Komplexität und die Unsicherheiten der Architekturveränderung sinnvoll und gerechtfertigt sind.

6.1.1 Phase 1 - Identifizierung der Ziele und Rahmenbedingungen

Identifizierung der Ziele

In der ersten Phase der Entscheidungsfindung (siehe Abb. 3.1 auf S. 44) über die Typo3-Architekturveränderung sind die Ziele systematisch zu identifizieren und widerspruchsfrei zu strukturieren. Das ist erforderlich, um durch die klare, eindeutige und detaillierte Ziel-Beschreibung 'sich ergänzende' oder 'miteinander konkurrierende' Zielbeziehungen erkennen zu können (siehe zu den Zielbeziehungen Abschnitt 4.1, S. 51ff.). Neben der Reduzierung von Komplexität, Risiken und Unsicherheiten ist die klare und detaillierte Ziel-Beschreibung die Grundlage für eine systematische Auswahl der besten Lösungsansätze zur Restrukturierung des Datenbankzugriffs von Typo3. Im weiteren Verlauf des Abschnitts werden die Ziele durch Unterstreichung hervorgehoben.

Das zentrale Ziel der Typo3-Architekturveränderung ist die Restrukturierung des Datenbankzugriffs zur Verbesserung der Portabilität, da in Typo3 (Version 3.8.1) ausschließlich auf MySQL-Datenbanken zugegriffen werden kann. Eine erste Architekturanalyse auf Basis der Dokumentation und des Quelltextes von Typo3 (siehe [Skrh06] oder [Damb06b]) zeigt, dass die Datenbankabfragen über native Datenbank-Funktionen der Skriptsprache PHP erfolgen, z. B. mittels 'mysql_query(SQL-Statement)' (in Listing 6.1 hervorgehoben durch Fettdruck). Aufgrund dieser MySQL-spezifischen PHP-Funktionen sind Zugriffe auf alternative Datenbanksysteme wie Oracle oder DB2 nicht möglich. Dies schränkt die Portabilität von Typo3 ein, da gerade in Unternehmensumgebungen MySQL-Datenbanken aus Sicherheits- oder Lizenzgründen häufig nicht eingesetzt werden können. Da der Sourcecode des Open-Source-Softwaresystems Typo3 frei verfügbar ist und eine gute Dokumentation vorliegt, kann zur Identifizierung der Schwachstellen sowie zur Verfeinerung der Ziele auf eine szenariobasierte Architekturanalyse verzichtet werden.

Die Datenbankzugriffe erfolgen über eine Datenbank-Abstraktionsschicht, implementiert als Bibliothek 'T3lib_db', welche die Zugriffsmethoden und Abfragemasken auf die Datenbank kapselt. Eine schematische Darstellung der vorhandenen Datenbank-Abstraktionsschicht ist in Abb. 6.2 wiedergegeben. Auf der linken Seite ist der Ausgangszustand dargestellt: Zwei Komponenten greifen über die Abstraktionsschicht 'T3lib_db' auf die MySQL-Datenbank zu. Auf der rechten Seite ist der Zielzustand abgebildet, in dem die Komponenten über eine erweiterte und ergänzte Abstraktionsschicht auf verschiedene Datenbanksysteme zugreifen können.

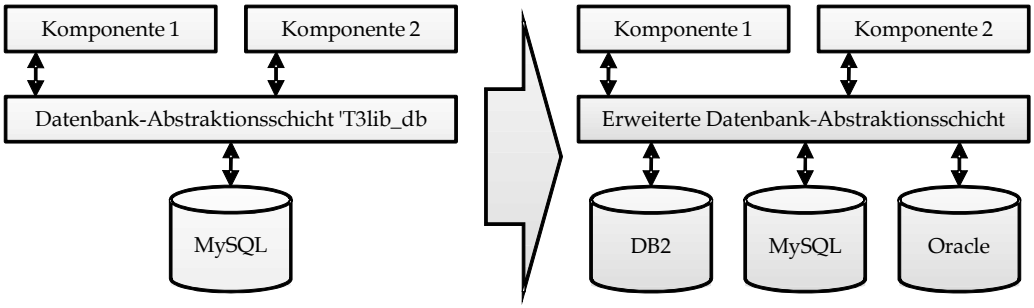


Abb. 6.2: Schematische Darstellung der Typo3-Architekturveränderung

Der Datenbankzugriff innerhalb der Datenbank-Abstraktionsschicht 'T3lib_db' erfolgt über die Funktion 'exec_INSERTquery'. In Listing 6.1 ist der Quelltext der Funktion 'exec_INSERTquery' dargestellt. Bei Aufruf wird durch die Funktion 'INSERTquery' aus den Parametern '$table' und '$table_values' ein SQL-Statement generiert (hervorgehoben durch Unterstreichung), mit dem der Tabellenname und die darin einzufügenden Daten übergeben werden. Durch die native PHP-Funktion 'mysql_query' wird mit dem generierten SQL-Statement die MySQL-Datenbank abgefragt. Der Quelltext der Funktion 'INSERTquery', mit der die Generierung des SQL-Statements erfolgt, ist in Listing 10.1 (Anhang 10.4, S. 176) dargestellt.

```
function exec_INSERTquery($table,$fields_values)  {
  $res = mysql_query($this->INSERTquery($table,$fields_values),
                     $this->link);
  if ($this->debugOutput) $this->debug('exec_INSERTquery');
  return $res;
 }
```

Listing 6.1: Quelltext der Funktion 'exec_INSERTquery'

Bei der Erweiterung der Abstraktionsschicht ist eine lose Kopplung zwischen der Datenbank-Abstraktionsschicht und den zugreifenden Komponenten einzuhalten, um den Implementierungsaufwand sowie den Analyse- und Testaufwand bei zukünftigen Wartungsaktivitäten zu reduzieren. Die lose Kopplung ist aktuell dadurch gegeben, dass die Komponenten lediglich Parameter an die Zugriffsfunktionen (u. a. die 'exec_INSERTquery') übergeben und daraus ein SQL-Statement generiert wird. Dieser Ablauf des Datenbankzugriffes soll auch nach der Architekturveränderung erhalten bleiben. Zu vermeiden ist es beispielsweise, dass aus den 'Extensions' heraus direkte Datenbankzugriffe an der Datenbank-Abstraktionsschicht vorbei erfolgen.

Ein konkurrierendes Ziel stellen die Zugriffszeiten dar. Die für die Generierung der Webseiten notwendigen Datenbankzugriffe sind der kritische Punkt für die Ladezeiten und damit für die Benutzbarkeit der Webseite insgesamt. Die Anzahl der Datenbankabfragen, die für das Laden einer Webseite erforderlich sind, hängt von der Komplexität und den Inhalten der Webseite ab. So sind beispielsweise für das Laden einer einfachen 'Hello-World'-Webseite ca. 30 Datenbankabfragen erforderlich; die Erfahrung zeigt, dass mit zunehmender Komplexität die Anzahl der Datenbankzugriffe meist exponentiell ansteigt. Aufgrund der Vielzahl an Datenbankabfragen führen selbst minimale Verzögerungen zu spürbar verschlechterten Ladezeiten und zu Performance-Problemen. Daher ist bei der Erweiterung der Datenbank-Abstraktionsschicht auf solche Veränderungen zu verzichten, die sich negativ auf die Datenbank-Zugriffszeiten auswirken.

Neben der Portabilität, der Wartbarkeit und den Zugriffszeiten darf die Funktionalität für den Zugriff auf die Datenbank und die Abfrage der Daten nicht verändert werden. Das ist notwendig, um eine Kompatibilität zu früheren Typo3-Versionen zu ermöglichen, insbesondere im Hinblick auf die Vielzahl an bereits existierenden 'Extensions'. Nach erfolgter Restrukturierung des Datenbankzugriffs sollen die 'Extensions' in unveränderter Art und Weise auf die Datenbank zugreifen können. Zudem ist der Implementierungsaufwand so gering wie möglich zu halten.

Strukturierung der Ziele

Im Anschluss an die Identifizierung der für die Typo3-Architekturveränderung relevanten Ziele erfolgt deren widerspruchsfreie Strukturierung, um Abhängigkeiten zwischen den Zielen erkennen und Widersprüche auflösen zu können (siehe zu den unterschiedlichen Beziehungsarten Abschnitt 4.1 auf S. 51ff.). Dies ist ein wichtiger Beitrag zur Senkung der Komplexität, der Unsicherheiten und der Risiken für die Typo3-Architekturveränderung. Das Verfahren zur widerspruchsfreien Strukturierung der Ziele umfasst die folgenden vier Schritte:

- *Schritt 1 – Verfeinerung grober und ungenauer Ziele:* Um die Ziele als Entscheidungskriterien einsetzen zu können, sind keine weiteren Verfeinerungen notwendig. Die Ziele sind bereits ausreichend genau beschrieben, da zur Analyse der Schwachstellen und der Problembereiche auf den vollständigen Quelltext von Typo3 sowie auf eine umfassende Dokumentation zurückgegriffen werden konnte.
- *Schritt 2 – Identifizierung sich ergänzender Ziel-Mittel-Beziehungen:* Zur Identifizierung der Ziel-Mittel-Beziehungen werden die Ziele unter Anwendung der Klassifikationsmethode aus der Entscheidungstheorie nach Instrumental- und Fundamentalzielen klassifiziert (siehe Abschnitt 2.3 auf S. 28 und die zusammenfassende Darstellung in Form des Ziel-Wirkungsmodells in Abb. 6.3 auf S. 94). Das wichtigste Fundamentalziel ist die Verbesserung der 'Portabilität'. Hierfür ist eine 'Restrukturierung des Datenbankzugriffs' erforderlich (erstes Instrumentalziel). Eine erste Grobanalyse zeigte, dass es sinnvoll ist, zur Restrukturierung die existierende 'Abstraktionsschicht zu erweitern' (zweites Instrumentalziel). Zur Reduzierung des Implementierungsaufwandes ist die Erweiterung so zu gestalten, dass eine 'lose

Kopplung' der Typo3-Komponenten und der Datenbankzugriffe ermöglicht wird (drittes Instrumentalziel). Die lose Kopplung ist darüber hinaus ein Instrumentalziel für die zukünftige 'Wartbarkeit' des Datenbankzugriffs innerhalb von Typo3. Die Wartbarkeit hat im Vergleich zu den anderen Fundamentalzielen eine geringere Priorität (hervorgehoben durch eine hellere Einfärbung).

- *Schritt 3 – Auflösung widersprüchlicher und konkurrierender Zielbeziehungen:* Das konkurrierende Ziel 'Zugriffszeiten' ist im Vergleich zur 'Portabilität' von geringer Priorität. Es wird dahingehend eingeschränkt, dass die 'Antwortzeiten' durch die Architekturveränderung nicht verschlechtert werden dürfen. Mögliche Optimierungsmöglichkeiten von Drittanbietern (z. B. von Zend [Arc+06]) sollen bei der aktuellen Architekturentscheidung zunächst unberücksichtigt bleiben und sind Gegenstand einer nachfolgenden Entscheidung.
- *Schritt 4 – Beschränkung auf die wichtigsten Ziele:* Da im Rahmen der Entscheidung lediglich drei Fundamentalziele zu berücksichtigen sind, ist eine Einschränkung der Ziele zur Reduzierung der kombinatorischen Komplexität nicht notwendig.

In Abb. 6.3 sind die strukturierten Ziele der Typo3-Architekturveränderung in Form eines Ziel-Wirkungsmodells dargestellt (zur Verbesserung der Übersichtlichkeit sind die Fundamentalziele durch eine graue Schattierung hervorgehoben). Die Instrumental- und Fundamentalziele stehen einerseits in einer 'sich ergänzend'-Beziehung zueinander. Speziell zwischen der Portabilität und den Zugriffszeiten existiert eine 'miteinander konkurrierend'-Beziehung.

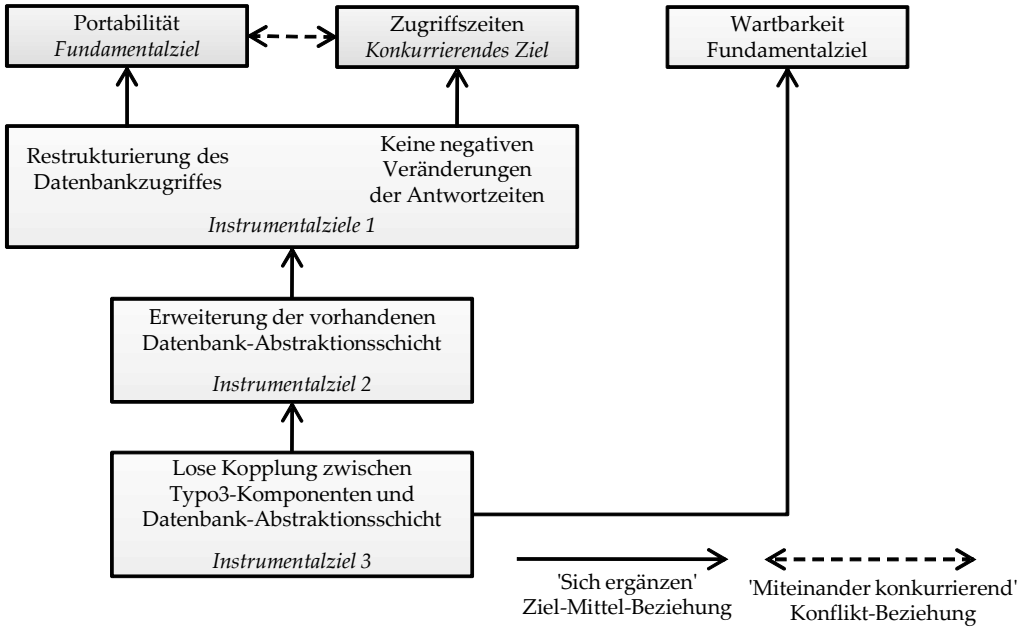


Abb. 6.3: Ziel-Wirkungsmodell der Typo3-Architekturveränderung

Identifizierung und Erhebung organisatorischer und technischer Rahmenbedingungen

Neben den Zielen stellen die organisatorischen sowie die technischen Rahmenbedingungen wichtige Kriterien für die Vorauswahl und Konkretisierung der Lösungsansätze dar (siehe Abschnitt 5.3, S. 66ff.). So können aus organisatorischer Sicht nur jene Lösungsansätze ausgewählt werden, die u. a. mit dem zur Verfügung stehenden Budget sowie mit der verfügbaren Menge an Entwicklern umgesetzt werden können. Darüber hinaus sind technische Rahmenbedingungen und Architekturvorgaben zu berücksichtigen, wie z. B. eine definierte Schichtenarchitektur oder ein festgelegtes Schnittstellendesign.

Da die Rahmenbedingungen der Typo3-Architekturveränderung 'miteinander konkurrieren' oder 'sich ergänzen' (siehe zu den Zielbeziehungen Abschnitt 4.1, S. 51ff.), ist ein systematisches Vorgehen für die Identifizierung und Erhebung erfor-

derlich. Das Ziel ist es, die relevanten Rahmenbedingungen zu identifizieren, Konflikte aufzulösen und durch eine widerspruchsfreie Strukturierung die kombinatorische Komplexität und die daraus resultierenden Unsicherheiten reduzieren zu können. Die Identifizierung und Strukturierung erfolgen anhand der in Abschnitt 5.1.3 (S. 59ff.) vorgestellten Checkliste der typischen Rahmenbedingungen, die bei Architekturveränderungen zu beachten sind.

- *Organisatorische Rahmenbedingungen*
  - *Managementvorgaben:* Typo3 ist unter der GPL lizenziert und steht damit als Open-Source-Anwendung zur Verfügung. Typo3 wird daher von einer Gemeinschaft entwickelt und gewartet. Aufgrund der fehlenden Weisungsgebundenheit der weltweit agierenden Entwickler wurde ein Leitfaden (Typo3-Coding-Guidelines [Skrh06b]) definiert, in dem neben einer Vielzahl an technischen Vorgaben auch Verantwortlichkeiten, z. B. für das Testen sowie die Freigaben neuer Komponenten und Funktionalitäten, festgelegt sind.
  - *Personelle Kapazität:* Wie alle Open-Source-Anwendungen sind die Entwickler von Typo3 räumlich verteilt – hauptsächlich im europäischen Raum. Für die geplante Architekturveränderung kann nur auf einen kleinen Teil der Entwickler zurückgegriffen werden, da ansonsten der Aufwand zur Versionierung und zur Abstimmung zu groß wird. Die Umsetzung der Veränderung sollte 70 Personentage nicht überschreiten.
  - *Finanzieller Spielraum:* Aufgrund des geringen finanziellen Spielraums werden wird eine Eigenentwicklung oder die Nutzung von Open-Source Softwarekomponenten favorisiert. Zusätzlich zur reinen Kostenbetrachtung sollte der eingesetzte personelle Aufwand in einem ausgeglichenen Verhältnis zum erwarteten Nutzen der Architekturveränderung stehen.
- *Technische Rahmenbedingungen:*
  - *Hardware:* Die Hardware-Anforderungen von Typo3 steigen mit der Anzahl der Nutzer der Webseite stetig an. Für eine minimale Grundkonfiguration von Typo3 sind laut Eigenangaben eine "modern CPU and at least 256 MB of RAM" [TReq06] notwendig. Die Erfahrungen aus der Praxis zeigen jedoch, dass mindestens ein Zweiprozessorsystem mit 2 GB Arbeitsspeicher erforderlich ist.
  - *Softwaretechnologien:* Aufgrund der Plattformunabhängigkeit der Skriptsprache PHP kann Typo3 auf jedem Betriebssystem ausgeführt werden, für das ein Webserver mit PHP-Interpreter verfügbar ist [TReq06]. Dies ist, mit wenigen Ausnahmen, bei allen aktuell verfügbaren Webservern der Fall [Arc+06], wie z. B. bei den Webservern Apache oder dem Internet Information Server (IIS).
  - *Architekturtechnologien*: Typo3 ist als Web-Anwendung in eine Client-Server-Struktur aufgeteilt. Ein Client ruft über den Browser die Webseiten auf, die Typo3 als Server generiert. Die Datenbank kann entweder zusätzlich auf demselben physischen/virtuellen Server installiert sein (lokaler Zugriff) oder es existieren separate Datenbankserver (entfernter, evtl. verteilter Zugriff). Innerhalb von Typo3 ist speziell für den Datenbankzugriff eine Schichtenarchitektur definiert (siehe vorangegangen Abschnitt und insbesondere Abb. 6.2 auf S. 92).
  - *Standards:* Bei der Architekturveränderung sind die Typo3-Coding-Guidelines [Skrh06b] einzuhalten. Zudem ist eine Abwärtskompatibilität zu früheren Typo3-Versionen einzuhalten. Ein bereits existierender Datenbestand muss nach der Restrukturierung des Datenbankzugriffs in identischer Art und Weise weiterverwendet werden können.
  - *Werkzeugumgebung*: Typo3 ist mit der Open-Source-Skriptsprache PHP implementiert. Zur Durchführung der Architekturentwicklung bzw. -veränderung stehen Entwicklungsumgebungen und Compiler – auch Open-Source basierte – in ausreichender Zahl zur Verfügung [Arc+06].

6.1.2 Phase 2 – Architekturbeschreibung und Risikoeinstufung

In der zweiten Phase des Entscheidungsprozesses (siehe Abb. 3.1 auf S. 44) erfolgt eine Beschreibung der relevanten Teile der Typo3-Architektur, um die Problembereiche und Schwachstellen klar und eindeutig zu beschreiben sowie die Abhängigkeiten zwischen den Typo3-Komponenten und der Datenbank erkennen zu können. Entsprechend den Ausführungen in Abschnitt 5.2.2 (siehe ab S. 63f.) ist das Verfahren zur Architekturbeschreibung in drei Schritte aufgeteilt.

Einstufung des Risikos einer Architekturentscheidung

Um den Betrachtungsraum und den Grad an formaler Genauigkeit der Architekturbeschreibung im Hinblick auf das Aufwand-Nutzen-Verhältnis festlegen zu können, erfolgt im ersten Schritt die Risikoeinstufung der Entscheidung. Entsprechend der Kriterien in Tab. 5.1 (siehe S. 63) ist zuerst zu überprüfen, unter welchen Bedingungen Typo3 als geschäftskritisches Softwaresystem aufzufassen ist. Typo3 ist die softwaretechnische Plattform für den Betrieb von ca. 4700 Webseiten [Rip+07]; die Mehrzahl dieser Webseiten ist dem nicht-kommerziellen Bereich zuzuordnen. Trotzdem wird Typo3 auch im professionellen und kommerziellen Bereich eingesetzt und ist beispielsweise die technologische Plattform, um den Kunden der Hotelkette Maritim einen weltweiten Webzugriff auf das Buchungs- und Reservierungssystem zu ermöglichen [Mrtm07]. Das Open-Source-Geschäftsmodell der Typo3-Entwicklung basiert darauf, durch Implementierung und Customizing von Typo3 im professionellen und kommerziellen Umfeld Gewinn zu erzielen. Die Fokussierung auf den kommerziellen Einsatz von Typo3 ist daher einer der Hauptgründe für die Restrukturierung des Datenbankzugriffs in Version 3.8.1. Durch die Möglichkeit, Typo3 auch mit anderen Datenbanksystemen als MySQL einsetzen zu können, soll die Verbreitung im Unternehmensumfeld verbessert werden. Da Typo3 bereits in professionellen und kommerziellen Bereichen genutzt wird und in Zukunft noch verstärkter eingesetzt werden soll, können die Typo3-Architekturveränderungen Auswirkungen auf geschäftskritische Bereiche von Unternehmen haben.

Für die Risikoeinstufung ist außerdem zu überprüfen, ob ein Rücksprung zu einer lauffähigen Version von Typo3 mit einem geringen Aufwand möglich ist. Ein solcher Rücksprung ist durch einen einfachen Austausch der angepassten und ergänzten Datenbank-Abstraktionsschicht 'T3lib_db' (siehe Abb. 6.2, S. 92) durch die unveränderte Version der 'T3lib_db' realisierbar. Eine Voraussetzung für die leichte Austauschbarkeit ist die Einhaltung der losen Kopplung zwischen der Datenbank-Abstraktionsschicht und den Typo3-Komponenten. Nur wenn die Art und Weise des Datenbankzugriffs der Typo3-Komponenten nicht verändert wurde, ist ein Rücksprung zu einer lauffähigen Version von Typo3 mit geringem Aufwand möglich. Da die Typo3-Architekturveränderung Auswirkungen auf geschäftskritische Bereiche von Unternehmen haben kann und ein Rücksprung zu einer lauffähigen Version mit geringem Aufwand möglich ist, wird die Architekturentscheidung der Risikoklasse 1 zugeordnet.

Beschreibung der relevanten Teile der Architektur

Zu Beginn der Architekturbeschreibung erfolgen zunächst die Analyse und Bewertung der zur Verfügung stehenden Informationen (siehe zum Vorgehen Abschnitt 5.2.2, S. 63f.). Hierfür steht einerseits der vollständige Quelltext von Typo3 in Form von PHP-Skripten zur Verfügung. Da Typo3 als Open-Source-Anwendung zur Verfügung steht, sind für den vollständigen Zugriff auf den Quelltext keine Lizenzkosten erforderlich. Zudem ist andererseits eine nahezu vollständige Dokumentation vorhanden, wie z. B. von Skrhj [Skrh06] oder Dambekalns [Damb06b]. Aufgrund des vollständigen Zugriffs auf den Quelltext und die gute Dokumentation sind keine speziellen Architekturanalysen oder Architektur-Rekonstruktionen notwendig.

Abhängig von der Risikoeinstufung werden im Anschluss die relevanten Teile der Architektur identifiziert. Gegenstand der Architekturveränderung ist gemäß des in Abb. 6.3 dargestellten Ziel-Wirkungsmodells (siehe S. 94) maßgeblich die Restrukturierung des Datenbankzugriffes zur Verbesserung der Portabilität. Für diese Architekturveränderung sind speziell jene Bibliotheken, Skripte und übergreifenden Funktionen von Typo3 relevant, die im direkten Bezug zum Datenbankzugriff stehen. Für die systematische Entscheidung über Lösungsansätze zur Architekturveränderung von Typo3 in der Ver-

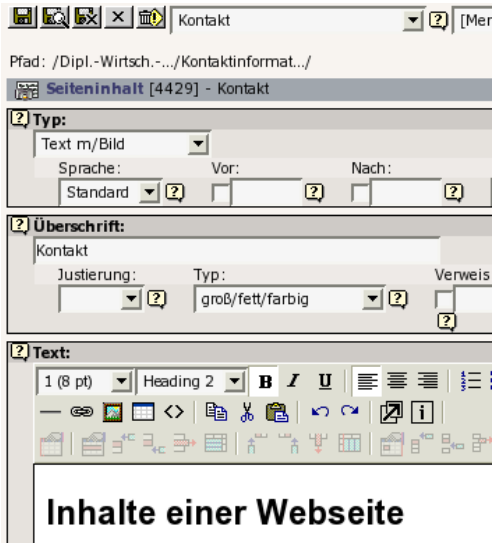
sion 3.8.1 sind daher die folgenden Teile der Architektur zu beschreiben, die in den nachstehenden Absätzen detailliert dargestellt werden[5]:

- *Typo3-Core (T3lib)*
- *Back- und Frontend*
- *Extensions und Extension-API*

Typo3-Core (T3lib): Der Kern von Typo3 beinhaltet 58 Bibliotheken, die als Klassen implementiert sind. Diese stellen insgesamt ein Framework für die Entwicklung von individuellen Webseiten dar. Die wichtigsten Bibliotheken der T3lib sind:

- *Datenbank-Zugriffsfunktionen:* Die Funktionen zum Datenbankzugriff sind in der Bibliothek 'T3lib_db' implementiert, u. a. die Funktion 'exec_INSERTquery' (siehe Listing 6.1 auf S. 93). Die für den Datenbankzugriff notwendigen SQL-Statements werden mittels der Funktionen in den Bibliotheken 'T3lib_sqlengine' und 'T3lib_sqlparser' generiert.
- *Datenmanipulation und -abfrage:* Die Bibliothek 'T3lib_tcemain' (die sog.' Core-Engine' oder der Typo3-Kern) enthält Funktionen zur Datenmanipulation und zur Datenabfrage aus der Datenbank, die auf den standardisierten Funktionen der 'T3lib_db'-Bibliothek aufbauen. Zur Laufzeit werden z. B. Webseiten-Inhalte durch den Benutzer im Backend eingegeben (siehe Abb. 6.4). Um die eingegebenen Daten in die Datenbank zu schreiben, wird vom Backend aus die Funktion 'process_datamap()' der 'T3lib_tcemain' genutzt. Die im Backend sichtbaren Eingabefelder für die Dateneingabe werden durch 'T3lib_tceforms' erzeugt. Weitere Backendfunktionen sind in 'T3lib_befunc' implementiert, z. B. Suchfunktionen.

Back- und Frontend: Das Backend ist die Benutzerschnittstelle zur Dateneingabe und zur Verwaltung der Webseiten. Ein Ausschnitt über die Backendfunktionen ist in Abb. 6.4 auf der rechten Seite dargestellt. Die Backendschnittstelle zur Datenbank ist 'tce_db', eine Laufzeitinstanz der 'T3lib_tcemain'. Um z. B. neue Inhalte einer Webseite in die Datenbank zu schreiben, gibt der Typo3-Benutzer die Daten in HTML-Formulare ein (siehe die Eingabefelder in Abb. 6.4 links). Drückt der Benutzer auf die Schaltfläche 'Speichern' (links oben) werden die eingegebenen Formulardaten mittels des Befehls HTTP-POST an 'tce_db' gesendet (rechts oben, hervorgehoben durch Unterstreichung). Durch 'tce_db' werden die Formulardaten ausgewertet und mittels 'process_datamap()' der 'T3lib_tcemain' in die Datenbank geschrieben (hervorgehoben durch Fettdruck). Dabei wird z. B. die Funktion 'exec_INSERTquery' aufgerufen (siehe Listing 6.1 auf S. 93).



Auszug aus 'index.php', Zeile 273-276

```
// Create new:
$content='<form action="'.
$this->doc->backPath.'tce_db.php"
method="post"> (...)'
```

Auszug aus 'tce_db.php', Zeile 175-197

```
// Executing the posted actions ...
   function main()    {
        (...)
        // Execute actions:
        $this->tce->process_datamap();
```

Auszug aus 'T3lib_tcemain', Zeile 2386

```
$GLOBALS['TYPO3_DB']->
     exec_INSERTquery($table,
                      $fieldArray);
```

Abb. 6.4: Ausschnitt aus dem Ablauf eines Datenbankzugriffs bei Typo3

5 Die Pfadangaben zu den in den folgenden Abschnitten aufgeführten PHP-Skripten und Komponenten sind im Anhang 10.4 (S. 177) aufgeführt.

Durch das Frontend von Typo3 werden die Webseiten auf Anfrage des Clients generiert; die Funktionalität ist implementiert im Script 'index_ts.php'. Analog zum Backend wird auf verschiedene Bibliotheken der 'Typo3-Core' zugegriffen. Die Generierung der Webseiten erfolgt anhand von 'TypoScript'-Anweisungen, der internen Skriptsprache von Typo3, mit der z. B. Navigationsmenüs von Webseiten gestaltet und konfiguriert werden können. Der zur Verarbeitung der 'TypoScript'-Anweisungen notwendige Interpreter ist in der 'T3lib_tsparser' implementiert. Die Funktionen zur Verarbeitung der Webseitentemplates, wie z. B. in welchem Frame die Daten aus der Datenbank enthalten sein sollen, sind in den Bibliotheken 'T3lib_tstemplate' und 'T3lib_tsparser_ext' implementiert. Eine auf die Anforderungen des Webseiten-Anbieters angepasste Konfiguration von 'TypoScript' ist in 'T3lib_tsstyleconfig' enthalten.

Extensions und Extension-API: In ihrer Gesamtheit stellen die 'Extensions' den dritten grundlegenden Teil der Typo3-Architektur dar. Durch die 'Extensions' wird der rudimentäre Funktionsumfang von Typo3 erweitert. Für Typo3 stehen aktuell ca. 3000 'Extensions' zur Verfügung [Rip+07]. Die 'Extensions' greifen über die Extension-API 'T3lib_extmgm' auf den Typo3-Core zu. Eine 'Extension' kann z. B. durch die API-Funktion 'addTCAcolumns' Felder zu einer Tabelle der Datenbank hinzufügen. Ein Beispiel für eine häufig verwendete 'Extension' ist der PhpMyAdmin [Labo06], eine Webanwendung zur komfortablen Verwaltung der Datenbestände in einer Datenbank, z. B. für das Anlegen von Sicherungskopien.

Der letzte Schritt vor der eigentlichen Architekturbeschreibung ist die Auswahl der geeigneten Sichten und der geeigneten Beschreibungssprache (siehe zum Vorgehen Abschnitt 5.2.2, ab S. 63f.). Für die vorliegende Typo3-Architekturveränderung ist eine funktionale Sicht auf den Datenbankzugriff zu beschreiben. Da die Bibliotheken als PHP-Klassen implementiert sind, ist die Beschreibung der Typo3-Architektur in Form eines UML-Klassendiagramms sinnvoll. Aufgrund der geringen Risikoeinstufung der Architekturentscheidung ist keine formale oder semiformale Architekturbeschreibung erforderlich. In Abb. 6.5 ist das Klassendiagramm dargestellt, welches die für die Restrukturierung des Datenbankzugriffs erforderlichen Komponenten umfasst. Im Klassendiagramm sind außerdem die in den vorangegangenen Absätzen erläuterten Abhängigkeitsbeziehungen zwischen den Typo3-Komponenten aufgeführt (siehe zu den Beziehungsarten S. 51ff.).

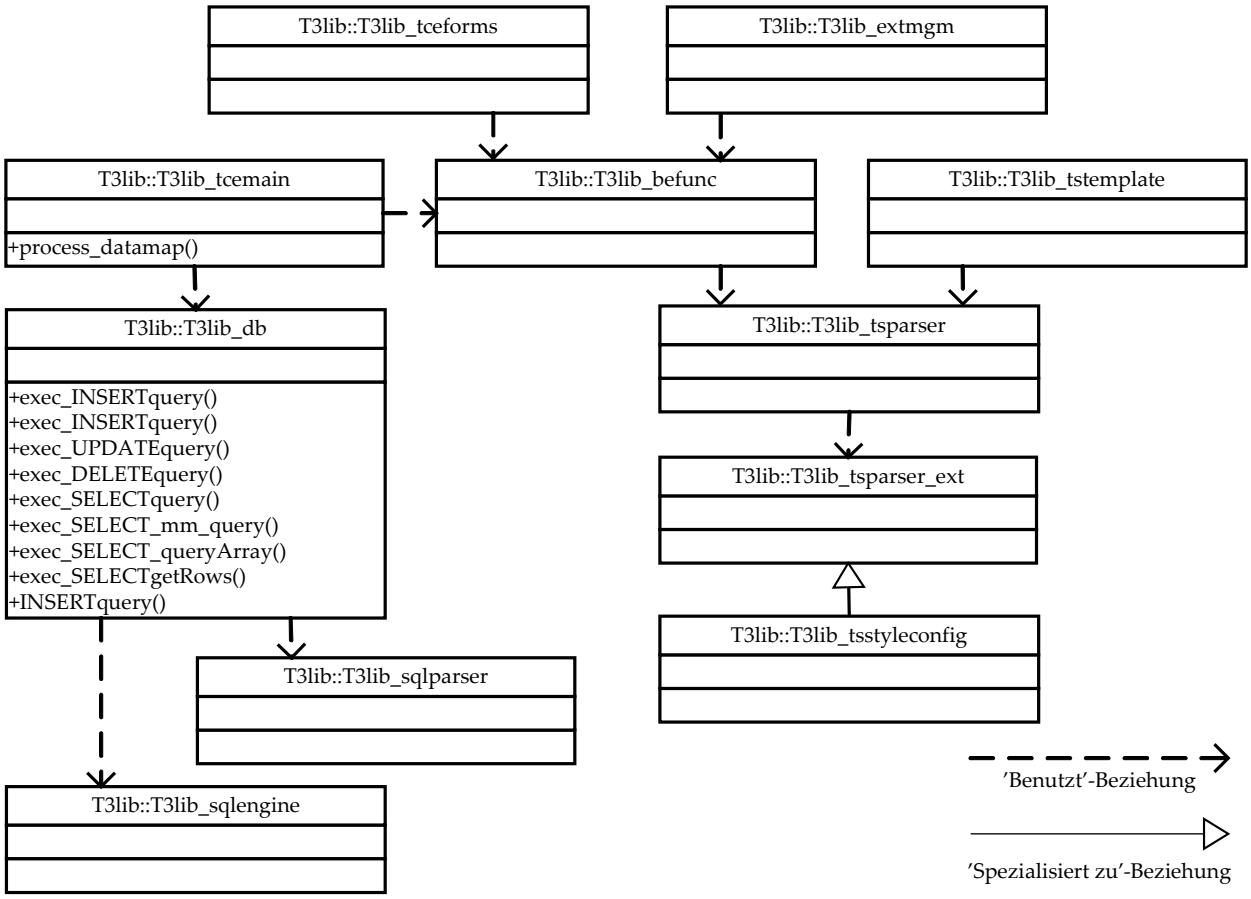


Abb. 6.5: Klassendiagramm mit den entscheidungsrelevanten Klassen und Methoden von Typo3

Ermittlung unbekannter Abhängigkeitsbeziehungen zwischen den Komponenten

Aufgrund der geringen Risikoeinstufung ist es nicht erforderlich, nach unbekannten Abhängigkeitsbeziehungen zu suchen. Diese Entscheidung unterliegt außerdem der Annahme, dass alle 'Extensions' über die 'Extension-API' und damit über die Datenbank-Abstraktionsschicht 'T3lib_db' auf die Datenbank zugreifen. Zwar kann technisch die 'Extension-API' umgangen werden und es ist ein direkter Datenbankzugriff an der Datenbank-Abstraktionsschicht 'T3lib_db' vorbei möglich. Die Typo3-Coding-Guidelines [Skrh06b] verbieten jedoch einen direkten Datenbankzugriff.

6.1.3 Phase 3 – Vorauswahl und Konkretisierung alternativer Lösungsansätze

Vorauswahl auf Kategorieebene

Nachdem die Ziele, Rahmenbedingungen und das Architekturmodell als Entscheidungsgrundlagen identifiziert, strukturiert und modelliert sind, kann eine systematische Auswahl geeigneter Lösungsansätze erfolgen (siehe zu den Phasen des Entscheidungsprozesses Abb. 3.1 auf S. 44). Da aus dem Ziel-Wirkungsmodell (siehe Abb. 6.3 auf S. 94) noch nicht klar erkennbar ist, mit welchen alternativen Lösungsansätzen die Datenbank-Abstraktionsschicht ergänzt und angepasst werden kann, um die Portabilität von Typo3 zu verbessern, erfolgt zunächst eine Vorauswahl auf Kategorieebene (siehe zu den optionalen und obligatorischen Tätigkeiten Tab. 10.2 auf S. 171ff.). Dabei werden im ersten Schritt zunächst diejenigen Kategorien von Mustern, Stilen, Technologien oder Softwareprodukten (siehe Abschnitt 7) ermittelt und ausgewählt, die zur Ergänzung und Erweiterung der Datenbankabstraktionsschicht geeignet sind. Das Ergebnis der Vorauswahl sind die folgenden Kategorien von Lösungsansätzen:

- *Bibliotheken:* Um die Abstraktionsschicht so zu erweitern, dass auf verschiedene Datenbanksysteme zugegriffen werden kann, müssen die notwendigen Zugriffsfunktionen, z. B. für Oracle- oder DB2-Datenbanken, in Typo3 implementiert werden. Hierfür kann das Konzept der Bibliotheken eingesetzt werden (siehe Abschnitt 7.4.3, S. 155f.). Bei dieser Kategorie von Lösungsansätzen sind die erforderlichen Datenbank-Zugriffsfunktionen in Bibliotheken implementiert, die von der Abstraktionsschicht referenziert und genutzt werden. Im Rahmen dieser Kategorie von Lösungsansätzen können entweder bereits existierende Bibliotheken eingesetzt werden, z. B. die ADOdb-Bibliothek [Lim04]. Alternativ können individuelle, datenbankspezifische Bibliotheken entwickelt werden, die die erforderlichen Zugriffsfunktionen (z. B. für Oracle oder DB2) beinhalten. Eine schematische Darstellung des Einsatzes von Bibliotheken ist in Abb. 6.6 wiedergegeben. Auf der rechten Seite ist dargestellt, wie die verschiedenen Typo3-Komponenten über die Funktionen, die in den Bibliotheken implementiert sind, auf die Datenbanken zugreifen.

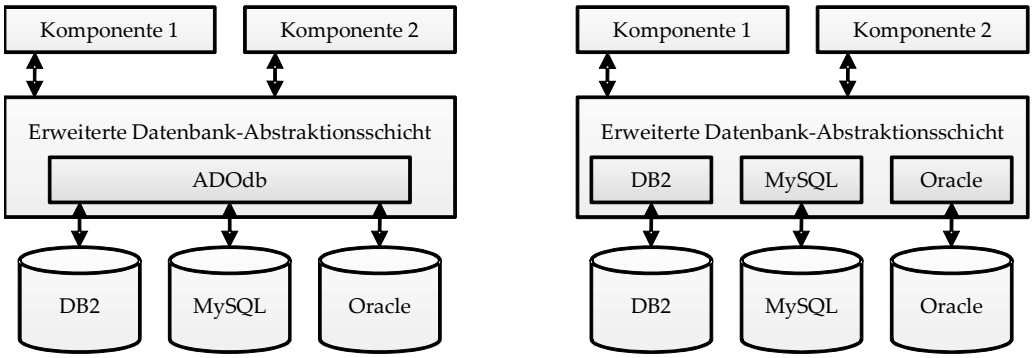


Abb. 6.6: Schematische Darstellung des Einsatzes von Bibliotheken

- *Programmiersprachen:* Eine weitere grundsätzliche Möglichkeit, um durch die Abstraktionsschicht auf verschiedene Datenbanksysteme zugreifen zu können, ist der Einsatz einer unabhängigen Datenbanksprache zur Anfrage und Manipulation der Daten. Ein Beispiel ist der Einsatz von ANSI-SQL (auch als SQL1 bezeichnet) als Alternative zur Generierung der datenbankspezifischer SQL-Statements. ANSI-SQL verfügt im Vergleich zur aktuellen SQL-Version (SQL-2 oder SQL-92) über einen reduzierten Funktionsumfang, der jedoch bei nahezu allen aktuellen Datenbanksystemen einheitlich implementiert ist. Da bestimmte Funktionen der aktuellen SQL-Version nicht klar definiert sind, z. B. Timestamps, sind die Funktionen in den jeweiligen Datenbanksystemen unterschiedlich implementiert. Daher können Anpassungen an den SQL-Statements erforderlich werden, wenn verschiedene Datenbanksysteme abgefragt werden müssen. So werden beispielsweise Timestamps bei Oracle im Format '12-AUG-2000 02:00:00' abgespeichert, bei MySQL weist derselbe Timestamp hingegen das Format '2000-08-12 02:00:00' auf.

Bei der Vorauswahl auf Kategorieebene sind im zweiten Schritt außerdem negative Auswirkungen auf die im Ziel-Wirkungsmodell enthaltenen Ziele zu überprüfen. Beim Einsatz von ANSI-SQL als alternative Datenbanksprache können Ver-

änderungen an den Zugriffszeiten oder Einschränkungen am funktionalen Verhalten auftreten. Um die Auswirkungen im Detail zu überprüfen, sind jedoch zusätzliche Analysen notwendig, die im Rahmen des anschließenden Rankings der Lösungsansätze durchgeführt werden.

Analyse und Ranking alternativer Lösungsansätze

Da bei der anschließenden Konkretisierung und Verfeinerung der Lösungsansätze aus Aufwandsgründen eine Beschränkung auf die geeignetsten Lösungsansätze erfolgen muss, werden die im vorangegangenen Abschnitt identifizierten Kategorien von Lösungsansätzen mithilfe einer szenariobasierten Architekturanalyse untersucht und im Anschluss in Form eines Rankings miteinander grob verglichen. Dadurch kann auf systematische und strukturierte Art und Weise die Teilmenge der Lösungsansätze ausgewählt werden, die sich am besten zur Erfüllung der Ziele eignet.

Schritte 1 und 2 – Auswahl der Szenarien und Erstellung der Architekturprototypen

Für die Analyse und das Ranking der alternativen Lösungsansätze erfolgt zunächst eine grobe szenariobasierte Architekturanalyse, wobei im ersten Schritt die Analyse-Szenarien zu ermitteln und auszuwählen sind. Für die grobe Analyse eignen sich insbesondere die folgenden beiden Szenarien:

- *Migration auf Oracle:* Für das Fundamentalziel Portabilität (siehe das Ziel-Wirkungsmodell in Abb. 6.3 auf S. 94) wird geprüft, welcher Aufwand mit der Migration von einer MySQL-Datenbank auf Oracle verbunden ist. Das erwünschte Ergebnis sind ein geringer Migrationsaufwand und eine problemlose Wiederverwendung der Datenbestände.
- *Migration auf SQL-2003:* Das zweite Szenario fokussiert auf die Verbesserung der Wartbarkeit von Typo3 durch die Überprüfung, welcher Aufwand mit der Migration auf SQL-2003 verbunden ist. SQL-2003 stellt eine Erweiterung des aktuellen SQL-Standards (SQL 2 oder SQL 92) dar. Es ist zu analysieren, welche Veränderungen am Prozess der Generierung der SQL-Statements notwendig sind, um die SQL-Statements SQL-2003-konform zu generieren. Das erwünschte Ergebnis ist ein im Vergleich zum unveränderten Datenbankzugriff reduzierter Wartungsaufwand.

  Durch das Szenario 'Migration auf SQL 2003' kann außerdem der Kopplungsgrad zwischen den Typo3-Komponenten und der Abstraktionsschicht ermittelt werden. Einerseits werden im Rahmen der Szenarioanalyse die Anpassungen am Prozess der Generierung der SQL-Statements durchgespielt. Andererseits wird damit implizit überprüft, ob Veränderungen an den 'Extensions' erforderlich sind. Sobald aufgrund der 'Migration auf SQL-2003' Veränderungen an den 'Extensions' erforderlich werden, ist die lose Kopplung zwischen den 'Extensions' und der Datenbank-Abstraktionsschicht 'T3lib_db' nicht mehr erfüllt.

Um die Szenarien anwenden zu können, wird im anschließenden zweiten Schritt eine prototypische Modellveränderung des existierenden Architekturmodells aus Abb. 6.5 (siehe S. 99) vorgenommen. Das existierende Modell wird dabei anhand der vorausgewählten Lösungsansätze in drei Richtungen weiterentwickelt. So wird einerseits die Implementierung der Bibliotheken mit Datenbank-Zugriffsfunktionen durchgespielt, wobei zwei alternative Lösungsansätze existieren. So kann entweder die bereits existierende ADOdb-Bibliothek implementiert werden oder es können neu entwickelte, datenbankspezifische Bibliotheken mit Zugriffsmethoden eingesetzt werden. Andererseits kann der Datenbankzugriff mit einer universellen Datenbanksprache erfolgen, wie z. B. ANSI-SQL (siehe für die Vorauswahl auf Kategorieebene S. 100). Somit werden drei Architekturprototypen erstellt, die als Grundlage für die im anschließenden dritten Schritt anstehende Verfeinerung und Konkretisierung der Lösungsansätze eingesetzt werden.

Schritt 3 – Modellveränderung und Analyse der drei Lösungsansätze

ADOdb-Alternative: Die ADOdb-Bibliothek stellt Transformations-Funktionen zur Verfügung, um ein generiertes SQL-Statement an verschiedene Datenbanksysteme anzupassen. Bei einem Zugriff der Typo3-Komponenten auf die Datenbank werden zunächst SQL-Statements generiert, wie u. a. SELECT- und UPDATE-Abfragen. Das generierte SQL-Statement wird im Anschluss mittels der Funktionen der ADOdb-Alternative im Hinblick auf die Spezifika der unterschiedlichen Daten-

bank-Systeme transformiert. Die Transformationen sind aufgrund von Lücken im SQL-Standard notwendig. So ist z. B. das Format von Timestamps im aktuellen SQL-Standard (SQL 2 bzw. SQL-92) nicht spezifiziert; dementsprechend sind Timestamps bei nahezu jedem Datenbanksystem unterschiedlich implementiert, z. B. bei Oracle im Format '12-AUG-2000 02:00:00', bei MySQL hingegen im Format '2000-08-12 02:00:00'. Zudem ist es erforderlich, die Rückgabewerte aus der Datenbankabfrage entsprechend umzuformen.

Für die Erstellung eines Architekturprototypen der ADOdb-Alternative wird speziell der Zugriff vom Backend auf die MySQL-Datenbank im Detail betrachtet. Der Backend-Datenbankzugriff ist vergleichbar mit anderen Zugriffen auf die Datenbank, insbesondere mit den Zugriffen von den 'Extensions' aus. Die Fokussierung auf aussagekräftige Architekturbereiche bei der Erstellung der Architekturprototypen ist im Hinblick auf das ökonomisch sinnvolle Aufwand-Nutzen-Verhältnis erforderlich. Die prototypische Modellveränderung ist in Abb. 6.7 dargestellt. Auf der linken Seite ist der Ausgangszustand, die unveränderte Datenbank-Abstraktionsschicht 'T3lib_db' abgebildet, die die Funktionen für den Datenbankzugriff zur Verfügung stellt (z. B. 'exec_INSERTquery'; siehe Listing 6.1 auf S. 93). Diese werden von der Backend-Benutzeroberfläche aus über die 'T3lib_tcemain' aufgerufen, woraufhin SQL-Statements generiert werden. Für eine detaillierte Beschreibung der Typo3-Architektur siehe Abschnitt 6.1.2 ab S. 97ff. Die skizzenhafte Darstellung in Abb. 6.7 ist aufgrund der Risikoeinstufung der Entscheidung (Klasse 1, siehe S. 96) ausreichend.

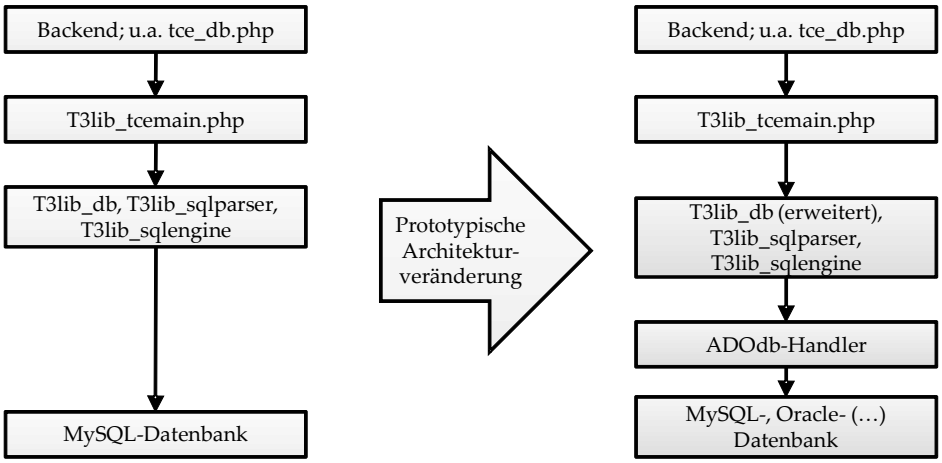


Abb. 6.7: Prototypische Modellveränderung der ADOdb-Alternative

Auf der rechten Seite der Abb. 6.7 ist der veränderte Datenbankzugriff dargestellt. Die Datenbank-Abstraktionsschicht 'T3lib_db' wird um eine Instanz der ADOdb-Bibliothek (einen sog. Handler) erweitert. Dieser Handler transformiert ein generiertes SQL-Statement hinsichtlich der Spezifika der unterschiedlichen Datenbanksysteme und leitet das transformierte SQL-Statement an die entsprechende Datenbank weiter. Zudem passt er die Rückgabewerte aus der Datenbankabfrage entsprechend an. Damit ist eine Anpassung der Komponenten für die Generierung der SQL-Statements 'T3lib_sqlparser' und 'T3lib_sqlengine' nicht erforderlich.

In Listing 6.2 ist anhand eines Quelltextbeispiels dargestellt, welche Veränderungen bei der Erweiterung der 'T3lib_db' vorzunehmen sind (hervorgehoben durch Unterstreichung, siehe für den Quelltext [Lim04]). Zuerst muss eine neue Verbindung zur Datenbank hergestellt werden; dies erfolgt durch die Übergabe des entsprechenden Parameters der Funktion 'NewADOConnection(<Datenbank>)'. Damit wird eine Verbindung zu einer – von ADOdb unterstützten – Datenbank aufgebaut. Die Datenbankverbindung (der Handler) '$DB' wird u. a. von der Funktion 'exec_INSERTquery()' (siehe Listing 6.1 auf S. 93) benutzt, um Daten in die Datenbank zu schreiben. Anstatt mittels der nativen MySQL-Funktion der Skriptsprache PHP 'mysql_query' die Datenbank abzufragen, erfolgt der Aufruf der ADOdb-Funktion 'Execute'. Die Generierung des SQL-Statements erfolgt durch Aufruf von 'INSERTquery' innerhalb der Komponenten 'T3lib_sqlengine' und 'T3lib_sqlparser'.

```
include('/path/to/adodb.inc.php');
$DB = NewADOConnection('mysql');
//Handler für Oracle: NewADOConnection('oracle');
//Handler für PostgeSQL (Version 7): NewADOConnection('postgres7');

$DB->Connect($server, $user, $pwd, $db);

function exec_INSERTquery($table,$fields_values)  {
  $res = $DB->Execute($this->INSERTquery($table,$fields_values);
    return $res;
 }
```

Listing 6.2: Erweiterungen an T3lib_db für den Datenbankzugriff mittels der ADOdb-Funktionen

Im Anschluss an die prototypische Modellveränderung erfolgt die Überprüfung der Zielerreichung und der Rahmenbedingungen anhand der beiden Szenarien. Bei der Anwendung des Szenarios 'Migration auf Oracle' (Szenario 1) auf den Architekturprototypen aus Abb. 6.7 (siehe S. 102) ist erkennbar, dass für die Migration lediglich eine Anpassung in der Komponente 'T3lib_db' notwendig ist. Für jedes neue Datenbanksystem ist eine entsprechende ADOdb-Instanz (ein Handler) zu implementieren. Aufgrund des geringen Migrationsaufwands kann Typo3 mittels der ADOdb-Alternative mit verschiedenen Datenbanksystemen portabel eingesetzt werden. Das ist jedoch nur bei den Datenbanksystemen möglich, die von ADOdb unterstützt werden. Offiziell zählen hierzu u. a. MySQL, Oracle, Firebird und PostgreSQL (siehe für eine detaillierte Aufstellung [ADOD06]). Wichtige Datenbanksysteme, wie u. a. der gerade bei Unternehmen häufig eingesetzte Microsoft-SQL-Server, werden jedoch nicht oder nur eingeschränkt unterstützt [Damb06b]. Bei der ADOdb-Alternative kann daher der Umfang der unterstützten Datenbanksysteme die Portabilität einschränken. Zudem ist unsicher, ob und wann die der ADOdb-Alternative zu Grunde liegenden Bibliotheken aktualisiert werden, wenn neue Versionen der bereits unterstützten Datenbanksysteme verfügbar sind.

Zur 'Migration auf SQL-2003' (Szenario 2) sind die 'T3lib_sqlparser' und 'T3lib_sqlengine' anzupassen, die für die Generierung des SQL-Statements verantwortlich sind. Beide Komponenten sind durch die Restrukturierung des Datenbankzugriffs nicht betroffen. Für die 'Migration auf SQL-2003' ist somit derselbe Implementierungsaufwand wie vor der Restrukturierung des Datenbankzugriffs erforderlich; der Kopplungsgrad mit den 'Extensions' ist ebenfalls identisch.

Im Ergebnis der groben Architekturanalyse kann ermittelt werden, dass mit der Implementierung der ADOdb-Alternative die Portabilität verbessert werden kann. Unter der Annahme, dass die lose Kopplung zwischen den Typo3-Komponenten und der Datenbank-Abstraktionsschicht 'T3lib_db' beibehalten wird, wird die Wartbarkeit zwar nicht verschlechtert, aber auch nicht verbessert. Für die Überprüfung der Zugriffszeiten sind detailliere Analysen erforderlich. Da die Bibliotheken für die ADOdb-Alternative bereits zur Verfügung stehen, sind außerdem keine Widersprüche mit den in Abschnitt 6.1.1 (siehe S. 94f.) dargestellten Rahmenbedingungen zu erkennen.

Modellveränderung und Analyse der Alternative datenbankspezifischer Bibliotheken: **I**m Rahmen der groben szenariobasierten Architekturanalyse des zweiten Lösungsansatzes werden datenbankspezifische Bibliotheken mit Zugriffsmethoden betrachtet. Diese Bibliotheken, die im Gegensatz zur ADOdb-Alternative zunächst zu entwickeln sind, werden von der Abstraktionsschicht 'T3lib_db' referenziert und können für Datenbankanfragen benutzt werden. Diese in der Skriptsprache PHP implementierten Bibliotheken beinhalten native PHP-Funktionen zum Zugriff auf verschiedene Datenbanksysteme. Neben nativen MySQL-Funktionen (z. B. 'mysql_query(SQL-Statement)', siehe Listing 6.1, S. 93) existieren in PHP u. a. native Oracle-Funktionen (z. B. 'ora_parse (SQL-Statement, Parameter)', [Arc+06]).

In Abb. 6.8 ist die prototypische Modellveränderung der Alternative Entwicklung datenbankspezifischer Bibliotheken zu sehen. Dabei wurde wiederum der Zugriff des Backends auf die MySQL-Datenbank als repräsentativer Ausschnitt des Architekturmodells ausgewählt (dargestellt auf der linken Seite). Von der Backend-Benutzeroberfläche aus werden die Datenbankzugriffs-Funktionen der 'T3lib_db' aufgerufen (z. B. 'exec_INSERTquery'; siehe Listing 6.1 auf S. 93) und daraufhin SQL-Statements für den Zugriff auf die MySQL-Datenbank generiert.

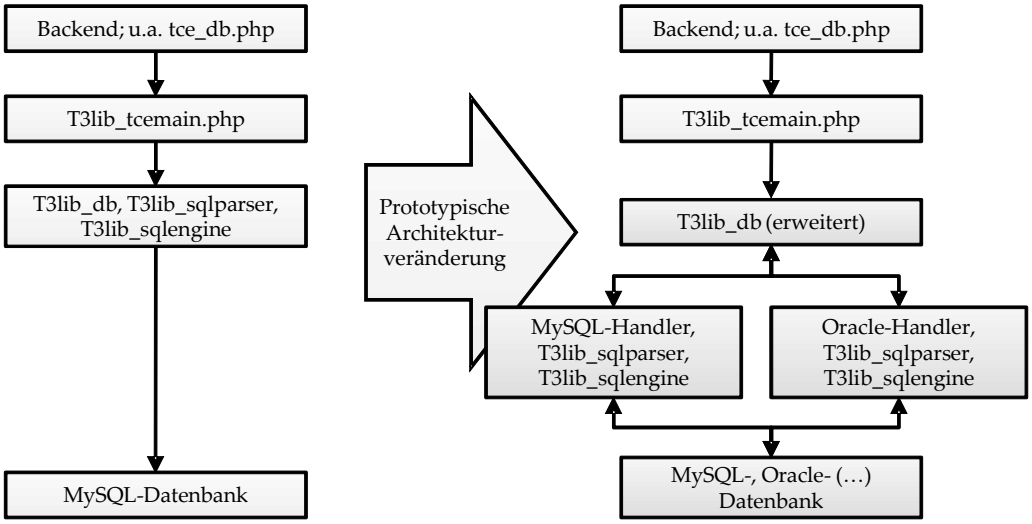


Abb. 6.8: Prototypische Modellveränderung der datenbankspezifischen Bibliotheken

Im prototypisch veränderten Modell (Abb. 6.8, rechte Seite) stellen die Bibliotheken bzw. deren Handler-Instanzen die Funktionalität für den Datenbankzugriff zur Verfügung. Dabei existiert je Datenbanksystem ein spezieller Handler, der angepasste Versionen der Komponenten 'T3lib_sqlparser' und 'T3lib_sqlengine' umfasst (in Abb. 6.8 dargestellt sind die Handler für MySQL und Oracle). Für jedes Datenbanksystem existiert somit ein angepasstes Vorgehen zur Generierung der SQL-Statements. Bei dieser Alternative ist zudem die Schnittstelle 'T3lib_db' zu erweitern, damit die Aufrufe aus dem Backend zum relevanten Handler weitergeleitet werden.

In Listing 6.3 ist anhand eines Quelltextbeispiels dargestellt, welche Veränderungen an der 'T3lib_db' vorzunehmen sind, um die Alternative zu implementieren (hervorgehoben durch Unterstreichung). Dargestellt ist die Entwicklung von zwei exemplarischen Handlern ('$MySQL_DB' und '$Oracle_DB') und verschiedenen Instanzen der 'exec_INSERTquery()'. Darüber hinaus werden spezielle Funktionen zur Generierung der SQL-Statements benötigt, z. B. die Funktion 'oracle_INSERTquery' für die Oracle-konforme Generierung des SQL-Statements. Diese Funktion verweist auf die datenbankspezifischen Bibliotheken, welche als angepasste Versionen der 'T3lib_sqlparser' und 'T3lib_sqlengine' implementiert werden.

```
$MySQL_DB = mysql_pconnect($Server, $ID, $Pass);

function exec_INSERTquery($table,$fields_values)  {
  $res = mysql_query($this->INSERTquery($table,$fields_values),
                     $this->MySQL_DB);
  return $res;
 }

$Oracle_DB = Ora_Logon("Benutzername@TNSNAME", "pass");

function exec_INSERTquery($table,$fields_values)  {
  $res = ora_parse($this->Oracle_DB,$this->
                   oracle_INSERTquery($table,$fields_values);

  return $res;
 }
```

Listing 6.3: MySQL- und Oracle-spezifischer Insertbefehl (T3lib_db)

Im Anschluss an die Modellveränderung wird durch die Anwendung der beiden Szenarien 'Migration auf Oracle' und 'Migration auf SQL-2003' überprüft, inwieweit die Portabilität und die Wartbarkeit verbessert und ob die Rahmenbedingungen eingehalten werden können. Zur Migration auf Oracle oder auf ein anderes Datenbanksystem (z. B. DB2) sind neue Datenbank-Handler zu entwickeln und zu implementieren (unter der Annahme, dass ein solcher Handler noch nicht implementiert ist). In Listing 6.3 ist ein Beispiel für einen 'Oracle-Handler' dargestellt. Da zur Migration auf ein anderes Datenbanksystem auch die Methoden für die Generierung des SQL-Statements zu entwickeln und zu implementieren sind, ist ein

höherer Migrationsaufwand als bei der ADOdb-Alternative notwendig. Der Aufwand im Falle des zweiten Szenarios 'Migration auf SQL-2003' ist ebenfalls höher als bei der ADOdb-Alternative, da mehrere Handler für die Generierung der SQL-Statements existieren. Zur Migration auf SQL-2003 müssten alle Handler angepasst werden.

Im Ergebnis der groben Architekturanalyse ist festzustellen, dass mit der Alternative der Entwicklung datenbankspezifischer Bibliotheken die Ziele in einem geringeren Umfang als mit der ADOdb-Alternative erfüllt werden können. Dennoch sind die Verbesserung der Portabilität und die Einschränkung bei der Wartbarkeit ausreichend bzw. akzeptabel. Kritisch ist bei dieser Alternative der Aufwand zur Entwicklung der datenbankspezifischen Bibliotheken, welcher maximal 70 Personentage betragen darf (siehe zu den Rahmenbedingungen Abschnitt 6.1.1, S. 94f.). Da im Gegensatz zur ADOdb-Alternative die Bibliotheken noch zu konzipieren, zu entwickeln und zu testen sind, ist ein Überschreiten der 70 Personentage wahrscheinlich. Im Rahmen der Verfeinerung und Konkretisierung der Alternativen ist dieser Punkt im Detail zu überprüfen.

Modellveränderung und Analyse der ANSI-SQL-Alternative: Bei der ANSI-SQL-Alternative wird im Vergleich zur aktuellen Version von SQL (SQL2 oder SQL-92) eine im Funktionsumfang reduzierte SQL-Syntax eingesetzt. Diese ist bei den aktuellen Datenbanksystemen einheitlich implementiert; ein in der ANSI-SQL-Syntax formuliertes SQL-Statement erfordert daher keine weiteren Transformationen, um damit verschiedene Datenbanksysteme abfragen zu können. Die ANSI-SQL-Alternative kann jedoch aufgrund der technischen Rahmenbedingungen nicht eingesetzt werden, da durch die reduzierte SQL-Syntax Veränderungen an der Datenstruktur notwendig werden. So sind beispielsweise die Datenbankstrukturen anzupassen, da die gerade bei relationalen Datenbanken wichtigen JOIN-Verknüpfungen erst mit aktuellem SQL (SQL 2 oder SQL-92) genutzt werden können. Damit ist jedoch die in den Rahmenbedingungen geforderte Abwärtskompatibilität zu früheren Typo3-Versionen nicht mehr gegeben (siehe Abschnitt 6.1.1, S. 94f.). Da aus diesen Gründen die ANSI-SQL-Alternative für die Typo3-Architekturveränderung nicht geeignet ist, ist auch keine prototypische Modellveränderung und Analyse notwendig.

Schritt 4 – Ranking und Vorauswahl

Im vierten Schritt erfolgt das Ranking und es wird eine Vorauswahl der geeigneten Lösungsansätze durchgeführt. Die ANSI-SQL-Alternative wird dabei aufgrund der fehlenden Abwärtskompatibilität nicht mehr betrachtet. Auf der Grundlage der groben szenariobasierten Architekturanalyse hat die ADOdb-Alternative das größere Potenzial zur Erfüllung der Ziele. Im Vergleich zur Entwicklung datenbankspezifischer Bibliotheken ist der Implementierungsaufwand auf den ersten Blick geringer. Zudem ist die ADOdb-Alternative bei der Entwicklung von geringerer Komplexität. Im Verlauf der Verfeinerung und Konkretisierung der ADOdb-Alternative werden jedoch massive Schwachpunkte im Hinblick auf die Zugriffszeiten ersichtlich.

Konkretisierung und Anpassung – ADOdb-Alternative

Da die Ergebnisse der im vorangegangenen Abschnitt durchgeführten Szenarioanalysen noch zu grob sind, um die Lösungsansätze direkt miteinander vergleichen zu können, werden die beiden Lösungsansätze verfeinert und konkretisiert. So kann der Implementierungsaufwand noch nicht auf Personentage genau bestimmt werden; zudem fehlt insbesondere die Überprüfung, in wie weit die Zugriffszeiten beeinflusst werden. Um die Risiken und Unsicherheiten zu reduzieren, sind detailliertere Analysen erforderlich. Es wird daher eine Verfeinerung und Konkretisierung der Lösungsansätze durchgeführt, die entsprechend der Ausführungen in Abschnitt 5.3.4 (siehe S. 73ff.) die folgenden fünf Schritte umfassen.

Schritt 1 – Aufstellung der Maßnahmenfolge

Der erste Schritt zur Verfeinerung und Konkretisierung der ADOdb-Alternative ist die Aufstellung und Verfeinerung der Maßnahmenfolge auf der Grundlage der prototypischen Modellveränderung, die in Abb. 6.7 (siehe S. 102) dargestellt ist. Demnach sind zur Implementierung die folgenden groben Aktivitäten notwendig:

- *Hinzufügung der Bibliothek:* Der erste Schritt ist die Hinzufügung der ADOdb-Bibliothek als 'Extension' (sog. Systemextension). Auf die ADOdb-Funktionalität kann somit über die 'Extension-API' des 'Typo3-Core' zugegriffen werden.
- *Veränderung der 'T3lib_db':* Um die ADOdb-Funktionalität nutzen zu können, sind Instanzen der ADOdb-Bibliothek zu implementieren (die sog. Handler). Die vorhandene Core-Bibliothek 'T3lib_db' wird erweitert, um auf die Instanzen der ADOdb-Bibliothek (die Handler) zugreifen zu können. Der Prozess für die Generierung der SQL-Statements in den Komponenten 'T3lib_sqlparser' und 'T3lib_sqlengine' wird hingegen nicht verändert. Zur Laufzeit wird das generierte Statement an einen ADOdb-Handler weitergeleitet, welcher es je nach ausgewählter Zieldatenbank transformiert, z. B. spezifisch für Oracle oder DB2. Der Handler transformiert zudem die Rückgabewerte aus der Datenbankabfrage.

Die konkretisierte Maßnahmenfolge der ADOdb-Alternative ist in Abb. 6.9 anhand eines Aktivitätsdiagramms abgebildet.

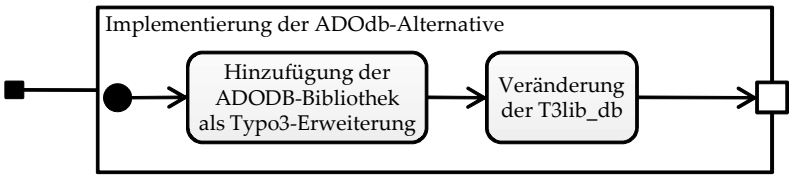


Abb. 6.9: Verfeinerung und Anpassung der Maßnahmenfolge der ADOdb-Alternative

In Abb. 6.10 ist das verfeinerte und angepasste prototypische Architekturmodell mithilfe eines vereinfachten Komponentendiagramms dargestellt (die Veränderungen sind durch Fettdruck hervorgehoben). Aus dem 'Typo3-Core', z. B. dem Backend oder der 'T3lib_tcemain', wird auf die 'Datenbank-Zugriffsfunktionen' der erweiterten 'T3lib_db' zugegriffen. Daraufhin wird ein SQL-Statement generiert und durch den ADOdb-Handler transformiert. Das Statement wird im Anschluss an die Datenbank weitergeleitet. Der Handler transformiert zudem die Rückgabewerte aus der Datenbankabfrage.

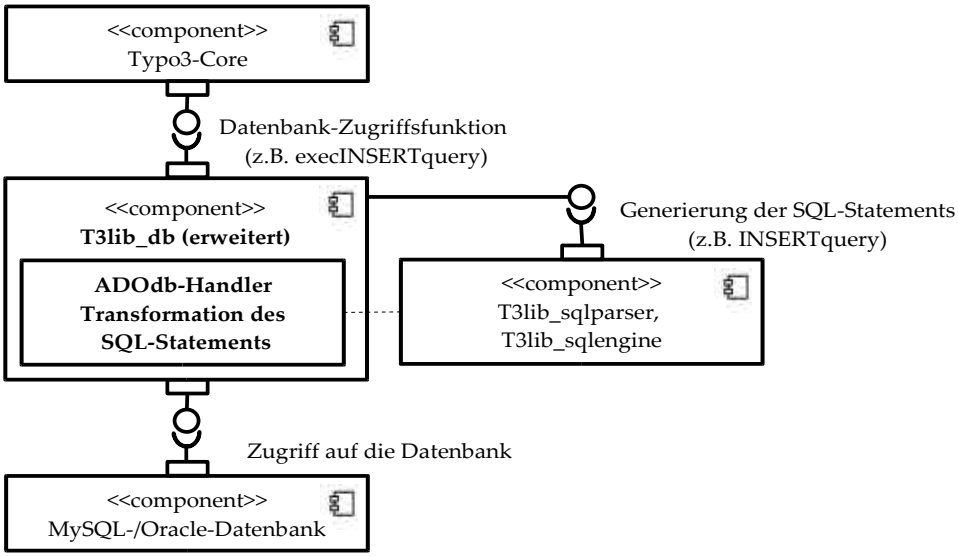


Abb. 6.10: Verfeinertes und angepasstes prototypisches Architekturmodell

Schritt 2 – Auswertung der Abhängigkeitsbeziehungen

Im Anschluss an die Verfeinerung der Maßnahmenfolge werden im zweiten Schritt die Abhängigkeitsbeziehungen ausgewertet, die zwischen den Typo3-Komponenten und der ADOdb-Alternative existieren sowie in Abb. 6.5 (siehe S. 99) dokumentiert sind. So 'benutzen' die Komponenten des 'Typo3-Core' die 'T3lib_db', welche die relevanten Funktionen für die Abfrage der Datenbank bereitstellt. Die 'T3lib_db' wiederum 'benutzt' die Funktionen der Bibliotheken 'T3lib_sqlparser' und 'T3lib_sqlengine', um SQL-Statements zu generieren. Diese Folge an 'benutzt'-Beziehungen muss bei der Reihenfolgeplanung der Maßnahmen berücksichtigt werden. So sind generell Veränderungen am Prozess bzw. am Algorithmus der Generierung der SQL-Statements zuerst durchzuführen, da die betreffenden Bibliotheken die Wurzel der 'benutzt'-Beziehungen darstellen. Erst im Anschluss können die geplanten Veränderungen und Erweiterungen an der 'T3lib_db' durchgeführt werden. Widersprüchliche Komponentenbeziehungen, beispielsweise zyklische Abhängigkeiten beim Datenbankzugriff der Typo3-Komponenten, sind aufgrund der guten losen Kopplung der Komponenten und der Datenbank-Abstraktionsschicht in der betrachteten Typo3-Version nicht bekannt.

Die Abhängigkeitsbeziehungen zwischen den Komponenten haben keine Auswirkungen auf die Reihenfolge der Maßnahmen zur Implementierung der ADOdb-Alternative, da bei diesem Lösungsansatz keine Veränderungen am Prozess bzw. am Algorithmus der Generierung der SQL-Statements vorgenommen werden. Lediglich die 'T3lib_db' wird um ADOdb-Handler ergänzt, die die Transformation des SQL-Statements für spezifische Datenbanksysteme übernehmen.

Schritt 3 – Ermittlung der Eigenschaften

Um einen systematischen Vergleich mit der Alternative datenbankspezifischer Zugriffsfunktionen zu ermöglichen, werden die Eigenschaften der ADOdb-Alternative ermittelt. Da eine erste Überprüfung der Verbesserung der Portabilität und der Wartbarkeit bereits im Rahmen des Rankings der Alternativen erfolgte (siehe S. 101ff.), sind speziell die Veränderungen an den Zugriffszeiten zu überprüfen. Die ADOdb-Bibliotheken sind in der Skriptsprache PHP implementiert. Da die Skripte zur Laufzeit durch einen Interpreter ausgeführt werden, können Verschlechterungen an den Zugriffszeiten auftreten. Dies wird an einem Walkthrough anhand des verfeinerten und angepassten prototypischen Architekturmodells Abb. 6.10 deutlich. Nachdem durch die Komponenten 'T3lib_sqlparser' und 'T3lib_sqlengine' das SQL-Statement generiert wurde, transformiert der ADOdb-Handler das SQL-Statement. Zudem transformiert der Handler die Rückgabewerte aus der Datenbankabfrage. Es ist zu befürchten, dass sich die Transformation der SQL-Statements negativ auf die Performance auswirkt. Da jedoch keine Veränderungen an den Zugriffszeiten auftreten dürfen (siehe Abb. 6.3 auf S. 94), können ergänzende Optimierungen notwendig werden, wie z. B. der Einsatz von Caches. Auf der Basis von Vergleichswerten wird abgeschätzt, dass dieser Fall und die damit verbundenen Optimierungen mit einer Wahrscheinlichkeit von 60 % eintreten werden.

In Abb. 6.11 ist die erweiterte Maßnahmenfolge abgebildet. Nach der 'Veränderung der T3lib_db' erfolgt die Analyse, ob die Zugriffszeiten negativ beeinflusst werden. Nur im Falle von Verschlechterungen an den Zugriffszeiten sind die erforderlichen Optimierungen durchzuführen.

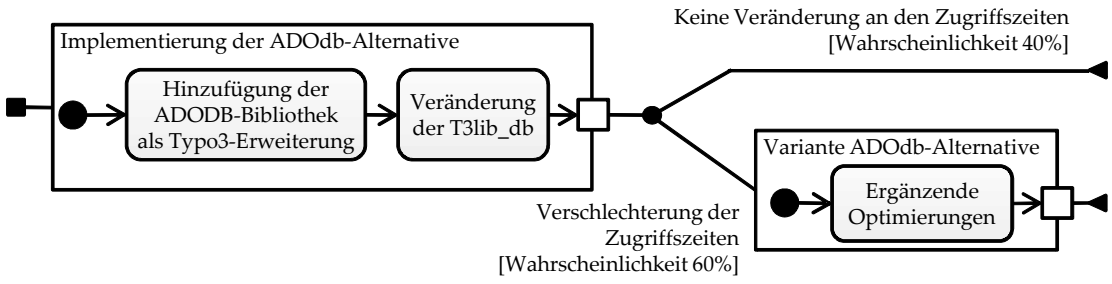


Abb. 6.11: Um Optimierungen erweiterte Maßnahmenfolge

Zur Quantifizierung der Zielerreichungsgrade der ADOdb-Alternative wird eine Notenskala, mit Noten im Intervall von eins bis vier, bevorzugt. Die Note eins repräsentiert die beste Zielerreichung, bei der Note vier hingegen werden die Ziele

nicht erreicht. Mit der ADOdb-Alternative können die Ziele wie folgt erreicht werden (siehe außerdem die zusammenfassende Darstellung in Tab. 6.1):

- *Portabilität:* Mit der ADOdb-Alternative wird in beiden Varianten (mit und ohne Optimierung) eine gute Portabilitätsverbesserung (Note 2) erzielt. Eine bessere Note kann nicht vergeben werden, da durch die ADOdb-Alternative zwar eine Vielzahl von Datenbanksystemen unterstützt wird (z. B. MySQL, Oracle, Firebird oder PostgreSQL [ADOD06]). Wichtige andere Datenbanksysteme, wie z. B. der Microsoft-SQL-Server, werden hingegen nicht oder nur eingeschränkt unterstützt [Damb06b].
- *Wartbarkeit:* Die Auswirkungen auf die Wartbarkeit werden ohne die ergänzenden Optimierungen aufgrund der guten losen Kopplung als sehr gut (Note 1) bewertet. Mit einer Wahrscheinlichkeit von 60 % werden nachträgliche Optimierungen erforderlich sein, welche die Verständlichkeit mindern können. In der Implementierungsvariante mit Optimierungen wird daher nur eine befriedigende Wartbarkeit (Note 3) erwartet.
- *Zugriffszeiten:* In 40% aller Fälle sind die Zugriffszeiten unverändert (Note 2) und keine weiteren Optimierungen notwendig. Sofern Optimierungen am Datenbankzugriff durchgeführt werden, können deutlich verbesserte Zugriffszeiten auf die Datenbank erwartet werden (Note 1).

Varianten der ADOdb-Alternative	Portabilität	Wartbarkeit	Zugriffszeiten
ADOdb-Bibliothek	2	1	2
ADOdb-Bibliothek mit ergänzender Optimierung	2	3	1

Tab. 6.1: Zielerreichung der ADOdb-Alternative (Noten, im Intervall von 1 bis 4)

Neben dem Grad der Zielerreichung ist der Implementierungsaufwand zu ermitteln, um die Alternativen ausgewogen und systematisch vergleichen zu können. In Abb. 6.12 ist der von den beteiligten Entwicklern geschätzte Implementierungsaufwand der ADOdb-Alternative dargestellt. Karsten Dambekalns schätzte den Aufwand zur Implementierung der ADOdb-Alternative mit 50 Personentagen (PT), wobei Optimierungen einen ähnlich großen Aufwand benötigen werden. Weitere Quellen für die Ermittlung der Aufwandszahlen sind Vergleichsdaten von ähnlichen Entwicklungsprojekten in Zusammenhang mit dem Einsatz der ADOdb-Bibliothek. Hierbei wird zuerst der Einzelaufwand für jede Aktivität in Form von Personentagen ermittelt. Im Anschluss ist der Gesamtaufwand jeder der beiden Implementierungsvarianten zu ermitteln. In der Variante ohne Optimierungen ist ein Aufwand in Höhe von 50 Personentagen (PT) erforderlich. Mit einer Wahrscheinlichkeit von 60 % sind jedoch Optimierungen erforderlich, die zusätzlich 25 Personentage erfordern.

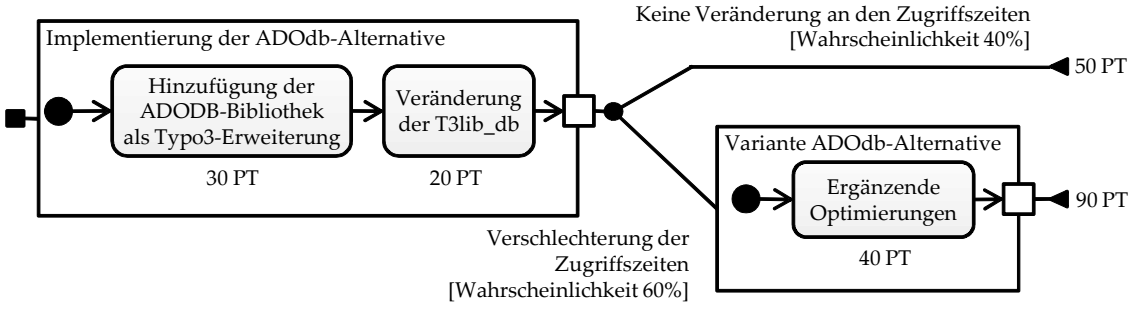


Abb. 6.12: Quantifizierung des Implementierungsaufwands (Personentage - PT)

Damit liegt der Implementierungsaufwand im Falle von zusätzlichen Optimierungen über der in den Rahmenbedingungen festgelegten Höchstgrenze von 70 Personentagen (siehe Abschnitt 6.1.1, S. 94f.). Die weiteren organisatorischen und technischen Rahmenbedingungen sind durch die ADOdb-Alternative jedoch erfüllt.

Schritt 4 – Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften

Im letzten Schritt erfolgt die Überprüfung und die Reaktion auf Veränderungen funktionaler Systemeigenschaften, da das Typo3-System auch nach realisierter Architekturveränderung identische Systemeigenschaften aufweisen muss (siehe die Ausführungen in Abschnitt 6.1.1 auf S. 94). Das ist vor allem im Hinblick auf die Einhaltung der Abwärtskompatibilität zu beachten. Da durch die ADOdb-Alternative der Datenbankzugriff restrukturiert wird, ist zu überprüfen, ob funktionale Veränderungen beim Datenbank-Zugriff drohen. Somit sind speziell die Funktionen für den Datenbankzugriff der Komponente 'T3lib_db' zu überprüfen, ob sie durch die Implementierung der ADOdb-Alternative verändert oder beeinträchtigt werden. Es wird erwartet, dass die Funktionen in identischer Art und Weise aufgerufen sowie die SQL-Statements generiert und korrekte Rückgabewerte geliefert werden können.

Die Datenbank-Abstraktionsschicht 'T3lib_db' stellt die folgenden Funktionen für den Zugriff auf die Datenbank bereit:

- exec_INSERTquery (Zeile 168ff.)
- exec_UPDATEquery (Zeile 184ff.)
- exec_DELETEquery (Zeile 198ff.)
- exec_SELECTquery (Zeile 217ff.)
- exec_SELECT_mm_query (Zeile 242ff.)
- exec_SELECT_queryArray (Zeile 265ff.)
- exec_SELECTgetRows (Zeile 288ff.)

Im Rahmen der Überprüfung werden ein Aufruf der Datenbank-Zugriffsfunktionen, die Generierung der SQL-Statements und die anschließende Transformation mithilfe von Walkthroughs durchgespielt. Die Walkthroughs zeigen, dass bei der Implementierung der ADOdb-Alternative das Verfahren zur Generierung der SQL-Statements nicht verändert wird, da die ADOdb-Handler ein bereits generiertes SQL-Statement erhalten. Veränderte funktionale Eigenschaften können hingegen bei der Transformation der SQL-Statements auftreten. Da die ADOdb-Bibliothek jedoch bei vielzähligen Projekten eingesetzt wird (siehe [Lim04] und [ADOD06]) und die Datenbank-Zugriffsfunktionen lediglich die grundlegende Abfrage und Manipulationsfunktionen (INSERT, SELECT, UPDATE, DELETE) von SQL nutzen, können identische funktionale Systemeigenschaften erwartet werden. Ergänzungen oder Korrekturen an der Maßnahmenfolge sind somit nicht erforderlich.

Konkretisierung und Verfeinerung – Alternative Entwicklung datenbankspezifischer Bibliotheken

Schritt 1 – Aufstellung der Maßnahmenfolge

Der erste Schritt zur Verfeinerung und Konkretisierung der zweiten betrachteten Alternative, der Entwicklung von Bibliotheken mit datenbankspezifischen Zugriffsfunktionen, ist die Verfeinerung und Anpassung der Maßnahmenfolge. Das ist die Grundlage, um im Detail ermitteln zu können, in welchem Umfang die Ziele erfüllt werden und welcher Implementierungsaufwand erforderlich ist. Anhand der in Abschnitt 6.1.3 (siehe S. 101ff.) durchgeführten prototypischen Modellveränderungen können die folgenden Maßnahmen abgeleitet werden, die zur Implementierung der Alternative erforderlich sind:

- *Hinzufügung der Bibliothek für den Zugriff auf Oracle-Datenbanken:* Im Rahmen der Implementierung der Alternative erfolgt zunächst die Entwicklung der Bibliotheken mit den Funktionen für den Datenbankzugriff. Hierbei wird einerseits eine Bibliothek mit Zugriffsmethoden für Oracle-Datenbanken entwickelt und implementiert, da Oracle in Unternehmen vergleichsweise häufig im Einsatz ist. Im Detail werden dabei die existierenden Komponenten 'T3lib_sqlparser' und 'T3lib_sqlengine', welche Zugriffsfunktionen für MySQL-Datenbanken beinhalten, um Oracle-spezifische Zugriffsfunktionen ergänzt und verändert. Analog zur ADOdb-Alternative wird die Bibliothek als 'Extension' implementiert; somit kann der 'Typo3-Core' über die 'Extension-API' auf die Bibliothek zugreifen.
- *Hinzufügung der Bibliothek für den Zugriff auf Microsoft-SQL-Server-Datenbanken:* Neben einer Bibliothek mit Zugriffsfunktionen für Oracle wird eine Bibliothek für den Zugriff auf Microsoft-SQL-Server entwickelt und implementiert.

Das Verfahren zur Entwicklung und Implementierung ist identisch; die Bibliothek wird ebenfalls als 'Extension' implementiert.

- *Veränderung der 'T3lib_db':* Im Anschluss ist die 'T3lib_db' so zu verändern, dass Instanzen (sog. Handler) der Bibliotheken von den Typo3-Komponenten genutzt werden können. Durch die Handler erfolgt die Generierung der SQL-Statements für Oracle und Microsoft-SQL-Server und die Abfrage oder Manipulation der Daten in der Datenbank. Die existierenden Komponenten 'T3lib_sqlparser' und 'T3lib_sqlengine' werden nicht verändert; diese ermöglichen weiterhin den Zugriff auf MySQL-Datenbanken. Die 'T3lib' ist außerdem dahingehend anzupassen, dass die Datenbankanfragen (z. B. über exec_INSERTquery, siehe Listing 6.1 auf S. 93) an die richtigen Handler weitergeleitet werden.

Die Maßnahmenfolge zur Implementierung der Alternative ist in Abb. 6.13 dargestellt.

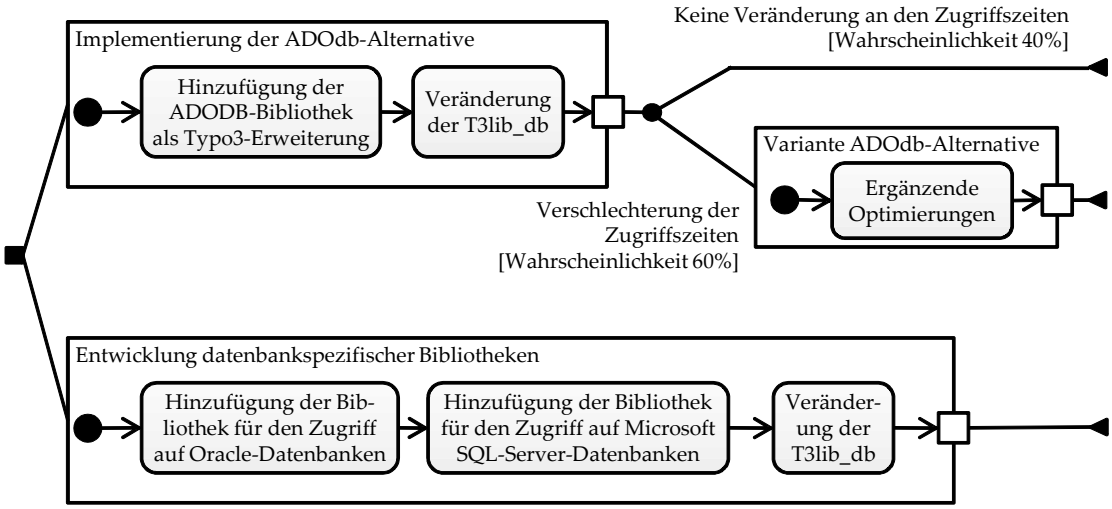


Abb. 6.13: Konkretisierung und Anpassung der Alternative Entwicklung datenbankspezifischer Bibliotheken

In Abb. 6.14 ist das verfeinerte und angepasste Architekturmodell der Alternative anhand eines vereinfachten Komponentendiagramms zu sehen (die Veränderungen sind durch Fettdruck hervorgehoben). Die Komponenten aus dem 'Typo-Core', z. B. das Backend oder die 'T3lib_tcemain', greifen auf die 'Datenbank-Zugriffsfunktionen' der erweiterten 'T3lib_db' zu. Die Anfrage wird an die jeweiligen Handler weitergeleitet, die unterschiedliche SQL-Statements generieren und diese an die Datenbank weiterleiten.

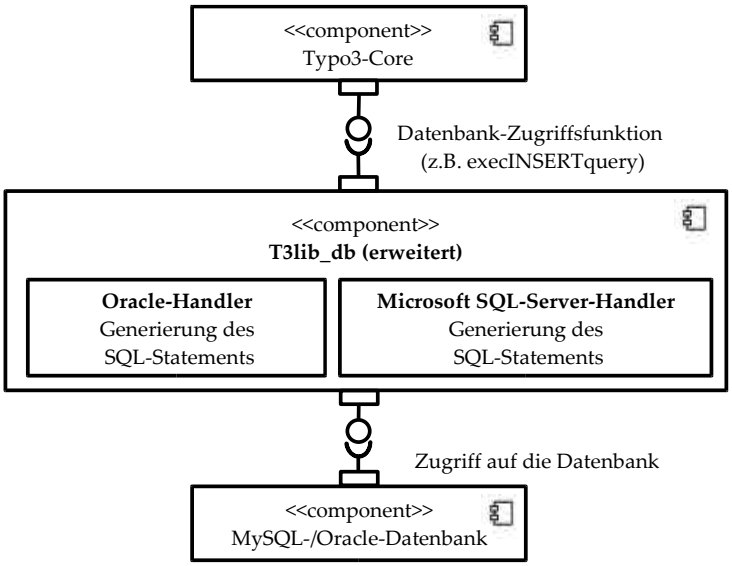


Abb. 6.14: Angepasstes prototypisches Architekturmodell

Schritt 2 – Auswertung der Abhängigkeitsbeziehungen

Im Anschluss an die Verfeinerung der Maßnahmenfolge sind die Abhängigkeitsbeziehungen zwischen den Typo3-Komponenten und den Bibliotheken mit datenbankspezifischen Zugriffsfunktionen zu betrachten und auszuwerten. In Abb. 6.5 (siehe S. 99) ist dokumentiert, wie der Typo3-Core die Datenbank-Abstraktionsschicht 'T3lib_db' benutzt, um Daten aus der Datenbank abzufragen. Die 'T3lib_db' wiederum benutzt die Funktionen der Bibliotheken 'T3lib_sqlparser' und 'T3lib_sqlengine', um SQL-Statements zu generieren. Da bei dieser Alternative der Prozess bzw. der Algorithmus zur Generierung der SQL-Statements verändert wird (siehe hierzu Abb. 6.8 und die Ausführungen auf S. 104), ist eine spezielle Reihenfolge bei der Implementierung einzuhalten. Die Generierung der SQL-Statements stellt dabei die Wurzel der Folge an 'benutzt'-Beziehungen dar und ist bei diesem Lösungsansatz zuerst anzupassen. Erst nachdem die relevanten Funktionen zur Generierung spezifischer SQL-Statements (für Oracle oder Microsoft-SQL-Server) implementiert und getestet sind, kann die 'T3lib_db' um die entsprechenden Handler erweitert werden. Da diese Reihenfolge bereits bei der in Abb. 6.13 (siehe S. 110) dargestellten Maßnahmenfolge berücksichtigt ist, sind keine Anpassungen erforderlich.

Schritt 3 – Ermittlung der Eigenschaften

Um einen systematischen Vergleich der beiden Lösungsansätze durchführen zu können, sind die Eigenschaften der aktuell betrachteten Alternative Entwicklung datenbankspezifischer Bibliotheken zu ermitteln. Da ein erster Überblick über den Umfang der Verbesserung der Portabilität und Wartbarkeit bereits im Rahmen des Rankings (siehe S. 101ff.) erarbeitet wurde, ist lediglich eine Überprüfung erforderlich, ob eine Verschlechterung der Zugriffszeiten auf die Datenbank zu befürchten ist. Ein Walkthrough des verfeinerten prototypischen Architekturmodells in Abb. 6.14 zeigt, dass im Gegensatz zur ADOdb-Alternative keine aufwendigen Transformationen der SQL-Statements und der Rückgabewerte aus der Datenbankabfrage erforderlich sind. Auf der Grundlage der Walkthroughs kann daher ermittelt werden, dass keine wesentlichen Veränderungen bei den Zugriffszeiten auf die Datenbank zu erwarten sind.

Bei der Alternative datenbankspezifischer Bibliotheken werden die Fundamentalziele Portabilität, Wartbarkeit und Zugriffszeiten im folgenden Umfang erreicht (siehe das Ziel-Wirkungsmodell in Abb. 6.3 auf S. 94). Analog zur ADOdb-Alternative wird der Zielerreichungsgrad in Form von Noten angegeben:

- *Portabilität:* Es wird eine sehr gute Portabilität erzielt (Note 1), da prinzipiell jedes Datenbanksystem unterstützt werden kann. Da jedoch im Gegensatz zur ADOdb-Alternative die relevanten Bibliotheken mit den datenbankspezifischen Zugriffsfunktionen zunächst zu entwickeln sind, ist mit dieser Alternative ein zusätzlicher Entwicklungsaufwand verbunden.
- *Wartbarkeit:* Die Auswirkungen auf die Wartbarkeit werden mit gut bewertet (Note 2). Die Erfahrungen in der Praxis zeigen, dass die individuelle Entwicklung einer Bibliothek meist fehleranfälliger und schlechter zu warten ist, als die Nutzung einer bereits vorliegenden und vielfältig eingesetzten Standardsoftware wie der ADOdb-Bibliothek.
- *Zugriffszeiten:* Da im Gegensatz zur ADOdb-Alternative keine Transformationen der SQL-Statements notwendig sind, werden keine Veränderungen an der Zugriffsgeschwindigkeit (Note 1) erwartet.

In Tab. 6.2 sind die Qualitätseigenschaften der beiden Alternativen zusammenfassend aufgeführt.

Betrachtete Lösungsansätze	Portabilität	Wartbarkeit	Zugriffszeiten
ADOdb-Bibliothek	2	1	2
ADOdb-Bibliothek mit ergänzender Optimierung	2	3	1
Entwicklung datenbank-spezifischer Bibliotheken	1	2	1

Tab. 6.2: Zielerreichung der betrachteten Alternativen (Noten im Intervall von 1 bis 4)

In Abb. 6.15 ist der Implementierungsaufwand der Alternative Entwicklung datenbankspezifischer Bibliotheken dargestellt. Zur Ermittlung der Aufwandszahlen wurden die beteiligten Entwickler befragt. Analog zur ADOdb-Alternative setzt sich der Implementierungsaufwand aus den Einzelaufwendungen der drei Maßnahmen zusammen; die Ermittlung der Anzahl der erforderlichen Personentage erfolgt dabei erneut auf Basis von Vergleichswerten. Da keine Varianten zu berücksichtigen sind, ist bei der Implementierung dieser Alternative mit einem Aufwand in Höhe von 70 Personentagen zu rechnen.

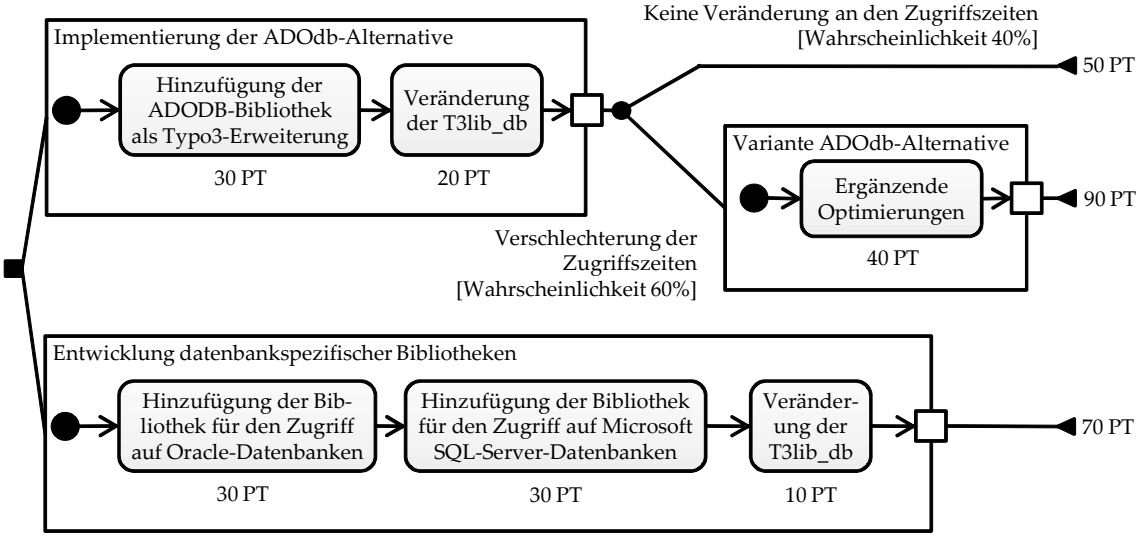


Abb. 6.15: Quantifizierung des Implementierungsaufwands

Somit stimmt der Implementierungsaufwand mit der Vorgabe der organisatorischen Rahmenbedingungen in Höhe von 70 Personentagen überein (siehe Abschnitt 6.1.1, S. 94f.). Aufgrund der Erfahrungen in der Praxis ist eine Überschreitung des Grenzwertes wahrscheinlich. Die weiteren organisatorischen und technischen Rahmenbedingungen können erfüllt werden.

Schritt 4 – Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften

Der vierte und letzte Schritt zur Verfeinerung und Konkretisierung der Alternative ist die Überprüfung auf Veränderungen am funktionalen Verhalten, da Typo3 aus Gründen der Abwärtskompatibilität auch nach den durchgeführten Restrukturierungen identische Systemeigenschaften aufweisen muss. Analog zur ADOdb-Alternative wird überprüft, ob die Datenbank-Zugriffsfunktionen der 'T3lib_db' (siehe die Liste auf S. 109) aufgerufen, die SQL-Statements generiert und die korrekten Rückgabewerte geliefert werden können.

Für die Überprüfung werden daher erneut ein Aufruf der Datenbank-Zugriffsfunktionen der 'T3lib_db', die Generierung der SQL-Statements und die anschließende Transformation durchgespielt. Durch die Implementierung der Alternative erfolgt die Generierung der SQL-Statements mithilfe der datenbankspezifischen Handler. Dabei können Veränderungen an den funktionalen Systemeigenschaften in solchen Fällen auftreten, in denen bisher genutzte MySQL-Funktionen, wie z. B. Such- oder Caching-Funktionen, bei der Verwendung der Datenbanken Oracle oder Microsoft-SQL-Server nicht mehr oder in anderer Konfiguration genutzt werden können. Das ist jedoch auszuschließen, da durch die Datenbank-Zugriffsfunktionen 'T3lib_db' lediglich einfache Abfrage- und Manipulationsfunktionen zur Verfügung gestellt werden (INSERT, SELECT, UPDATE, DELETE). Es ist allerdings wahrscheinlich, dass die Rückgabewerte (die Abfrageergebnisse) eine unterschiedliche Struktur oder ein abweichendes Format aufweisen. Das ist darauf zurückzuführen, dass die Datenbanksysteme unterschiedliche Tabellenformate verwenden. So unterstützt Oracle das Standard-MySQL-Tabellenformat MyISAM oder das ältere ISAM-Format nur unvollständig. Da diese geringfügigen Probleme jedoch im Rahmen der Implementierung der Alternative behoben werden können, sind keine Korrekturen an der Maßnahmenfolge erforderlich.

6.1.4 Phase 4 – Bewertung und Entscheidung

Im Anschluss an die Konkretisierung und Verfeinerungen erfolgt in der vierten Phase eine Bewertung der beiden Alternativen (siehe zu den Phasen des Entscheidungsprozesses Abb. 3.1 auf S. 44). Der Bewertungsschritt ist erforderlich, da aufgrund der Vielzahl an Eigenschaften nicht klar erkennbar ist, welche Alternative zu bevorzugen ist. So werden neben dem unterschiedlichen Implementierungsaufwand die Fundamentalziele Portabilität, Wartbarkeit und Zugriffszeiten in unterschiedlichem Umfang erfüllt. Darüber hinaus sind bei der ADOdb-Alternative aufgrund der drohenden Performance-Probleme unterschiedliche Implementierungsvarianten zu betrachten. Um eine Entscheidung nach rationalen Gesichtspunkten treffen zu können, ist eine systematische Auswertung der Eigenschaftswerte erforderlich; zum Einsatz kommt dabei das Bewertungsverfahren aus der Entscheidungstheorie. Zwar ist damit ein zusätzlicher Aufwand verbunden; dieser ist jedoch dadurch gerechtfertigt, da eine Fehlentscheidung und zusätzlicher Korrektur- oder Restrukturierungsaufwand vermieden werden kann.

Normalisierung der Eigenschaftswerte

Im ersten Schritt zur Bewertung der Alternativen bei der Typo3-Architekturveränderung erfolgt die Normalisierung der verschiedenen und unterschiedlich skalierten Eigenschaftswerte. Für die Bewertung ist der Grad der Zielerreichung bei der Portabilität, der Wartbarkeit, den Zugriffszeiten sowie der Implementierungsaufwand auf eine einheitliche Skala zu normalisieren. Gemäß den Ausführungen im Abschnitt 6.1.3 (siehe S. 107f.) steht der Grad der Zielerreichung in Noten, der Implementierungsaufwand in Personentagen (PT) zur Verfügung. Es erfolgt eine lineare Normalisierung der Noten und Personentage; speziell die vier Notenstufen werden in 0.25er Schritten normalisiert:

- *Portabilität:* Die bestmögliche Portabilitätsverbesserung ist die Note 1, die zum Wert 1.0 normalisiert wird; die schlechteste mit der Note 2 wird zum Wert 0.75 umgeformt.
- *Wartbarkeit:* Bei der Wartbarkeit sind drei Notenstufen zu normalisieren. Die positiven und negativen Extremwerte werden jeweils zu 1.0 und 0.5 normalisiert. Die Note 2 wird mit dem Wert 0.75 normalisiert.
- *Zugriffszeiten:* Analog zur Portabilität werden die beiden Extremwerte zu 0.75 und 1.0 umgeformt.
- *Implementierungsaufwand:* Der geringste Aufwand mit nur 50 PT wird vom Entscheidungsträger am stärksten bevorzugt (Wert 1.0). Der mit 90 PT höchste Aufwand liegt 20 PT über dem Grenzwert aus den Rahmenbedingungen und wird mit dem Wert 0.0 normalisiert. Der Aufwand in Höhe von 70 PT entspricht dem Mittelwert (aus 50 und 90 PT) und wird daher mit 0.5 normalisiert.

In Tab. 6.3 sind die normalisierten Eigenschaftswerte der Lösungsansätze zur Typo3-Architekturveränderung zusammengefasst dargestellt. In den linken Spalten sind die Eigenschaftswerte, in den rechten Spalten die normalisierten Werte aufgeführt (hervorgehoben durch graue Einfärbung).

Alternativen	Portabilität		Wartbarkeit		Zugriffszeiten		Aufwand	
ADOdb-Bibliothek	2	0.75	1	1.00	2	0.75	50 PT	1.00
ADOdb-Bibliothek mit ergänzender Optimierung	2	0.75	3	0.50	1	1.00	90 PT	0.00
Entwicklung datenbank-spezifischer Bibliotheken	1	1.00	2	0.75	1	1.00	70 PT	0.50

Tab. 6.3: Normalisierung der Eigenschaftswerte (Noten und Personentage – PT)

Gewichtung nach Relevanz

Im Anschluss sind die Prioritäten der Eigenschaftswerte zu berücksichtigen, da schon bei der Aufstellung des Ziel-Wirkungsmodells (siehe Abb. 6.3 auf S. 94) erkannt wurde, dass die Fundamentalziele eine unterschiedliche Priorität für die Entscheidungsfindung haben. Im Rahmen der Typo3-Architekturveränderung ist die Portabilitätsverbesserung das wichtigste Ziel. Die Zugriffszeiten haben hingegen eine geringere Relevanz. Das Ziel mit der geringsten Relevanz ist die Wartbarkeit. Darüber hinaus ist ein möglichst geringer Implementierungsaufwand notwendig – dessen Relevanz ist für den Entscheidungsträger genauso hoch wie die der Portabilität. Die Eigenschaftswerte der ADOdb-Alternative und der Alternative Entwicklung datenbankspezifischer Bibliotheken werden daher mit den in Tab. 6.4 dargestellten Faktoren gewichtet.

Konsequenz	Gewichtungsfaktor
Portabilität	0.40
Wartbarkeit	0.10
Zugriffszeiten	0.20
Implementierungsaufwand	0.30
Summe	**1.00**

Tab. 6.4: Gewichtungsfaktoren

In Tab. 6.5 sind die gewichteten Eigenschaftswerte der Typo3-Lösungsansätze dargestellt (hervorgehoben durch graue Einfärbung, die ungewichteten Werte sind in der jeweils linken Spalte dargestellt). Auf dieser Grundlage kann der Gesamtwert der Alternative datenbankspezifischer Bibliotheken durch Addition der Einzelwerte erfolgen. Auf rationaler Ebene wird diese Alternative mit 0.83 bewertet. Da bei der ADOdb-Alternative zwei Implementierungsvarianten zu berücksichtigen sind, kann noch kein Gesamtwert errechnet werden.

Alternativen	Portabilität (0.4)		Wartbarkeit (0.1)		Zugriffszeiten (0.2)		Aufwand (0.3)	
ADOdb-Bibliothek	0.75	0.30	1.00	0.10	0.75	0.15	1.00	0.30
ADOdb-Bibliothek mit ergänzender Optimierung	0.75	0.30	0.50	0.05	1.00	0.20	0.00	0.00
Entwicklung datenbank-spezifischer Bibliotheken	1.00	0.40	0.75	0.08	1.00	0.20	0.50	0.15

Tab. 6.5: Gewichtung der normalisierten Eigenschaftswerte

Berücksichtigung von Wahrscheinlichkeiten

Gemäß den Ausführungen in Kapitel 6.1.3 (siehe insbesondere Abb. 6.11 auf S. 107) sind bei der ADOdb-Alternative zwei Implementierungsvarianten zu unterscheiden: einmal ohne und einmal mit ergänzenden Optimierungen. Die Notwendigkeit der Optimierungen resultiert aus erhöhten Antwortzeiten auf die Datenbankzugriffe und einer drohenden Verschlechterung der Zugriffszeiten. Die Probleme treten mit einer Wahrscheinlichkeit von 60 % ein. Im Gegenzug kann erwartet werden, dass in 40 % aller Fälle keine Optimierungen erforderlich sind.

Tab. 6.6 fasst die Ermittlung des Erwartungswertes der ADOdb-Alternative zusammengefasst. Die Wahrscheinlichkeiten beider Implementierungsvarianten werden mit den Eigenschaftswerten multipliziert (siehe für die Ausgangwerte Tab. 6.5). In der Summe hat die ADOdb-Alternative einen Wert von 0.67 für den Entscheidungsträger. Sie ist damit von deutlich geringerem Wert als die Alternative datenbankspezifischer Bibliotheken.

Alternativen	Portabilität	Wartbarkeit	Zugriffszeiten	Aufwand
ADOdb-Bibliothek **(0.4)**	0.12	0.04	0.06	0.12
ADOdb-Bibliothek mit ergänzender Optimierung **(0.6)**	0.18	0.03	0.12	0.00
Entwicklung datenbank-spezifischer Bibliotheken	Bereits ermittelter Gesamtwert in Höhe von 0.83			

Tab. 6.6: Berücksichtigung von Wahrscheinlichkeiten

Entscheidung über die betrachteten Lösungsansätze

Durch die Anwendung des Entscheidungsprozesses wurden zwei Lösungsansätze zur Typo3-Architekturveränderung systematisch identifiziert, untersucht und bewertet. Die ADOdb-Alternative hat für den Entscheidungsträger einen Wert von 0.67, die Entwicklung von Bibliotheken mit datenbankspezifischen Zugriffsfunktionen wurde hingegen mit 0.83 bewertet. Für die Architekturveränderung bzw. die Restrukturierung des Datenbankzugriffs sollten daher datenbankspezifische Bibliotheken entwickelt und implementiert werden.

6.1.5 Konsequenzen der tatsächlichen Architekturveränderung von Typo3

Die im Rahmen der Entscheidungsfindung ermittelten Eigenschaftswerte und die Bewertungsergebnisse der beiden Alternativen decken sich mit den praktischen Erfahrungen, die bei der tatsächlichen Architekturveränderung von Typo3 gemacht wurden [Damb06a]. Bei der tatsächlichen Architekturveränderung wurde jedoch in Anbetracht der drängenden Portabilitätsprobleme die – nur scheinbar sinnvolle – ADOdb-Alternative implementiert. Mit der Version 4.0 kann Typo3 zwar in Kombination mit verschiedenen Datenbanken genutzt werden, die von der ADOdb-Bibliothek unterstützt werden. Dies sind u. a. MySQL, Oracle, Firebird und PostgreSQL in den jeweils aktuellen Versionen (siehe für eine vollständige Übersicht der unterstützten Datenbanken [Damb06b] und [ADOD06]).

Es hat sich jedoch gezeigt, dass mit der Nutzung der ADOdb-Alternative deutliche Einbußen bei den Zugriffszeiten verbunden sind. Mittels eines Benchmark-Tests wurden diese Einbußen quantifiziert [Benc04]. Dabei wurden die Zugriffszeiten von Typo3 auf eine MySQL- und eine Postgres-Datenbank in verschiedenen Konfigurationen (verschiedene ADOdb-Versionen, mit und ohne Optimierungen) getestet. Hierbei konnte ermittelt werden, dass sich die Zugriffszeiten auf eine MySQL-Datenbank über ADOdb-Zugriffsfunktionen fast verdoppelten (siehe Abb. 6.16 und Abb. 6.17 auf S. 117). Unter Nutzung von Optimierungsfunktionen (z. B. Turck MMCache [Turc06]) konnten die Zugriffszeiten wiederum reduziert werden.

Die Einbußen bei den Zugriffszeiten sowie die Notwendigkeit von Optimierungen hätten unter Anwendung des Entscheidungsprozesses bereits frühzeitig erkannt und vermieden werden können. Als Ausweg schlägt Typo3 Core Developer Karsten Dambekalns vor, die ADOdb-Funktionalität um weitere Handler zu ergänzen, "(...) die eher nativen Charakter haben und ohne die Nutzung von ADOdb auf die Zieldatenbank zugreifen (...)" [Damb06a]. Dies entspricht der zweiten betrachteten Alternative, der Entwicklung datenbankspezifischer Bibliotheken mit nativen Zugriffsfunktionen. Diese wäre auf rationaler Ebene geeigneter als die ADOdb-Alternative gewesen. Der Aufwand, der zur Implementierung und Optimierung der ADOdb-Alternative notwendig war, hätte vermieden werden können.

In Tab. 6.7 sind die Ergebnisse der Benchmark-Studie aufgeführt; das Benchmarking erfolgte durch den Vergleich der Datenbankzugriffe auf MySQL und PostgrSQL in unterschiedlichen Konfigurationen. Die Abb. 6.16 zeigt die Zugriffszeiten der einzelnen Datenbankzugriffe in grafischer Form und in Abb. 6.17 sind die Veränderungen der Zugriffszeiten dargestellt. Der Referenzpunkt ist die erste Messung der MySQL ohne Nutzung der ADOdb-Funktionalität. Die ausführlichen Protokolle sind in [Benc04] aufgeführt.

Datenbank-Konfiguration	***Anfragen je Sekunde***	***Millisekunden je Anfrage***
MySQL	12,59	396,98
MySQL, DBAL	6,32	791,62
*PostgreSQL, DBAL **	5,73	871,97
PostgreSQL, DBAL	5,74	871,38
*PostgreSQL, DBAL ***	5,66	883,13
*MySQL, DBAL ****	90,88	55,02
*PostgreSQL, DBAL ****	17,99	277,98
*PostgreSQL, DBAL **,****	18,16	275,37
* Anwendung von ADOdb 4.22 anstatt 4.51 ** mit ADOdb C Erweiterung 5.0.1 *** mit Turck MMCache 2.4.6		

Tab. 6.7: Benchmark Ergebnisse der TYPO3 DBAL Erweiterung

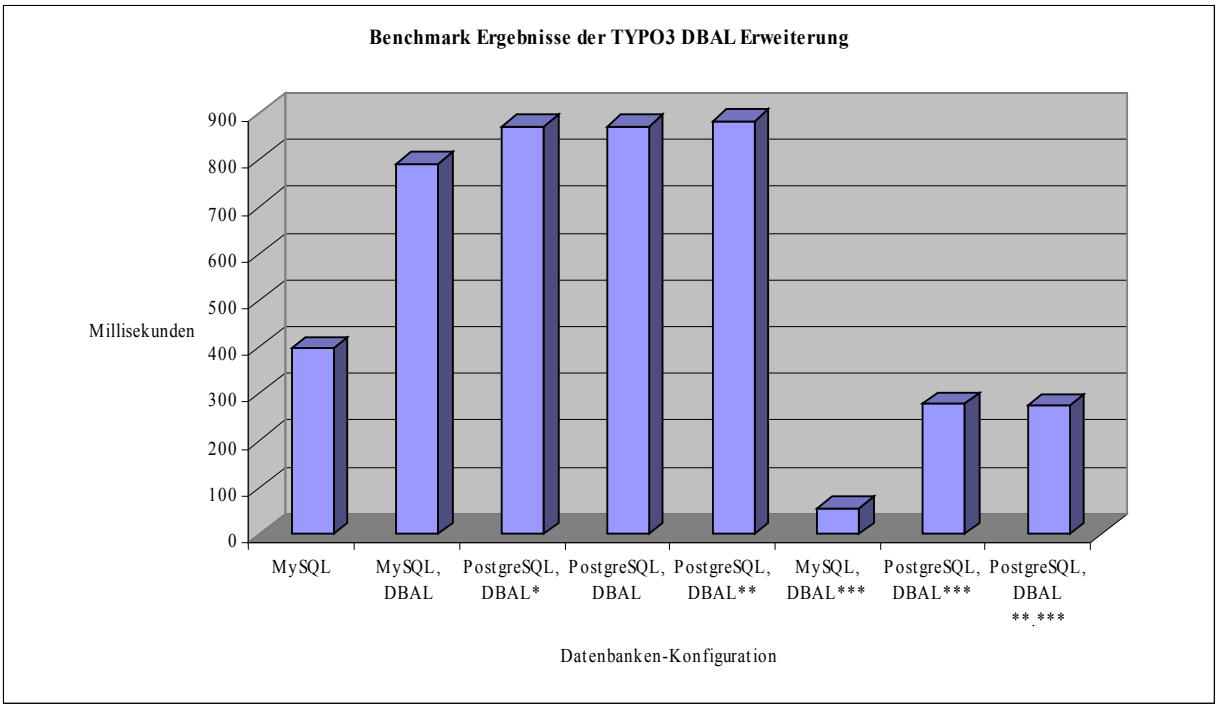


Abb. 6.16: Benchmark Ergebnisse der TYPO3 DBAL Erweiterung

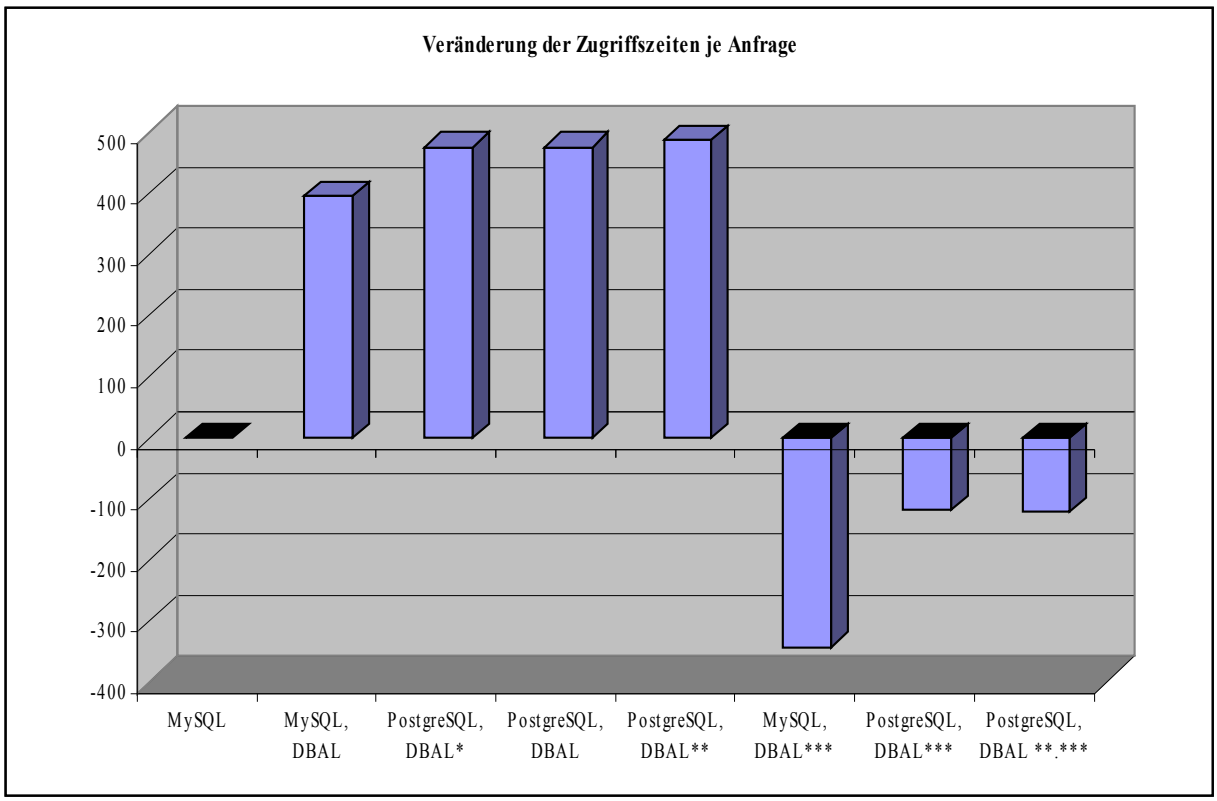


Abb. 6.17: Veränderung der Zugriffszeiten je Anfrage

6.2 Erhöhung der Sicherheit eines E-Banking-Systems

Das zweite Praxisbeispiel einer Architekturentscheidung ist die Verbesserung der Sicherheit eines E-Banking-Portals am Beispiel der Software *Multicash@online* [Omni08]. Die Software stellt über eine webbasierte Oberfläche den Zugang zum Transaktionssystem von Banken her, damit der Bankkunde seine Bankgeschäfte durchführen kann, wie z. B. Tätigung von Überweisungen, Einrichtung von Daueraufträgen oder die Abfrage des Kontostandes. Der in diesem Abschnitt beschriebene Entscheidungsprozess zur Architekturveränderung der Software *Multicash@online* steht stellvertretend für Architekturveränderungen bei vergleichbaren webbasierten E-Banking-Portalen, die im Privatkundengeschäft eingesetzt werden.

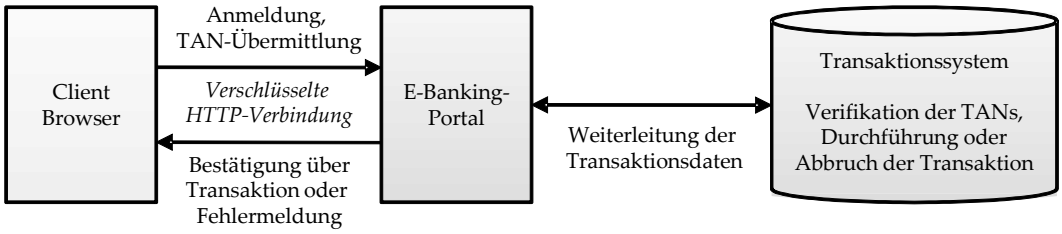


Abb. 6.18: Schematische Übersicht über den Zugriff auf ein webbasiertes E-Banking Portal

Das Ziel der betrachteten Architekturveränderung ist die Verbesserung der Sicherheit eines E-Banking-Systems für Privatkunden am Beispiel der webbasierten Software *Multicash@online*. Die wichtigsten IT-Sicherheitsziele sind die Wahrung der *Vertraulichkeit*, der *Verfügbarkeit* und der *Datenintegrität [Ecke05]*. Während Verfügbarkeit und Integrität durch moderne IT-Konzepte, z. B. redundante Rechnersysteme mit hoher Ausfallsicherheit und starke Verschlüsselungsalgorithmen, weitgehend erfüllt werden können, stellt die Vertraulichkeit eine besondere Herausforderung dar. Die Vertraulichkeit wird bei der Software *Multicash@online* durch den passwortgeschützten Zugriff auf das Konto sowie die Absicherung der Transaktionen, z. B. Überweisungen oder die Einrichtung von Daueraufträgen, durch geheime und persönliche Transaktionsnummern (TAN) sichergestellt. Eine TAN entspricht dabei einer elektronischen Unterschrift. Wie in Abb. 6.18 dargestellt, verifiziert das Transaktionssystem die TAN und führt die gewünschte Transaktion durch oder bricht diese ab, wenn eine ungültige TAN eingegeben und übermittelt wurde. Die Verbindung zum Client-Browser erfolgt über eine verschlüsselte HTTP-Verbindung. Diese Architektur eines webbasierten E-Banking-Systems und der Prozess zur Verifizierung der TAN ist in vergleichbarer Form bei nahezu allen deutschen Banken im Privatkundenbereich vorzufinden [Ste+02]. Gerade die TAN sind jedoch der Schwachpunkt für die Vertraulichkeit, da sie z. B. gestohlen oder ausgespäht (sog. Phishing) werden können. Ein Angreifer hat somit vollen Zugriff auf das Konto; die Erfahrungen aus der Praxis zeigen, dass Überweisungen auf Auslandskonten oftmals nicht mehr widerrufen und zurückgebucht werden können. Die Vielzahl an erfolgreichen Phishing-Vergehen, bei denen TAN illegal abgefragt und ausgespäht werden, zeigt die Bedeutung dieser Schwachstelle [Reim07].

Im Gegensatz zur Architekturentscheidung beim Open-Source-Softwareprodukt Typo3 (siehe Abschnitt 6.1, S. 90ff.) sind webbasierte E-Banking-Systeme individuelle Softwarelösungen, die aus Sicherheitsgründen hohen Sicherheits- und Zertifizierungsanforderungen entsprechen müssen. Das hat in der Praxis zur Folge, dass die Sicherheit schon bei der Gestaltung der Softwarearchitektur eine große Rolle spielt und im Rahmen der Entscheidungsfindung, z. B. bei der Auswahl und der Konkretisierung der alternativen Lösungsansätze, entsprechend tiefgründig sowie detailliert berücksichtigt werden muss. Die Erfahrungen aus der Praxis zeigen außerdem, dass bei der Entscheidungsfindung aus Sicherheits- und Geheimhaltungsgründen nicht alle Informationen über das Softwaresystem im erforderlichen Detaillierungsgrad zur Verfügung stehen, wie z. B. der Quelltext oder die Schnittstellen-Definitionen sicherheitskritischer Komponenten [Reim07]. In diesen Fällen können die betreffenden Komponenten der Softwarearchitektur, im Gegensatz zur detaillierten Typo3-Architekturanalyse, nur auf einer abstrakten, generischen Ebene betrachtet und analysiert werden. Die Problematik fehlender Informationen über das zugrunde liegende Softwaresystem ist aber nicht nur auf sicherheitskritische Softwaresysteme beschränkt. Auch bei über Jahre gewachsenen Softwaresystemen (sog. Legacy-Systems) existieren in der Praxis oft unbekannte Systemteile oder Komponenten, die nicht mehr rekonstruiert werden können, da der Quelltext oder das entsprechende Entwickler-

Know-how nicht mehr zur Verfügung stehen. Dies ist ein wesentlicher Unterschied zur vorangegangenen Architekturentscheidung bei der Open-Source-Softwareanwendung Typo3, bei der der Sourcecode frei zugänglich und das Wissen über interne Zusammenhänge und Strukturen über viele Entwickler verteilt ist.

6.2.1 Phase 1 – Identifizierung der Ziele und Rahmenbedingungen

Identifizierung der Ziele

In der ersten Phase der Entscheidungsfindung (siehe Abb. 3.1 auf S. 44) sind die Ziele systematisch zu identifizieren und widerspruchsfrei zu strukturieren. Das ist einerseits zur Erkennung 'sich ergänzender' oder 'miteinander konkurrierender' Zielbeziehungen notwendig, für die eine klare, eindeutige und detaillierte Ziel-Beschreibung erforderlich ist (siehe zu den Zielbeziehungen Abschnitt 4.1, S. 51ff.). Andererseits ist die klare und detaillierte Ziel-Beschreibung die Grundlage für eine systematische Auswahl der besten Lösungsansätze. Die Ziele werden im weiteren Verlauf des Abschnitts durch Unterstreichung hervorgehoben.

Das dominierende und fundamentale Ziel der Architekturveränderung des webbasierten E-Banking-Systems *Multicash@online* ist die Verbesserung der Sicherheit, insbesondere die Verbesserung der Vertraulichkeit bei der Authentifizierung und Verifizierung der Transaktionen. Um die Vertraulichkeit zu verbessern, ist eine höhere Sicherheit beim Zugriff auf das Transaktionssystem zu ermöglichen, indem das TAN-Listen-Verfahren durch ein alternatives Verfahren ersetzt wird. In Abb. 6.19 ist einerseits das aktuelle TAN-Listen und andererseits das alternative Verfahren zur Client-Authentifizierung dargestellt. Aktuell steht dem Bankkunden eine Liste mit mehreren TANs zur Verfügung, die er beliebig oder anhand eines entsprechenden Indexes (sog. indexierte TAN-Listen, iTAN) zur Authentifizierung seiner Aufträge nutzen kann. Ziel der Architekturentscheidung ist es, die beste Alternative zur Authentifizierung der Aufträge zu ermitteln und auszuwählen, um sie im Anschluss implementieren zu können.

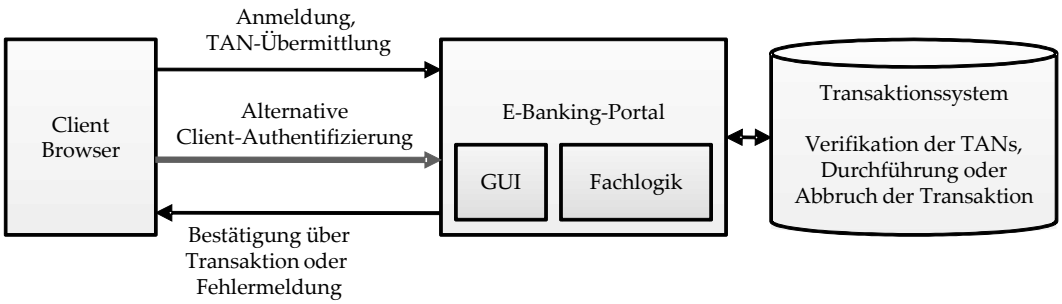


Abb. 6.19: Übersicht über die geplante Architekturveränderung

Schon jetzt kann abgeschätzt werden, dass im Rahmen der Entscheidungsfindung eine Folge mehrere Lösungsansätze berücksichtigt werden muss. Aufgrund der Vielzahl beteiligter Systeme (u. a. Transaktionssystem, E-Banking-Portal etc.) kann ein alternatives Verfahren zur Client-Authentifizierung nicht durch einen einzelnen Veränderungsschritt implementiert werden. Die kombinatorische Komplexität sowie die daraus resultierenden Risiken und Unsicherheiten erfordern ein Vorgehen in mehreren Schritten. Bei der Architekturveränderung des E-Banking-Portals handelt es sich daher um eine mehrstufige Entscheidung über eine Kombination von Lösungsansätzen.

Neben der Verbesserung der Sicherheit darf die Wartbarkeit des Softwaresystems, insbesondere des E-Banking-Portals, nicht beeinträchtigt werden. Das E-Banking-Portal, veranschaulicht in Abb. 6.19, besteht einerseits aus der Benutzeroberfläche, die im Browser des Clients dargestellt wird, und andererseits aus einer Fachlogik (implementiert als Applikationsserver), um beispielsweise die Anfragen an das Transaktionssystem weiterzuleiten oder aktuelle und personalisierte Mitteilungen abzubilden. Das Portal muss permanent weiterentwickelt werden, u. a. aufgrund veränderter Sicherheitsanforderungen. Dafür ist es notwendig, dass die Analysier- und Testbarkeit des E-Banking-Portals möglichst in unveränderter Form

beibehalten werden. Eine wichtige Grundlage hierfür ist einerseits die lose Kopplung zwischen der Benutzeroberfläche des E-Banking-Portals mit dem Browser auf der Client-Seite. Andererseits sollte eine lose Kopplung zwischen Fachlogik und Transaktionssystem angestrebt werden, um umfangreiche sowie aufwendige Anpassungen an den notwendigen Sicherheitstests zu vermeiden.

Das dritte Fundamentalziel ist die Beibehaltung der Transaktionszeiten. Durch die Architekturveränderung darf die Zeit, die für die Ausführung einer Transaktion benötigt wird, nicht erhöht werden. Im Gegensatz zur Sicherheit und Wartbarkeit sind die Transaktionszeiten im Privatkundengeschäft zwar von geringerer Bedeutung, da die Erfahrungen aus der Praxis zeigen, dass Privatkunden im Gegensatz zu Firmenkunden deutlich weniger Transaktionen vornehmen. Die Durchführungszeit, die für die Abarbeitung einer Transaktion benötigt wird, ist jedoch ein wichtiger Faktor für die Akzeptanz des E-Banking-Systems auf der Kundenseite.

Die bereits existierende Funktionalität des E-Banking-Portals, vor allem im Hinblick auf die Durchführung und das Management einer Kunden-Transaktion, darf durch die Architekturveränderung nicht verändert oder eingeschränkt werden. So muss es beispielsweise auch nach erfolgter Restrukturierung immer noch möglich sein, eine Transaktion abzubrechen oder zu korrigieren. Zudem ist der Implementierungsaufwand so gering wie möglich zu halten.

Strukturierung der Ziele

Um Abhängigkeiten und widersprüchliche Ziel-Beziehungen erkennen und auflösen zu können, werden die Ziele im Anschluss an die Identifizierung strukturiert und in ein Ziel-Wirkungsmodell eingeordnet. Das Verfahren zur Strukturierung umfasst die folgenden vier Schritte:

- *Schritt 1 – Verfeinerung grober und ungenauer Ziele:* Die im vorangegangenen Abschnitt beschriebenen Ziele sind ausreichend detailliert, um auf dieser Grundlage geeignete Lösungsansätze auszuwählen, vergleichen und bewerten zu können.
- *Schritt 2 – Identifizierung sich ergänzender Ziel-Mittel-Beziehungen:* Die identifizierten Ziele werden unter Anwendung der Klassifikationsmethode aus der Entscheidungstheorie nach Instrumental- und Fundamentalzielen klassifiziert. Dabei wird ein Ziel-Wirkungsmodell erstellt (siehe Abb. 6.20). Das dominierende Fundamentalziel ist die Verbesserung der 'Sicherheit', speziell die 'Vertraulichkeit'. Um die 'Vertraulichkeit' zu verbessern, ist ein 'sicherer Zugriff auf das Transaktionssystem' zu ermöglichen (erstes Instrumentalziel), indem das TAN-Verfahren zur Verifikation der Transaktionen durch ein 'Verfahren mit höherer Sicherheit' ersetzt wird. Die 'Wartbarkeit' ist das zweite Fundamentalziel der Architekturentscheidung. Im Rahmen der Architekturveränderung ist darauf zu achten, dass die 'Analysier- und Testbarkeit' des Softwaresystems erhalten bleiben. Das dritte Fundamentalziel 'Transaktionszeiten' hat im Vergleich zu den beiden anderen Fundamentalzielen die geringste Priorität. Dennoch sind die Durchführungszeiten der Transaktionen entsprechend den Ausführungen im vorangegangenen Abschnitt ein wichtiger Faktor für die Akzeptanz des E-Banking-Systems und dürfen daher im Vergleich zum unveränderten E-Banking-System nicht negativ beeinträchtigt werden.
- *Schritt 3 – Auflösung widersprüchlicher und konkurrierender Zielbeziehungen:* Das Ziel 'Transaktionszeiten' konkurriert mit den Sicherheits-Zielen, da sich eine Ersetzung des TAN-Verfahrens zur Verbesserung der 'Vertraulichkeit' direkt auf die Durchführungszeiten für eine Kunden-Transaktion auswirken kann. Die 'Transaktionszeiten' sind bei der betrachteten Architekturentscheidung im Vergleich zur 'Sicherheit' nur von geringer Relevanz, da im Privatkundengeschäft deutlich weniger Transaktionen durchgeführt werden als im Firmenkundengeschäft. Daher wird lediglich festgelegt, dass die Durchführungszeiten nicht negativ verändert werden dürfen.
- *Schritt 4 – Beschränkung auf die wichtigsten Ziele:* Eine Priorisierung und Einschränkung der Ziele ist nicht notwendig, da im Rahmen der Entscheidung lediglich drei Fundamentalziele zu berücksichtigen sind.

In Abb. 6.20 sind die Fundamental- und Instrumentalziele der Architekturentscheidung in Form eines Ziel-Wirkungsmodells dargestellt. Die unterschiedliche Einfärbung der Fundamentalziele spiegelt deren Bedeutung und Gewichtung wider.

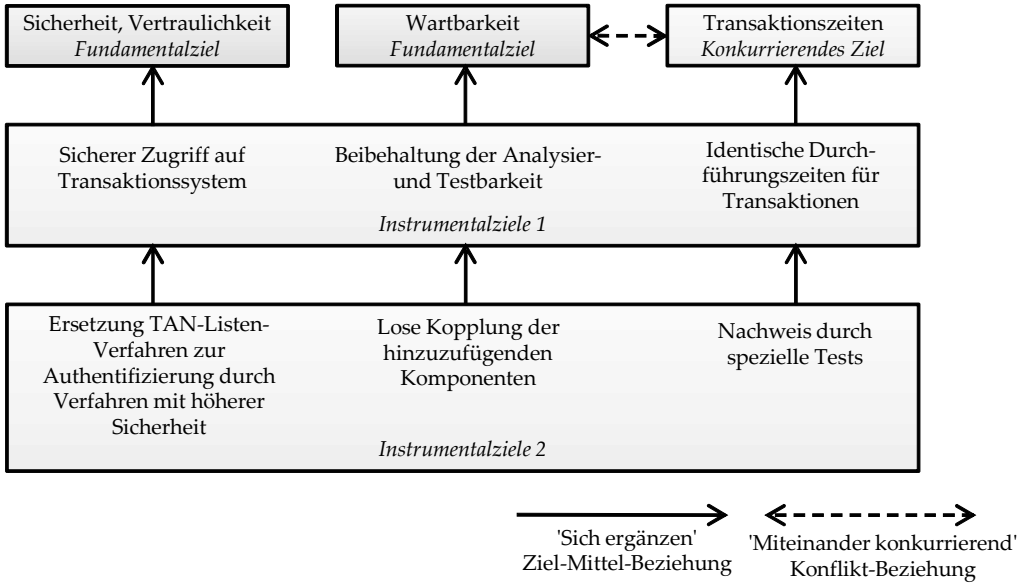


Abb. 6.20: Ziel-Wirkungsmodell für die Restrukturierung des E-Banking-Systems

Erhebung der Rahmenbedingungen

Neben den Zielen stellen die organisatorischen und technischen Rahmenbedingungen wichtige Entscheidungskriterien dar, die insbesondere bei der Auswahl, der Verfeinerung und der Konkretisierung der alternativen Lösungsansätze berücksichtigt werden müssen. Die Architekturveränderung des webbasierten E-Banking-Systems *Multicash@online* unterliegt den in der folgenden Liste aufgeführten Rahmenbedingungen. Da detaillierte Informationen über das Transaktionssystem und den Applikationsserver nicht zur Verfügung stehen, kann nicht im Detail festgelegt werden, welche Entwicklerressourcen und welche Werkzeugumgebung für die Architekturveränderung benötigt werden. Die groben Vorgaben werden daher im weiteren Verlauf der Entscheidungsfindung konkretisiert.

- *Organisatorische Rahmenbedingungen:*
  - *Managementvorgaben*: Die E-Banking-Software ist für die betreffenden Banken ein geschäftskritisches System, welches den Kunden möglichst ohne Unterbrechungen zur Verfügung stehen soll. Es ist daher ein Lösungsansätze auszuwählen, deren Implementierung keine oder nur eine kurze Unterbrechung des Produktivbetriebs erfordert.
  - *Personelle Kapazität, Finanzieller Spielraum*: Aufgrund der Kritikalität des E-Banking-Systems wird den Sicherheitsverbesserungen eine hohe Bedeutung eingeräumt. Es werden daher jene Lösungsansätze bevorzugt, die möglichst kurzfristig implementiert werden können. Dabei ist jedoch ist ein minimaler Aufwand auf der Client-Seite anzustreben. Da Sicherheitsverbesserungen vom Bankkunden selten als zusätzliches Produktmerkmal wahrgenommen werden, dürfen für den Bankkunden keine oder nur geringe zusätzliche Kosten anfallen. Außerdem sollten sich die Veränderungen an der Benutzerführung auf ein Minimum beschränken und mit geringem zeitlichen Aufwand erlernbar sein [Ste+02].
- *Technische Rahmenbedingungen:*
  - *Hardware, Softwaretechnologien:* Für den Client sollte möglichst keine neue Hardware sowie Software notwendig sein. Auch nach der Architekturveränderung sollte der Client mit einem beliebigen Browser auf das E-Banking-System über das HTTP-Protokoll zugreifen können.

Zur Reduzierung von Risiken erfolgt eine schrittweise Implementierung des alternativen Verfahrens zur Client-Authentifizierung. Zuerst werden in einer Pilotphase nur ausgewählte Bankkunden das alternative TAN-Verfahren testen können. Bei erfolgreichem Abschluss der Pilotphase wird der Kreis der Kunden, die das neue Verfahren nutzen, schrittweise vergrößert. Das hat zur Folge, dass das bereits existierende TAN-Verfahren und der neu implementierte Lösungsansatz in der Übergangszeit parallel nutzbar sein müssen.

- *Architekturtechnologien*: Wie jede Webanwendung ist das E-Banking-System in eine Client-Server-Struktur aufgeteilt. Ein Client ruft über den Browser die Benutzeroberfläche des E-Banking-Systems auf. Das Portal ist als dynamische Webseite implementiert. Über eine Brücke wird von der Portal-Seite aus auf das Transaktionssystem zugegriffen (siehe hierzu die Ausführungen im folgenden Abschnitt und insbesondere in Abb. 6.21 auf S. 123). Gerade zum Transaktionssystem und zum Applikationsserver stehen aus Sicherheits- und Geheimhaltungsgründen jedoch keine detaillierten Informationen zur Verfügung.
- *Standards:* Bei der Architekturveränderung ist zu beachten, dass die Datenübertragung zwischen Client und Portal über das HTTP-Protokoll erfolgt, welches mittels SSL verschlüsselt wird.
- *Werkzeugumgebung:* Im Gegensatz zum Web-Content-Management-System Typo3 sind E-Banking-Softwaresysteme, wie beispielsweise das *Multicash@online*, individuell erstellte Softwaresysteme, die mit individuell angepassten Werkzeugen entwickelt wurden. Da im Rahmen der Architekturveränderung ein bereits existierendes webbasiertes E-Banking-System verändert wird, kann davon ausgegangen werden, dass die bereits vorhandenen Werkzeuge zur Veränderung der Architektur wiederverwendet werden können.

6.2.2 Phase 2 – Architekturbeschreibung und Risikoeinstufung

Im Anschluss an die Identifizierung sowie Strukturierung der Ziele und Rahmenbedingungen erfolgt in der zweiten Phase der Entscheidungsfindung die Beschreibung der Architektur des *Multicash@online*-E-Banking-Systems (siehe zu den Phasen des Entscheidungsprozesses Abb. 3.1 auf S. 44). Die Architekturbeschreibung ist erforderlich, um auf der Grundlage der klaren und eindeutigen Beschreibung der Problembereiche und Schwachstellen einerseits alternative Lösungsansätze zur Architekturveränderung auswählen sowie andererseits Abhängigkeiten zwischen den Komponenten des E-Banking-Systems erkennen zu können.

Einstufung des Risikos einer Architekturentscheidung

Zunächst erfolgt die Einstufung des Geschäftsrisikos, um ableiten zu können, wie detailliert die Beschreibung und wie tiefgründig die Analysen bei der Vorauswahl, der Verfeinerung sowie der Konkretisierung der Lösungsansätze sein müssen. Ein solches E-Banking-Portal – *Multicash@online* ist dabei nur ein Beispiel – ist für eine Bank ein geschäftskritisches Softwaresystem, da die Bankkunden über das Portal ihre Aufträge, z. B. Überweisungen, an die Bank übermitteln. Da jedoch das bereits existierende TAN-Verfahren während der Übergangszeit parallel zum neu implementierten Verfahren genutzt werden kann, sind ein Rücksprung oder ein Wechsel zu einer lauffähigen Version jederzeit möglich. Die Architekturveränderung ist entsprechend der in Tab. 5.1 (siehe S. 63) aufgeführten Kriterien der Risikoklasse 1 zuzuordnen.

Beschreibung der relevanten Teile der Architektur

Die Grundlage für die Auswahl und Beschreibung der relevanten Teile der Architektur des E-Banking-Systems ist die Analyse der zur Verfügung stehenden Informationen. Wie alle derzeit bekannten E-Banking-Portale für das Privatkundengeschäft ist das Portal aus Sicherheitsgründen Closed-Source. Eine detaillierte Analyse des Quelltextes, wie im Beispiel der Typo3-Architekturveränderung (siehe Abschnitt 6.1.2, S. 96f.), kann daher nicht erfolgen. Es steht jedoch eine informale Beschreibung der Architektur des *Multicash@online* Portals zur Verfügung (siehe Abb. 6.21 [Omni08]). Trotz des informalen Charakters der Architekturbeschreibung – informale Beschreibungen sind als Entscheidungsgrundlagen in der Praxis durchaus üblich – können die wesentlichen und für die Entscheidung relevanten Komponenten des E-Banking-Systems identifiziert und beschrieben werden.

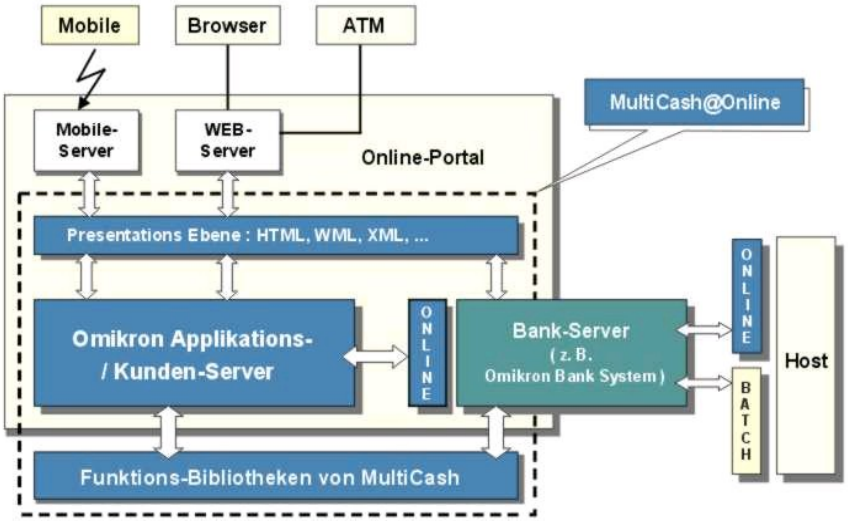


Abb. 6.21: Informale Architekturbeschreibung der 'Multicash@online'-Architektur

Im Anschluss erfolgt die Abgrenzung der für die Architekturentscheidung relevanten Teile der Architektur des E-Banking-Systems. Für die anstehende Architekturveränderung sind jene Komponenten zu beschreiben, die an der Übermittlung und der Verifikation der TAN beteiligt sind (siehe Abb. 6.21 und Abb. 6.22):

- *Portal-GUI (Präsentationsebene):* Die Portal-GUI umfasst die Benutzeroberfläche, über die der Client bzw. der Bankkunde das System bedient. Die Benutzeroberfläche ist als dynamische Webseite implementiert und beinhaltet verschiedene HTML-Darstellungselemente.
- *Client-Browser, Webserver:* Die Benutzeroberfläche des Portals wird durch den Webserver dynamisch generiert und im Browser des Clients dargestellt. Die Kommunikation zwischen dem Browser des Clients und dem Webserver erfolgt über eine verschlüsselte HTTP-Verbindung.
- *Portal-Fachlogik (Applikationsserver) und Funktions-Bibliotheken:* Die eingegebenen Daten des Clients werden vom Applikationsserver verarbeitet; dabei wird auf die Funktionsbibliotheken des *Multicash@online*-Softwaresystems zurückgegriffen. So steht z. B. eine Funktionsbibliothek zur Verfügung, um nach der Anmeldung des Bankkunden am System zu überprüfen, ob aktuelle persönliche Mitteilungen vorliegen.
- *Transaktionssystem (Bank-Server, Host):* Das Portal ist an das Transaktionssystem der Bank über eine Schnittstelle angeschlossen. Das Transaktionssystem kann als monolithischer Teil der Architektur aufgefasst werden. Veränderungen am Transaktionssystem können aus Geheimhaltungs- und Sicherheitsanforderungen nur durch die Bank selbst erfolgen.

Da die betreffenden Teile der Architektur als eigenständige Komponenten implementiert sind, erfolgt eine Beschreibung in Form eines UML-Komponentendiagramms, das in Abb. 6.22 dargestellt ist.

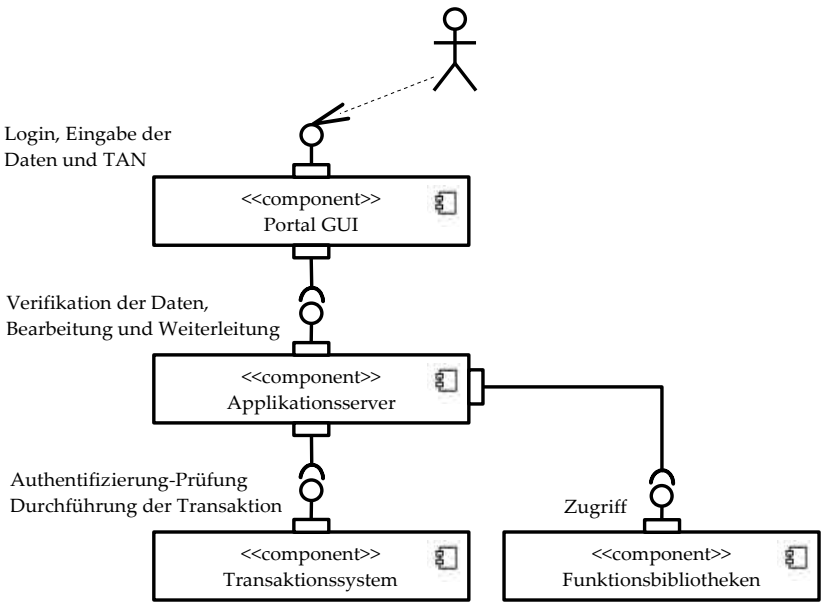


Abb. 6.22: Relevante Komponenten der Architektur des E-Banking-Systems

6.2.3 Phase 3 – Vorauswahl und Konkretisierung alternativer Lösungsansätze

Aufstellung eines Grobplanes der Architekturveränderung

Schon bei der Strukturierung der Ziele und bei der Aufstellung des Ziel-Wirkungsmodells (siehe Abb. 6.20, S. 121) wurde erkannt, dass die Restrukturierungen zur Verbesserung der Sicherheit des E-Banking-Systems *Multicash@online* nicht in einem einzelnen Schritt durchgeführt werden können. Aus der Vielzahl beteiligter Systeme (u. a. Transaktionssystem, E-Banking-Portal etc.) resultiert eine hohe kombinatorische Komplexität mit beachtlichen Risiken und einer Vielzahl von Unsicherheiten. Zur Reduzierung der kombinatorischen Komplexität ist ein Vorgehen in mehreren Schritten erforderlich; es handelt sich somit um eine mehrstufige Entscheidung über eine Kombination von Lösungsansätzen.

Erstellung alternativer Lösungsszenarien

Bei mehrstufigen Architekturentscheidungen ist zur Reduzierung der Komplexität und der daraus resultierenden Unsicherheiten und Risiken zunächst ein Grobplan aufzustellen (siehe die Argumentation in Abschnitt 5.3.1, S. 67ff.). Der erste Schritt hierbei ist die Skizzierung von Lösungsszenarien, um zu erkennen, mit welchen Architekturveränderungen die Fundamentalziele der Entscheidung – Sicherheit, Wartbarkeit und Transaktionszeiten – im gewünschten Umfang erfüllt werden können.

Das erste Lösungsszenario beschreibt ein Verfahren, bei der die TAN erst bei der Durchführung der Transaktion auf der Seite des Bankkunden generiert werden (Abb. 6.23). Bisher steht dem Bankkunden eine Liste mit mehreren TANs zur Verfügung, die er beliebig oder anhand eines entsprechenden Indexes (sog. indexierte TAN-Listen, iTAN) zur Authentifizierung seiner Aufträge nutzen kann. Da bei Verlust der TAN-Liste ein nahezu unbeschränkter Zugriff auf das Konto des Bankkunden möglich ist, werden die TANs in diesem Lösungsszenario nach der Eingabe eines Auftrages im E-Banking-Portal generiert und an den Bankkunden übermittelt. Das kann per Nachrichtenversand in einem Mobilfunknetz (SMS) oder per E-Mail erfolgen. Der 'TAN-Generator' umfasst dabei den relevanten Algorithmus, der bereits zur Generierung der TAN-Listen dient. Vom E-Banking-Portal aus wird die Generierung der TANs beauftragt und die generierte TAN wird an den Bankkunden über einen geeigneten Übertragungsweg übermittelt.

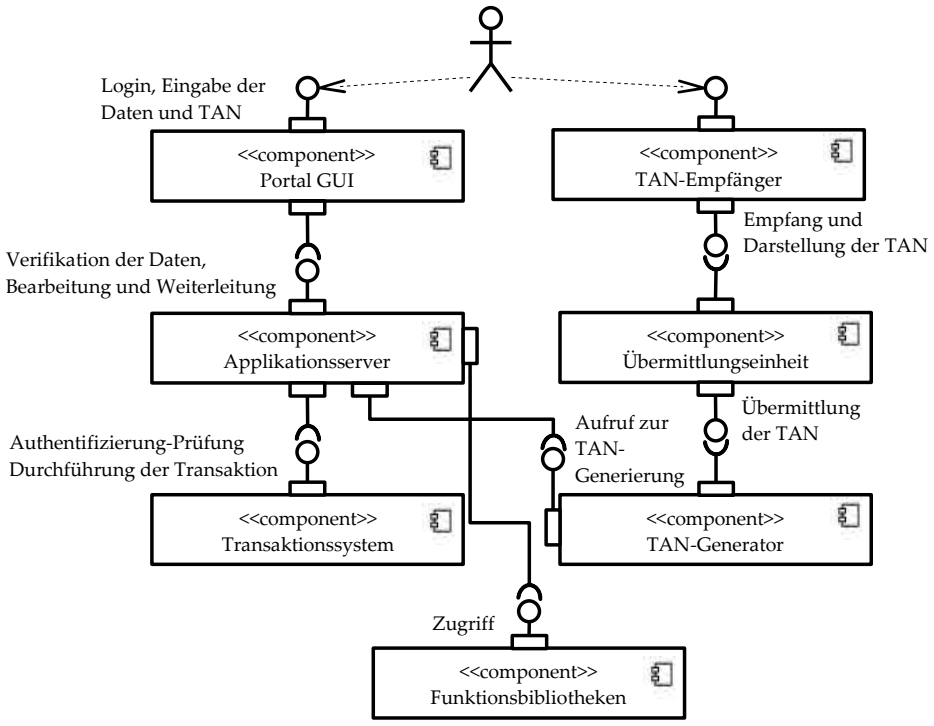


Abb. 6.23: Skizze des Lösungsszenarios 'serverbasierte TAN-Generierung'

Das zweite Lösungsszenario beschreibt ebenfalls die Generierung der TAN bei der Durchführung der Transaktion durch den Bankkunden (siehe Abb. 6.24). Jedoch ist der Algorithmus in einem Gerät oder einer eigenständigen Anwendung implementiert. Der Benutzer generiert die TAN auf Knopfdruck selbst. Da, wie auch bei einer TAN- oder iTAN-Liste, das Gerät oder der Rechner mitsamt der Anwendung zur TAN-Generierung gestohlen werden kann, ist ein zusätzlicher Schutzmechanismus erforderlich, um die TAN-Generierung starten zu können. Dies kann beispielsweise durch ein Codewort realisiert werden, welches dem Bankkunden vom E-Banking-Portal mitgeteilt wird; erst nach Eingabe des Codewortes können auf der Seite des Bankkunden gültige TANs generiert werden.

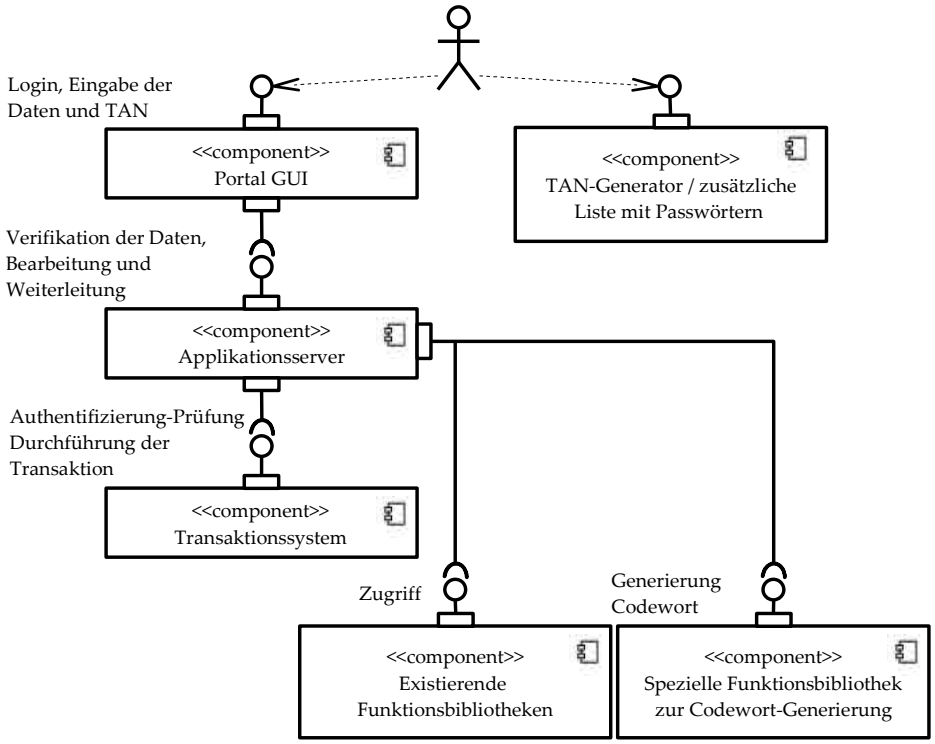


Abb. 6.24: Skizze des Lösungsszenarios 'clientbasierte TAN-Generierung'

Überprüfung der Lösungsszenarien und Ermittlung der Differenzen

Im Anschluss an die Skizzierung der Lösungsszenarien erfolgt im zweiten Schritt die Überprüfung im Hinblick auf die Ziele der Entscheidung sowie die Ermittlung der Unterschiede zwischen den Lösungsszenarien und dem Modell der existierenden Architektur. Dieser Schritt ist notwendig, um einerseits zu überprüfen, ob mit den skizzierten Lösungsszenarien die Ziele im erforderlichen Umfang erfüllt werden können oder ob Variationen und Anpassungen vonnöten sind. Andererseits ist der grobe Umfang der Restrukturierungsmaßnahmen zu ermitteln.

Im Rahmen der Überprüfung der Lösungsszenarien im Hinblick auf die Zielerreichung ist vor allem das Ziel 'sicherer Zugriff auf das Transaktionssystem' zu kontrollieren, da es die höchste Priorität für die Entscheidungsfindung besitzt (siehe das Ziel-Wirkungsmodell in Abb. 6.20, S. 121). Die Ziele 'Wartbarkeit' und 'Transaktionszeiten' werden in der dritten Phase des Entscheidungsprozesses, im Rahmen der Verfeinerung und Konkretisierung der Lösungsansätze, im Detail überprüft, da hierfür detaillierte Angaben über die Komponentenstruktur und die Abhängigkeitsbeziehungen der Lösungsszenarien erforderlich sind. Ein Walkthrough der Lösungsszenarien führt zu dem Ergebnis, dass das TAN-Listen-Verfahren durch ein TAN-Generierungs-Verfahren ersetzt werden kann; dabei ist es gleich, ob die Generierung serverseitig oder clientseitig erfolgt. Durch das Generierungs-Verfahren kann die Sicherheit bei der Client-Authentifizierung erhöht werden, da die TANs speziell für einen Auftrag generiert werden. Die TAN-Listen, die bei Diebstahl einen nahezu vollständigen Zugriff auf das Konto des Betroffenen ermöglichen, können ersetzt werden.

Die ermittelten Unterschiede zwischen den beiden Lösungsszenarien und dem Modell der existierenden Architektur des E-Banking-Systems sind in Tab. 6.8 und Tab. 6.9 aufgeführt. Die Differenzen entsprechen den Maßnahmen zur Weiterentwicklung der Architektur des E-Banking-Systems.

	Differenzen zwischen Lösungsszenario 1 'serverbasierte TAN-Generierung' und existierender Architektur
1	Serverbasierte Generierung der TANs bei Eingabe des Auftrags
2	Übermittlung der TANs an ein Endgerät des Clients
3	Einsatz eines separaten Endgerätes für die Darstellung der TANs, z. B. ein Mobilfunkgerät
4	Kapselung des Algorithmus in separater Komponente sowie Integration in E-Banking-System

Tab. 6.8: Differenzen zwischen Lösungsszenario 1 und existierender Architektur

	Differenzen zwischen Lösungsszenario 2 'clientbasierte TAN-Generierung' und existierendem Architekturmodell
1	Clientbasierte Generierung der TANs bei Eingabe des Auftrags
2	Zusätzliche Generierung eines Codewortes auf Anforderung des Bankkunden
3	Einsatz eines separaten Endgerätes für die Generierung und die Darstellung der TANs, z. B. ein Mobilfunkgerät oder ein Token-Generator (z.B. [Reim07], [Viss07] und [Bwba08])
4	Kapselung des Algorithmus in separater Komponente sowie Integration in Endgerät

Tab. 6.9: Differenzen zwischen Lösungsszenario 2 und existierender Architektur

Berücksichtigung und Auflösung von Abhängigkeitsbeziehungen

Bevor der Grobplan der Architekturveränderung aufgestellt werden kann, sind zunächst die Abhängigkeitsbeziehungen zwischen den in Tab. 6.8 und Tab. 6.9 dargestellten Maßnahmen zu ermitteln. Dieser Schritt ist für die Reihenfolgeplanung der Maßnahmen innerhalb des Grobplans notwendig. Die Grundlage für die Ermittlung der Abhängigkeitsbeziehungen sind die in Abb. 6.23 und Abb. 6.24 (siehe S. 125f.) beschriebenen Komponentenbeziehungen.

Für das erste Lösungsszenario 'serverbasierte TAN-Generierung' stellt der Algorithmus zur Generierung der TANs die Wurzel in der Hierarchie der 'benutzt'-Beziehungen dar (siehe zu den Abhängigkeitsbeziehungen S. 51ff.). Der Algorithmus ist in einem Generator zu implementieren, bevor die weiteren Architekturveränderungen vorgenommen werden. Dazu zählt einerseits die Anbindung des Generators an den Applikationsserver, damit eine TAN auf Anforderung des Kunden generiert werden kann. Andererseits muss ein geeignetes Übermittlungsverfahren implementiert werden, um die generierte TAN an ein Empfangsgerät auf der Seite des Clients zu übermitteln. Solch ein Empfangsgerät kann beispielsweise ein Mobiltelefon sein.

Bei der 'clientbasierten TAN-Generierung', dem zweiten Lösungsszenario, erfolgt die Generierung der TAN in einem entsprechenden Endgerät auf der Clientseite, z. B. durch ein spezielles Softwareprogramm oder einen Hardware-Generator. Der Algorithmus zur TAN-Generierung stellt wiederum die Wurzel der 'benutzt'-Beziehungen dar. Um den Generator auf der Clientseite entwickeln zu können, ist der Generierungs-Algorithmus zunächst in einer Komponente zu kapseln. Erst im Anschluss können die Anpassungen am Applikationsserver und die Entwicklung eines geeigneten Verfahrens zur Überprüfung des Codewortes erfolgen, welches zur Generierung der TAN in das Endgerät eingegeben werden muss und eine erhöhte Sicherheit bietet.

Priorisierung und Aufstellung des Grobplanes

Auf der Grundlage der Ermittlung der Unterschiede zwischen den Lösungsszenarien (siehe Tab. 6.8 und Tab. 6.9) und der im vorangegangenen Schritt identifizierten Abhängigkeitsbeziehungen kann der folgende Grobplan der anstehenden

Architekturveränderung mit den Zielen 'Sicherheit', 'Wartbarkeit' und 'Transaktionszeiten' (siehe Abschnitt 6.2.1, S. 119ff.) erstellt werden. Da drei grobe Maßnahmenfolgen für die Architekturveränderung notwendig sind, wird die Entscheidung, welche Lösungsansätze für welche Maßnahmenfolge geeignet sind, in drei Stufen aufgeteilt. Eine Verfeinerung der Maßnahmenfolgen erfolgt im Rahmen der anschließenden Auswahl und Verfeinerung der Lösungsansätze je Entscheidungsstufe. Für jede der drei Maßnahmenfolgen werden außerdem auf der Grundlage des Ziel-Wirkungsmodells (siehe Abb. 6.20, S. 121) die relevanten Zwischenziele ausgewählt:

- *Stufe 1:* Auswahl und Verfeinerung alternativer Lösungsansätze für TAN-Generierung und der Übertragung

  Zwischenziele: Sicheres Verfahren bei der TAN-Generierung, lose Kopplung (Wartbarkeit) zwischen dem Generator und dem E-Banking-System

- *Stufe 2:* Auswahl und Verfeinerung alternativer Lösungsansätze zur Kapselung des Algorithmus der TAN-Generierung

  Zwischenziel: Lose Kopplung

- *Stufe 3:* Auswahl und Verfeinerung alternativer Lösungsansätze zur Codewort-Generierung und Gateway-Integration sowie endgültige Entwicklung des Endgerätes

  Zwischenziele: Lose Kopplung, unveränderte Durchführungszeiten je Transaktion (Transaktionszeiten)

Stufe 1 – Lösungsansätze für TAN-Generierung und Übertragungsmedium

Im Anschluss an die Entwicklung des Grobplans erfolgt die Aufstellung kombinierter Lösungsansätze für jede der drei Entscheidungsstufen. Der Grobplan wird dabei verfeinert und es werden für jede Stufe geeignete Lösungsansätze ausgewählt, verfeinert und konkretisiert. Da die skizzierten Lösungsszenarien zwei grundsätzlich unterschiedliche Verfahren zur Generierung der TANs beschreiben, ist zunächst eine Verfahrensentscheidung an den Anfang des Entscheidungsbaumes zu stellen (siehe Abb. 6.25). Somit können beide Lösungsszenarien in einem Entscheidungsbaum abgebildet werden.

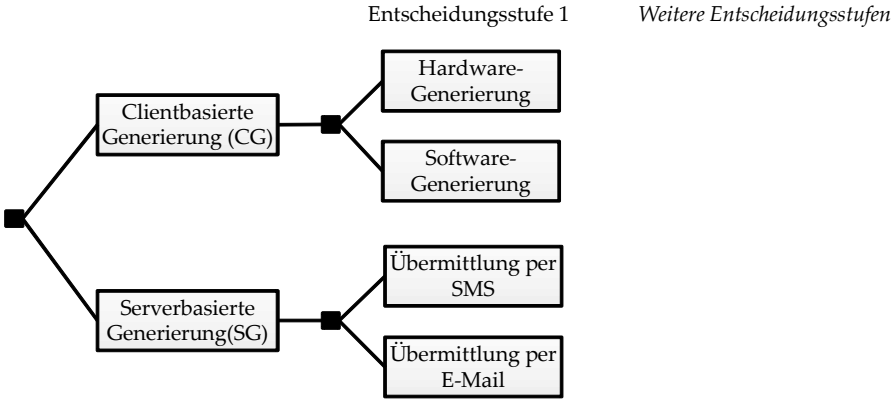


Abb. 6.25: Erste Stufe der Architekturentscheidung

Der im vorangegangenen Schritt aufgestellte Grobplan zeigt, dass in der ersten Stufe ein Prototyp des Endgerätes zu entwickeln ist. Ein solcher Prototyp ist ein nahezu vollwertiges Software- oder Hardwareprodukt, welcher jedoch nur die grundlegenden Funktionen in minimalem Umfang umfasst, um die nachfolgenden Entwicklungsschritte, u. a. die Integration des gekapselten Algorithmus zur TAN-Generierung, konzipieren und testen zu können.

Da die Schritte zur Auswahl, Verfeinerung und Konkretisierung der alternativen Lösungsansätze für jede Stufe identisch zur vorangegangenen Typo3-Architekturentscheidung sind, werden die Lösungsansätze in den folgenden Abschnitten nur in Kurzform beschrieben. Detaillierte Ausführungen zu den jeweiligen Lösungsansätzen, Maßnahmenfolgen und Eigenschaften sind im Anhang aufgeführt (siehe Abschnitt 10.5.1 S. 179ff.).

Im Falle der 'clientbasierten Generierung' stehen die folgenden beiden Lösungsansätze zur Verfügung:

- *Hardware-Generator (Details S. 179):* Ein Hardware-Generator ist eine Hardwarekomponente, z. B. ein Security-Token, welche den Algorithmus zur Generierung einer TAN beinhaltet (siehe für ein Beispiel z.B. [Reim07] und [Bwba08]). Um die Generierung zu starten, ist zusätzlich ein Codewort einzugeben, z. B. eine Ziffernfolge, welches dem Bankkunden über das E-Banking-Portal mitgeteilt wird. Als Prototyp wird dazu eine Java-Anwendung für ein Mobiltelefon entwickelt, mit der Test-TANs generiert werden können. Test-TANs sind gültige TANs, mit denen zwar keine Aufträge authentifiziert, aber die notwendigen Tests durchgeführt werden können.

  Zielerreichung: Da die einzige Verbindung zwischen dem Hardware-Generator und den restlichen Teilen des E-Banking-Systems die manuelle Übergabe des Codewortes und der generierten TANs (beim Prototyp die Test-TANs) ist, kann von einer ausreichenden 'losen Kopplung' ausgegangen werden (Zwischenzielerreichung 100 %). Der Lösungsansatz ist zudem sehr 'sicher', da zur Generierung einer TAN ein gültiges Codewort eingegeben werden muss (Zwischenzielerreichung 100 %). Der 'Implementierungsaufwand' dieses Lösungsansatzes wird auf 10 Personentage geschätzt. Die Grundlage für die Aufwandsschätzung sind Vergleichsdaten ähnlicher Projekte bzw. Architektur- sowie Softwaresystemveränderungen.

- *Software-Generator (Details S. 180):* Dies ist eine eigenständige und betriebssystemunabhängige Software-Anwendung, die auf dem Rechner des Clients installiert und ausgeführt wird. Analog zum Hardware-Generator beinhaltet die Software den Algorithmus zur TAN-Generierung. Um die Generierung zu starten, muss der Bankkunde das ihm über das E-Banking-Portal mitgeteile Codewort eingeben.

  Zielerreichung: Da analog zum 'Hardware-Generator' die einzige Verbindung zwischen der Software und dem restlichen E-Banking-System die manuelle Übergabe des Codewortes und der generierten TANs ist, kann auch hierbei von einer ausreichenden 'losen Kopplung' ausgegangen werden (Zwischenzielerreichung 100 %). Im Gegensatz zum 'Hardware-Generator' kann die Generierungs-Software durch Kopieren vervielfältigt werden und stellt somit ein potenzielles Risiko für die 'Sicherheit' dar (Zwischenzielerreichung 70 %). Der 'Implementierungsaufwand' dieses Lösungsansatzes wird auf 10 Personentage geschätzt.

Bei der 'serverbasierten Generierung' stehen die folgenden beiden Lösungsansätze zur Verfügung:

- *Übermittlung per SMS (Details S. 180):* Bei diesem Lösungsansatz ist zunächst eine Anwendung für den Applikationsserver zu entwickeln, mit der Test-TANs generiert werden können. Die generierten TANs werden über ein Gateway zu einem Mobilfunkbetreiber übermittelt, welcher die TAN per Kurzmitteilung (SMS) an das Mobiltelefon des betreffenden Bankkunden schickt. Zur prototypischen Implementierung des Gateways kann ein einzelnes Mobiltelefon an das E-Banking-System angebunden werden.

  Zielerreichung: Aufgrund der Vielzahl an Vermittlungsinstanzen ist bei diesem Lösungsansatz eine 'stärkere Kopplung' mit dem E-Banking-System zu erwarten (Zwischenzielerreichung 80%). Die 'Sicherheit' des Verfahrens ist jedoch hoch, da das Mobilfunkgerät des Bankkunden durch einen PIN abgesichert ist und das gesamte Verfahren zur Generierung der TANs auf der Seite des E- Banking-Systems erfolgt (Zwischenzielerreichung 100%). Der 'Implementierungsaufwand' wird auf 5 Personentage geschätzt.

- *Übermittlung an eine E-Mail-Adresse (Details S. 181):* Beim zweiten Lösungsansatz der 'serverseitigen TAN-Generierung' wird die generierte TAN über einen Mail-Server an die E-MailAdresse des betreffenden Bankkunden geschickt. Die E-Mail ist dabei verschlüsselt und nur der betreffende Bankkunde verfügt über den notwendigen Schüssel, um den Inhalt der E-Mail im Klartext lesen zu können. Für eine prototypische Implementierung ist wie bei der 'Übermittlung per SMS' zunächst eine Anwendung für den Applikationsserver zu entwickeln, mit der Test-TANs generiert werden können. Die Anwendung muss zudem die Verschlüsselung und den Versand der E-Mail durchführen.

  Zielerreichung: Da die generierten TANs zunächst in einer verschlüsselten E-Mail und an einen Mailserver übertragen und von dort aus versendet werden, entsteht eine geringfügig 'stärkere Kopplung' mit dem E-Banking-System (Zwischenzielerreichung 90 %). Da die E-Mail während der Übermittlung leicht abgefangen werden kann, existiert bei diesem Verfahren eine potenzielle Lücke für die 'Sicherheit' des Verfahrens (Zwischenzielerreichung 90 %). Der 'Implementierungsaufwand' wird auf 10 Personentage geschätzt, da die Verschlüsselung der E-Mails einen zusätzlichen Aufwand erfordert.

Stufe 2 – Kapselung des Algorithmus zur TAN-Generierung

In der zweiten Stufe der Entscheidung sind geeignete Lösungsansätze auszuwählen, um den Algorithmus zur Generierung der TANs einerseits zu kapseln sowie den Algorithmus bei der 'clientbasierten Generierung' wiederverwenden zu können. In Abb. 6.26 ist dargestellt, welche Lösungsansätze für die Kapselung zur Verfügung stehen. Da der Algorithmus zur Generierung der TANs jedoch im Transaktionssystem der Bank hinterlegt ist, ist aus Sicherheitsgründen kein direkter Zugriff im Rahmen der Architekturveränderung möglich. Der Aufwand zur Kapselung des Algorithmus ist daher Aufgabe der jeweiligen Bank und wird somit in der Entscheidungsfindung nicht berücksichtigt.

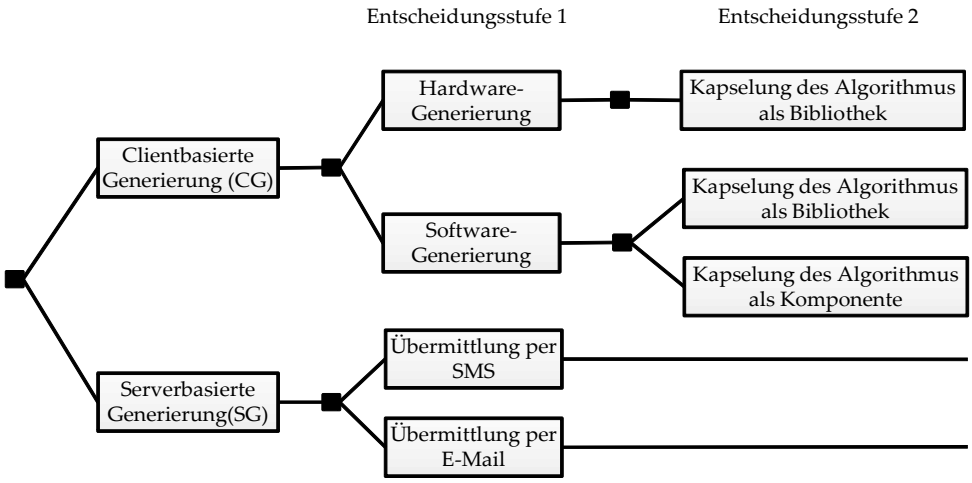


Abb. 6.26: Zweite Stufe der Architekturentscheidung

Im Falle der 'clientbasierten Generierung' sind nach dem in Abschnitt 7 aufgeführten Katalog (siehe S. 138ff.) die folgenden Lösungsansätze für die Algorithmus-Kapselung in Betracht zu ziehen; dabei sind die Abhängigkeiten zu den Endgeräte-Alternativen Hardware- und Software-Generierung zu beachten:

- *Bibliothek (Details S. 182):* Entsprechend der Ausführungen in Abschnitt 7.4.3 (S. 155) ist eine Bibliothek ein Container, um Funktionalität in einer bestimmten Programmiersprache wiederverwendbar abzulegen. Hierbei wird der Algorithmus zur Generierung der TANs, auf der Grundlage benutzerspezifischer Parameter und des Codewortes, in einer Bibliothek implementiert. Wird die Bibliothek im Zusammenhang mit der Hardware-Generierung eingesetzt, sind eingeschränkte Hardwareressourcen zu beachten, insbesondere Speicherressourcen.

  Zielerreichung: Sofern das Schnittstellendesign der Bibliothek einen direkten Zugriff auf alle relevanten Funktionen ermöglicht, ist von einer ausreichenden 'losen Kopplung' auszugehen. Jedoch ist der Zugriff auf eine Bibliothek grundsätzlich nur in derselben oder einer kompatiblen Programmiersprache möglich, in der auch die Funktionen in der Bibliothek implementiert sind (Zielerreichung 90 %). Die Abschätzung des 'Implementierungsaufwandes' ist nicht erforderlich, da die Durchführung der Algorithmus-Kapselung der jeweiligen Bank überlassen ist.

- *Komponente (Details S. 182):* Als Alternative zu einer Bibliothek kann der Algorithmus auch in Form einer Komponente gekapselt werden. Dies hat den Vorteil, dass der Zugriff auf die Komponente nicht von der Programmiersprache abhängig ist, in der die Funktionalität der Komponente implementiert ist (siehe Abschnitt 7.4.3 auf S. 155). Da für den Bereich der eingebetteten Systeme kein geeignetes Komponentenmodell zur Verfügung steht, kann dieser Lösungsansatz nur im Zusammenhang mit der Software-Generierung verwendet werden.

  Zielerreichung: Sofern der Zugriff auf den Algorithmus zur TAN-Generierung ausschließlich über eine Schnittstelle der Komponente erfolgt, ist von einer ausreichenden 'lose Kopplung' auszugehen (Zielerreichung 100 %). Eine Abschätzung des 'Implementierungsaufwandes' wird nicht durchgeführt, da die Kapselung der jeweiligen Bank überlassen ist.

Bei der 'serverbasierten Generierung' kann auf die Auswahl und Verfeinerung von alternativen Lösungsansätzen verzichtet werden. In diesem Fall ist durch die Bank zu gewährleisten, dass das Transaktionssystem um eine geeignete Schnittstelle

erweitert wird, die den Zugriff auf den Algorithmus zur Generierung der TANs gewährleistet. Sofern die Schnittstelle einen direkten Zugriff auf die relevanten Funktionen ermöglicht, ist von einer ausreichend losen Kopplung auszugehen.

Stufe 3 – Lösungsansätze zur Codewort-Generierung und Gateway-Integration

In der letzten Stufe der Architekturveränderung erfolgt die Planung der Implementierung des neuen TAN-Verfahrens. Dabei sind die Endgeräte und der gekapselte Algorithmus in das E-Banking-System *Multicash@online* zu integrieren. Außerdem ist der Applikationsserver entsprechend anzupassen. Im Rahmen der 'clientseitigen-Generierung' sind abschließend noch die alternativen Lösungsansätze zur Code-Wortgenerierung auszuwählen. Zudem sind für die 'serverseitige Generierung' die Lösungsansätze für ein Mobilfunk- oder für ein E-Mail-Gateway auszuwählen. Abb. 6.27 gibt einen Überblick über die Lösungsansätze der dritten Stufe und die möglichen Kombinationen von Lösungsansätzen. Jedes Dreieck (am rechten Rand) steht für eine mögliche Kombination an Lösungsansätzen. Jede Kombination erfüllt die Ziele jedoch in einem anderen Umfang und benötigt dafür einen anderen Implementierungsaufwand. Da die Kombinationen an Lösungsansätzen aufgrund der Vielzahl verschiedener Eigenschaften nur bedingt miteinander vergleichbar sind, ist eine abschließende Bewertung erforderlich.

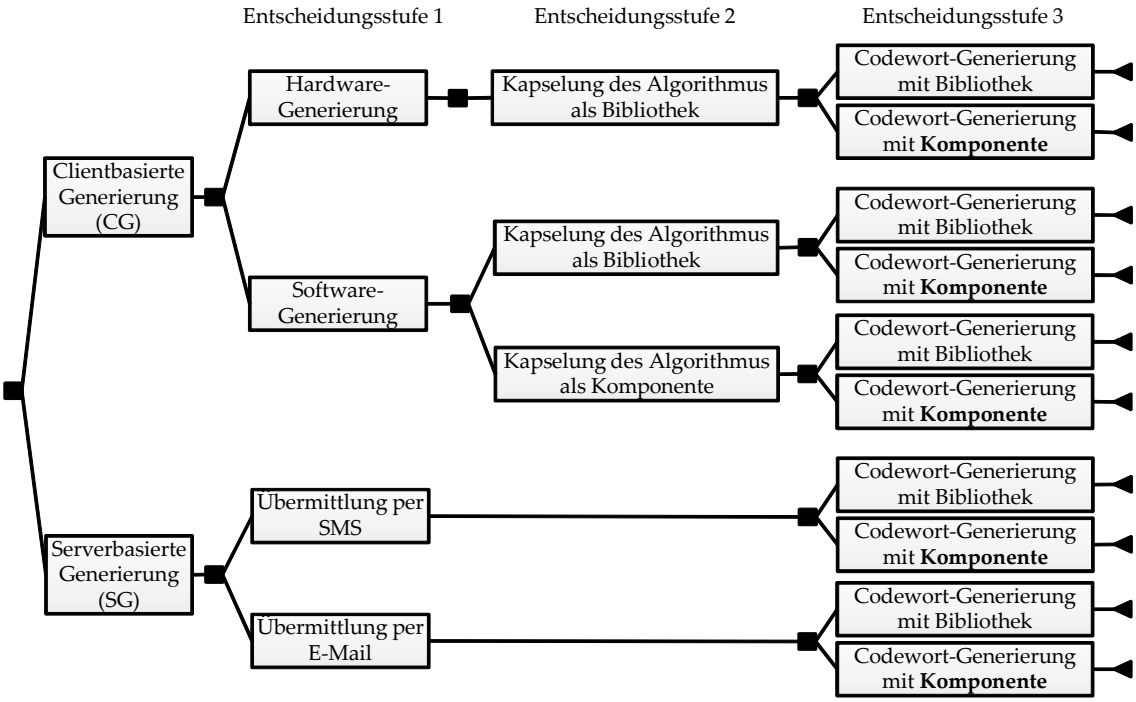


Abb. 6.27: Dritte und letzte Stufe der Architekturentscheidung

Im Falle der 'clientbasierten Generierung' sind – neben der abschließenden Entwicklung des Endgerätes – jene Lösungsansätze auszuwählen, mit denen die Generierung des Codewortes realisiert werden kann. Ein solches Codewort wird dann erzeugt, wenn der Bankkunde über das E-Banking-Portal den Auftrag eingegeben hat, u. a. Kontonummer und Bankleitzahl sowie Betrag bei einer Überweisung. Der Applikationsserver kann mittels Bibliotheken oder Komponenten um diese Funktionalität erweitert werden:

- *Bibliothek (Details S. 184):* Bei diesem Lösungsansatz wird die Funktionalität zur Generierung eines Codewortes als zusätzliche Funktionsbibliothek implementiert (siehe für die bereits existierenden Funktionsbibliotheken Abb. 6.21, S. 123). Im Falle der Hardware-Generierung ist zu berücksichtigen, dass die Generierung der TAN in einem Endgerät erfolgt. Das Codewort ist auf eine Ziffernfolge beschränkt, da nur eingeschränkte Eingabemöglichkeiten in Form mehrfachbelegter Tasten von 0-9 zur Verfügung stehen.

Zielerreichung: Mit der Implementierung der relevanten Funktionen zur Generierung des Codewortes in Form einer Bibliothek kann eine 'lose Kopplung' zwischen dem Applikationsserver und der Codewort-Generierung erreicht werden (Zielerreichung 90 %). Grundsätzlich werden dabei die 'Durchführungszeiten je Transaktion' erhöht, da zunächst das Codewort generiert und durch den Bankkunden in den Software- oder Hardware-Generator eingegeben werden muss, bevor die TANs generiert werden können (Zielerreichung aufgrund der Veränderung an den Transaktionszeiten 20 %). Der 'Implementierungsaufwand' wird im Falle der Hardware-Generierung auf 110 Personentage geschätzt, da neben der Entwicklung der Bibliothek auch das relevante Endgerät zu konzipieren und zu entwickeln ist; für die Entwicklung des Software-Generators wird der Aufwand auf 80 Personentage geschätzt.

- *Komponente (Details S. 184):* Ein anderer Lösungsansatz ist die Entwicklung einer Komponente, die die relevanten Funktionen zur Codewort-Generierung umfasst (in Abb. 6.27 auf S. 131 durch Fettdruck hervorgehoben). Der Vorteil dieses Lösungsansatzes liegt darin, dass für die Entwicklung der Komponente auch eine andere und eventuell effizientere Programmiersprache genutzt werden kann, als jene, in der der Applikationsserver und die bereits existierenden Funktionsbibliotheken implementiert sind. Auch bei diesem Lösungsansatz muss im Falle der Hardware-Generierung beachtet werden, dass nur eingeschränkte Eingabemöglichkeiten (mehrfachbelegte Tasten von 0-9) zur Verfügung stehen

  Zielerreichung: Auch bei diesem Lösungsansatz kann eine ausreichende 'lose Kopplung' zwischen dem Applikationsserver und der Funktionalität zur Generierung des Codewortes erreicht werden; zudem besteht die Möglichkeit, dass eine effiziente Programmiersprache nahezu frei ausgewählt werden kann (Zielerreichung 100 %). Analog zum Lösungsansatz Bibliothek ist von erhöhten 'Transaktionszeiten' auszugehen (Ziererreichung 20 %). Der 'Implementierungsaufwand' wird im Falle der Hardware-Generierung auf 100 Personentage geschätzt, da neben der Entwicklung der Bibliothek auch die relevante Hardware-Komponente zu konzipieren und zu entwickeln ist. Für die Entwicklung des Software-Generators wird der Aufwand auf 70 Personentage geschätzt.

Neben der 'clientbasierten Generierung' kann die Generierung der TANs auch serverseitig erfolgen. Die Übertragung der TANs an den Bankkunden kann entweder über ein Mobilfunk-Gateway oder in verschlüsselter Form per E-Mail erfolgen. Als Gateway können entsprechend der Ausführungen in Abschnitt 7 (S. 138ff.) ein Broker oder ein Vermittler eingesetzt werden:

- *Broker (Details S. 185):* Die Verbindung zwischen Applikationsserver und dem Mobilfunk-Gateway oder dem Mailserver kann über einen Broker erfolgen. Der Broker agiert hierbei als Makler und überträgt die Nachrichten vom Applikationsserver aus kommend an die relevanten Stellen. Der Fokus des Brokers liegt dabei auf einem ausgewogenen Lastenausgleich. Ist beispielsweise ein Mailserver überlastet und kann daher die E-Mail nicht direkt versendet werden, wählt der Broker automatisch einen Mailserver mit noch freien Kapazitäten.

  Zielerreichung: Da der Broker in der Praxis meist vom Applikationsserver getrennt implementiert werden muss, wird die 'lose Kopplung' nur zum Teil erfüllt (Zielerreichung 50 %). Die generierten TANs werden jedoch nur mit geringem Zeitverzug an den Bankkunden übermittelt; daher kann von einer identischen 'Durchführungszeit der Transaktionen' ausgegangen werden (Zielerreichung 100 %). Der 'Implementierungsaufwand' wird im Falle der Nutzung des Mobilfunk-Gateways auf 56 Personentage geschätzt, da keine weiteren Endgeräte entwickelt werden müssen. Im Falle der Übermittlung per E-Mail wird der Implementierungsaufwand auf 66 Personentage geschätzt, da außerdem ein Verschlüsselungsverfahren zu implementieren ist.

- *Vermittler (Details S. 185):* Die Verbindung zwischen Applikationsserver und dem Mobilfunk-Gateway oder dem Mailserver kann auch über einen Vermittler erfolgen (in Abb. 6.27 auf S. 106 durch Fettdruck hervorgehoben). Der Vermittler kann neben der vom Broker bekannten Funktion des Maklers zusätzliche Funktionalität beinhalten. Daher werden bei diesem Lösungsansatz die Erzeugung der Kurznachricht bzw. die Erzeugung und Verschlüsselung der E-Mail komplett durch den Vermittler realisiert. Die erzeugte Kurznachricht oder E-Mail wird im Anschluss an ein freies Mobilfunk-Gateway oder einen noch freien Mailserver gesendet.

  Zielerreichung: Da die Funktionalität zur Übertragung der TANs, entweder per E-Mail oder per Kurznachricht, vollständig im Vermittler implementiert ist, wird die erforderliche 'lose Kopplung' zwischen dem Applikationsserver und dem Vermittler erreicht (Zielerreichung 100 %). Wie bei der vorangegangenen Alternative kann von einer identischen 'Durchführungszeit der Transaktionen' ausgegangen werden (Zielerreichung 100 %). Der 'Implementierungsaufwand' wird im Falle der Nutzung des Mobilfunk-

Gateways auf 76 Personentage geschätzt. Im Falle der Übermittlung per E-Mail wird der Implementierungsaufwand auf 86 Personentage geschätzt, da zusätzlich ein Verschlüsselungsverfahren zu implementieren ist.

6.2.4 Phase 4 – Bewertung und Entscheidung

In der letzten Phase der Entscheidungsfindung werden die aufgestellten Kombinationen von Lösungsansätzen anhand ihrer Zielerreichung systematisch verglichen und bewertet (siehe zu den Phasen des Entscheidungsprozesses Abb. 3.1 auf S. 44). Das ist erforderlich, da die einzelnen Ziele 'Sicherheit', 'Wartbarkeit' und 'Transaktionszeiten' in ganz unterschiedlicher Art und Weise erfüllt werden und ein direkter Vergleich der verschiedenen Lösungsansätze aufgrund der Komplexität mit großen Unsicherheiten verbunden ist (siehe für die Kriterien für diese Tätigkeit Tab. 10.2, S. 171ff.). So wird bei der 'serverbasierten Generierung', bei der die Übermittlung der TAN per Kurznachricht (SMS) erfolgt und ein Broker eingesetzt wird, die Sicherheit zu 100 % bei einem geringen Implementierungsaufwand in Höhe von 61 (56+5) Personentagen erfüllt (siehe 'Kombination 7'). Erfolgt die Übertragung dagegen per E-Mail, wird die Sicherheit nur zu 90 % bei einem Aufwand von 76 (66+10) Personentagen erreicht (siehe 'Kombination 9'). Zudem wird das Ziel der 'Verbesserung der Wartbarkeit' bei jeder Kombination von Lösungsansätzen in unterschiedlichem Umfang erfüllt.

Alle Eigenschaftswerte der verschiedenen Kombinationen von Lösungsansätzen sind in einer Übersicht in Tab. 6.10 aufgeführt. Die Kurzform der Kombinationen ergibt sich aus den Anfangsbuchstaben der Lösungsansätze (je Ast) des in Abb. 6.27 (siehe S. 131) dargestellten Entscheidungsbaumes.[6]

Kombinationen		***Stufe 1***			***Stufe 2***	***Stufe 3***		
		LK (%)	*Si (%)*	*AW (PT)*	*LK (%)*	*LK (%)*	*Tz (%)*	*AW (PT)*
1	*CG, HWG, KB, CGB*	100	100	10	90	90	20	110
2	*CG, HWG, KB, CGK*	100	100	10	100	100	20	100
3	*CG, SWG, KB, CGB*	100	70	10	90	90	20	80
4	*CG, SWG, KB, CGK*	100	70	10	100	100	20	70
5	*CG, SWG, KK, CGB*	100	70	10	90	90	20	80
6	*CG, SWG, KK, CGK*	100	70	10	100	100	20	70
7	*SG, SMS, GB*	80	100	5	100	50	100	56
8	*SG, SMS, GV*	80	100	5	100	100	100	76
9	*SG, EM, GB*	90	90	10	100	50	100	66
10	*SG, EM, GV*	90	90	10	100	100	100	86

Tab. 6.10: Übersicht über Zielerreichung und Implementierungsaufwand der Kombinationen von Lösungsansätzen

Normalisierung der Eigenschaftswerte

Der erste Schritt zur Bewertung der unterschiedlichen Kombinationen von Lösungsansätzen ist die Darstellung der Eigenschaftswerte in einer aggregierten Form. Das ist notwendig, da für die Wartbarkeit und den Implementierungsaufwand bei jeder Kombination zwei Werte existieren (von der ersten und der dritten Stufe). Für die Zielerreichungswerte der Wartbarkeit und des Aufwands wird je Kombination der Durchschnitt gebildet. Alle Eigenschaftswerte in aggregierter Form sind in Tab. 6.11 in einer Übersicht zu finden (siehe folgende Seite)

6 LK – Lose Kopplung/Wartbarkeit, Si – Sicherheit, AW – Implementierungsaufwand, Tz -Transaktionszeiten, CG – Clientbasierte Generierung, HWG – Hardware-Generator, SWG – Software-Generator, KB – Kapselung als Bibliothek, KK – Kapselung als Komponente, CGB – Codewort Generierung mittels Bibliothek, CGK – Codewort Generierung mittels Komponente,GB – Gateway Broker, GV – Gateway Vermittler

Kombinationen/ Zielerreichung		*Sicherheit (%)*	*Wartbarkeit (%)*	*Transaktionszeiten (%)*	*Aufwand (PT)*
1	*CG, HWG, KB, CGB*	100	93	20	120
2	*CG, HWG, KB, CGK*	100	100	20	110
3	*CG, SWG, KB, CGB*	70	93	20	90
4	*CG, SWG, KB, CGK*	70	100	20	80
5	*CG, SWG, KK, CGB*	70	93	20	90
6	*CG, SWG, KK, CGK*	70	100	20	80
7	*SG, SMS, GB*	100	77	100	61
8	*SG, SMS, GV*	100	93	100	81
9	*SG, EM, GB*	90	80	100	76
10	*SG, EM, GV*	90	97	100	96

Tab. 6.11: Übersicht über die Eigenschaftswerte der Kombinationen von Lösungsansätzen in aggregierter Form

Im zweiten Teil des ersten Schrittes erfolgt die Normalisierung der Eigenschaftswerte. Dabei ist zunächst zu bestimmen, welche Wertfunktion für die Normalisierung einzusetzen ist. Die Ermittlung der Normalisierungsfunktionen kann durch Befragung des Entscheidungsträgers oder anhand spezieller Methoden durchgeführt werden, z. B. die 'Direct-Ratio-Methode' [Fish67] (siehe Abschnitt 2.9, S. 38). Im Rahmen der Architekturveränderung des E-Banking-Systems werden die Eigenschaftswerte wie folgt normalisiert:

- *Normalisierungsfunktion für 'Sicherheit' und 'Wartbarkeit':* Aufgrund der hohen Bedeutung der 'Sicherheit' und der 'Wartbarkeit' bevorzugt der Entscheidungsträger ausschließlich eine hohe Zielerreichung. Die Werte sind daher mit einer progressiven Funktion zu normalisieren. Konkret bedeutet dies, dass ausschließlich ein Wert ab 80-90 % vom Entscheidungsträger bevorzugt wird und dementsprechend mit 0.8 oder 0.9 zu normalisieren ist. Werden die 'Wartbarkeit' oder die 'Sicherheit' mit einem Wert unter 80 % erfüllt, werden diese Werte mit einem Wert nahe null normalisiert. Die progressive Normalisierungsfunktion ist in Abb. 6.28 dargestellt.
- *Normalisierungsfunktion für 'Transaktionszeiten' und 'Implementierungsaufwand':* Diese beiden Ziele haben für den Entscheidungsträger eine untergeordnete Bedeutung. Die 'Verbesserungen bei der Sicherheit' und die 'Beibehaltung der Wartbarkeit' sind für den Entscheidungsträger wichtiger. Da die Werte zwischen den jeweils besten oder schlechtesten Ausprägungen bei den 'Transaktionszeiten' und dem 'Aufwand' vom Entscheidungsträger gleich stark (in positiver wie negativer Sicht) bevorzugt werden, sind sie mit einer linearen Funktion zu normalisieren (siehe Abb. 6.28).

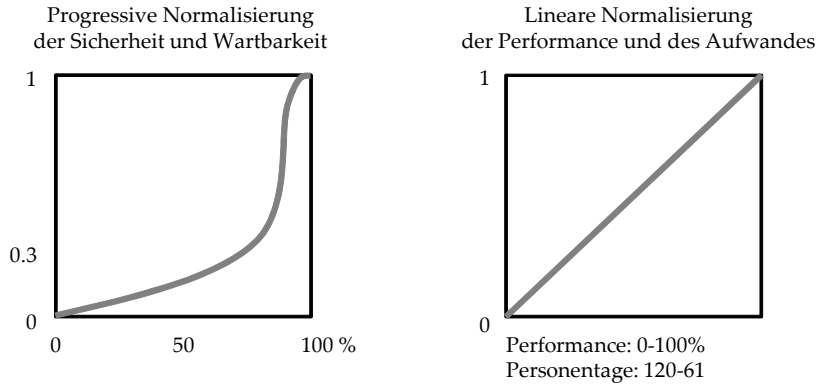


Abb. 6.28: Progressive und lineare Normalisierungsfunktionen

Die entsprechend der Normalisierungsfunktionen umgeformten Werte sind in Tab. 6.12 aufgezeigt. Die normalisierten Werte sind durch Fettdruck hervorgehoben, die Ausgangswerte aus Tab. 6.11, siehe S. 134, sind vor dem Semikolon kursiv dargestellt.

Kombinationen		*Sicherheit (%)*	*Wartbarkeit (%)*	*Transaktionszeiten (%)*	*Aufwand (PT)*
1	*CG, HWG, KB, CGB*	*100*; **1.0**	*93*; **0.9**	*20*; **0.2**	*120*; **0.0**
2	*CG, HWG, KB, CGK*	*100*; **1.0**	*100*; **1.0**	*20*; **0.2**	*110*; **0.2**
3	*CG, SWG, KB, CGB*	*70*; **0.3**	*93*; **0.9**	*20*; **0.2**	*90*; **0.5**
4	*CG, SWG, KB, CGK*	*70*; **0.3**	*100*; **1.0**	*20*; **0.2**	*80*; **0.7**
5	*CG, SWG, KK, CGB*	*70*; **0.3**	*93*; **0.9**	*20*; **0.2**	*90*; **0.5**
6	*CG, SWG, KK, CGK*	*70*; **0.3**	*100*; **1.0**	*20*; **0.2**	*80*; **0.7**
7	*SG, SMS, GB*	*100*; **1.0**	*77*; **0.4**	*100*; **1.0**	*61*; **1.0**
8	*SG, SMS, GV*	*100*; **1.0**	*93*; **0.9**	*100*; **1.0**	*81*; **0.7**
9	*SG, EM, GB*	*90*; **0.9**	*80*; **0.4**	*100*; **1.0**	*76*; **0.7**
10	*SG, EM, GV*	*90*; **0.9**	*97*; **0.9**	*100*; **1.0**	*96*; **0.4**

Tab. 6.12: Übersicht über die normalisierten Eigenschaftswerte der Kombinationen von Lösungsansätzen

Gewichtung nach Relevanz

Im zweiten Schritt der Bewertung der Kombinationen von Lösungsansätzen erfolgt die Gewichtung der Ziele. So haben die 'Sicherheitsverbesserungen' und die 'Beibehaltung der Wartbarkeit' eine höhere Gewichtung als der 'Implementierungsaufwand' und die 'unveränderten Transaktionszeiten'. Da sich diese unterschiedliche Gewichtung der Ziele in der Bewertung der Lösungsansätze niederschlagen muss, sind die Eigenschaftswerte mit einem Gewichtungsfaktor zu multiplizieren. Die Gewichtungsfaktoren je Ziel ergeben sich einerseits durch das in Abb. 6.20 dargestellte Ziel-Wirkungsmodell (siehe S. 121) und durch Befragung des Entscheidungsträgers. Die ermittelten Gewichtungsfaktoren sind in Tab. 6.13 zusammengefasst.

Ziele der betrachteten Entscheidung	**Gewichtungsfaktor**
Verbesserung der Sicherheit	0.35
Beibehaltung der Wartbarkeit	0.35
Unveränderte Transaktionszeiten	0.15
Minimaler Implementierungsaufwand	0.15
Summe	**1.00**

Tab. 6.13: Gewichtungsfaktoren der Ziele

Die gewichteten Eigenschaftswerte sind in Tab. 6.14 aufgeführt und mit Fettdruck hervorgehoben; die Ausgangswerte aus Tab. 6.12 (siehe S. 135) sind kursiv dargestellt.

Kombinationen		*Sicherheit*	*Wartbarkeit*	*Transaktionszeiten*	*Aufwand*	*Summe*
1	*CG, HWG, KB, CGB*	*1.0*; **0.35**	*0.9*; **0.32**	*0.2*; **0.03**	*0.0*; **0.00**	**0.70**
2	*CG, HWG, KB, CGK*	*1.0*; **0.35**	*1.0*; **0.35**	*0.2*; **0.03**	*0.2*; **0.03**	**0.76**
3	*CG, SWG, KB, CGB*	*0.3*; **0.11**	*0.9*; **0.32**	*0.2*; **0.03**	*0.5*; **0.08**	**0.53**
4	*CG, SWG, KB, CGK*	*0.3*; **0.11**	*1.0*; **0.35**	*0.2*; **0.03**	*0.7*; **0.11**	**0.60**
5	*CG, SWG, KK, CGB*	*0.3*; **0.11**	*0.9*; **0.32**	*0.2*; **0.03**	*0.5*; **0.08**	**0.53**
6	*CG, SWG, KK, CGK*	*0.3*; **0.11**	*1.0*; **0.35**	*0.2*; **0.03**	*0.7*; **0.11**	**0.59**
7	*SG, SMS, GB*	*1.0*; **0.35**	*0.4*; **0.14**	*1.0*; **0.15**	*1.0*; **0.15**	**0.79**
8	*SG, SMS, GV*	*1.0*; **0.35**	*0.9*; **0.32**	*1.0*; **0.15**	*0.7*; **0.11**	**0.92 (max)**
9	*SG, EM, GB*	*1.0*; **0.35**	*0.4*; **0.14**	*1.0*; **0.15**	*0.7*; **0.11**	**0.71**
10	*SG, EM, GV*	*1.0*; **0.35**	*0.9*; **0.32**	*1.0*; **0.15**	*0.4*; **0.06**	**0.84**

Tab. 6.14: Übersicht über die normalisierten Eigenschaftswerte der Kombinationen von Lösungsansätzen

Im Anschluss an Gewichtung der Werte kann der Wert je Kombination von Lösungsansätzen errechnet werden. Der dritte Schritt, die Berücksichtigung wahrscheinlicher und unwahrscheinlicher Varianten, kann im Rahmen dieser Architekturentscheidung entfallen, da keinerlei Wahrscheinlichkeiten zu berücksichtigen sind. Die aufsummierten Werte je Kombination sind in der letzten Spalte in Tab. 6.14 dargestellt.

Entscheidung über die betrachteten Lösungsansätze

Entsprechend der in Tab. 6.14 aufgelisteten Werte je Kombinationen von Lösungsansätzen sollte sich der Entscheidungsträger für die 'Kombination 8' entscheiden, da diese mit 0.92 den höchsten Wert besitzt. Nicht den höchsten, aber auch einen hohen Wert mit deutlich über 0.70 besitzen die 'Kombinationen 2, 7 und 10'. Die schlechtesten Kombinationen von Lösungsansätzen sind die 'Kombinationen 3 und 5'. Nach rationalen Gesichtspunkten sollte sich der Entscheidungsträger für 'Kombination 8' entscheiden.

6.2.5 Konsequenzen vergleichbarer Architekturveränderungen

Ein auf Kurznachrichten (SMS) basierendes Verfahren zur Übermittlung der TANs und zur Authentifizierung der Transaktionen der Bankkunden ist derzeit bei verschiedenen Kreditinstituten, u. a. bei der Postbank, unter dem Namen Mobile TAN (mTAN) oder SMSTAN implementiert [Reim07] (und [Ste+02]). Wie im Rahmen des skizzierten Lösungsszenarios 'serverbasierte TAN-Generierung' (siehe Abb. 6.23, S. 125) beschrieben werden dem Bankkunden generierte TANs per Kurznachricht zugesandt. Die Erfahrungen aus dem praktischen Einsatz des Verfahrens bei der Postbank zeigen, dass dieses Verfahren mit erheblichen Kosten für den Versand der Kurznachrichten verbunden ist. Dies spiegelt sich auch im Rahmen der Entscheidungsfindung wider, bei der dem Lösungsansatz 'Übertragung per verschlüsselter E-Mail' ein höherer Wert zugewiesen wird. Die hohe Sicherheit dieses Verfahrens überprüfte z. B. die TÜV Rheinland Group [Reim07].

Auch die 'clientbasierte TAN-Generierung', das zweite Lösungsszenario (siehe Abb. 6.24, S. 126), wird in der Praxis eingesetzt; dabei sind jedoch zwei Varianten zu unterscheiden. Bei der ersten Variante werden bestimmte Merkmale der EC-Karte zur TAN-Generierung genutzt, wie Kontonummer oder Ablaufdatum der Karte. Steckt der Bankkunde seine EC-Karte in ein spezielles Lesegerät, wird eine entsprechende TAN generiert. Diese Variante wird unter dem Namen Sm@rt-TAN-Generator vermarktet (siehe Abb. 6.29). Da bei dieser Variante ein Codewort fehlt, sind hierbei die auf S. 118 beschriebenen Sicherheitsrisiken zu bedenken. Bei der zweiten Variante, die bei der BW-Bank unter dem Namen eTAN eingesetzt wird, gibt der Bankkunde ein Codewort auf dem Gerät ein, welches ihm eine TAN generiert [Bwba08]. Da aus Kostengründen die Bankkunden in beiden Varianten die Geräte zur TAN-Generierung nur gegen ein Entgelt erhalten und sich die eingesetzten Geräte zudem als fehleranfällig erweisen (siehe die Fehlermeldung im Display des Sm@rt-TAN-Generators in Abb. 6.29), ist die serverbasierte Form der TAN-Generierung derzeit vorzuziehen. Diese negativen Konsequenzen hätten durch die Anwendung des Entscheidungsprozesses erkannt und eventuell vermieden werden können.

Abb. 6.29: Sm@rt-TAN-Generator

Kapitel 7

Katalog mit Lösungsansätzen und deren Qualitätseigenschaften

Um Risiken und Unsicherheiten bei der Auswahl von Lösungsansätzen zu reduzieren, erfolgt in den nachstehenden Abschnitten eine aggregierte Übersicht über bekannte sowie häufig verwendete Lösungsansätze und deren Qualitätseigenschaften. Gerade bei der Verbesserung der Architekturqualität existieren oftmals große Unsicherheiten, mit welchen Lösungsansätzen die identifizierten Schwachstellen und Problembereiche der Architektur restrukturiert werden können. Während funktionale Ziele durch Verfeinerung und Zerlegung analysiert und mit entsprechenden Lösungsansätzen hinterlegt werden können, sind nicht-funktionale Qualitätsziele deutlich komplexer und widersprüchlicher bei der Analyse und Verfeinerung. Aufgrund der Widersprüche ist häufig nicht klar erkennbar, welche Lösungsansätze einerseits zur Lösung eines Qualitätsproblems eingesetzt werden können sowie welche Lösungsansätze andererseits aufgrund von Wechselwirkungen und Seiteneffekten mit der existierenden Architektur zur Verstärkung des Qualitätsproblems beitragen. Die Erfahrungen in der Praxis zeigen, dass beispielsweise die Wartbarkeit eines Softwaresystems häufig entweder als Flexibilität und Variabilität oder als Analysierbarkeit und Testbarkeit aufgefasst wird. Aufgrund der Widersprüche ist unklar, mit welchen Lösungsansätzen der gewünschte Aspekt der Wartbarkeit verbessert werden kann.

Um die Auswahl von Lösungsansätzen in der dritten Phase des Entscheidungsprozesses (siehe Abschnitt 5.3.2, S. 70) systematischer zu gestalten, um dadurch Komplexität, Unsicherheiten und Risiken reduzieren zu können, werden die verfügbaren Entwurfsmuster, Architekturmuster und -stile anhand ihrer Qualitätseigenschaften kategorisiert. Die Kategorisierung ist notwendig, da mittels der existierenden Kategorisierungsansätze zu wenig nach Qualitätseigenschaften differenziert werden kann (siehe die Argumentation in Abschnitt 2.7, S. 35ff.). So werden beispielsweise alle Architekturmuster, welche zur Verbesserung der Zuverlässigkeit beitragen, in eine entsprechende Kategorie eingeordnet. Für jede Kategorie wird zudem ermittelt, welche anderen positiven und negativen Qualitätseigenschaften sie besitzen. Zur Vereinheitlichung und zur Vermeidung von Widersprüchen erfolgt die Untersuchung anhand der Qualitätsmerkmale der ISO-Norm 9126 [IS9126]. Eine Übersicht über die für Architekturentscheidungen relevanten Qualitätsmerkmale wird in Abb. 7.1 gegeben (siehe hierzu insbesondere Abschnitt 2.3 ab S. 26).

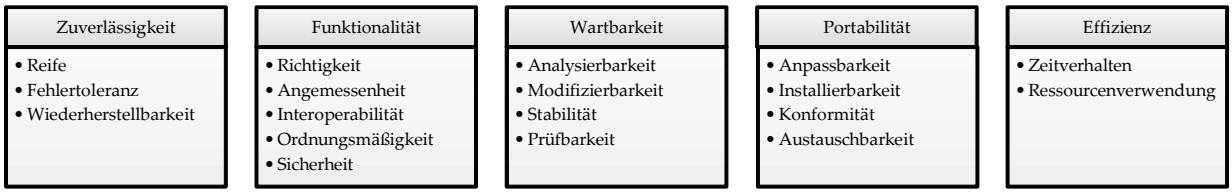


Abb. 7.1: Für Architekturentscheidungen relevante Qualitätsmerkmale der ISO-Norm 9126

Die Qualitätsmerkmale der ISO-Norm 9126 stellen nur einen Ausschnitt der in der Praxis relevanten Qualitätsmerkmale dar. Im tatsächlichen Entscheidungsfall kann es bei der Auswahl von Lösungsansätzen erforderlich sein, weitere Qualitätseigenschaften zu überprüfen. Beispiele sind die Flexibilität, die Variabilität oder die Testbarkeit, die in der ISO-Norm 9126 nur am Rande betrachtet werden.

7.1 Entwurfsmusterkategorien

Zur Kategorisierung der Entwurfsmuster werden im Rahmen der Dissertation fünf Kategorien vorgeschlagen (siehe Tab. 7.1), in denen die Entwurfsmuster mit ähnlichen Qualitätseigenschaften zusammengefasst sind. In den folgenden Abschnitten wird für jede der Kategorien untersucht, welche positiven und negativen Auswirkungen im Hinblick auf die für Architekturentscheidungen relevanten inneren Qualitätsmerkmale der ISO-Norm 9126 zu erwarten sind (siehe Abb. 7.1).

Entwurfsmuster-Kategorie	**Beschreibung**
Flexible Klassenstrukturen	Verbesserung der Flexibilität durch flexible Objekterzeugung und flache Klassenhierarchien
Variable Objekteigenschaften	Veränderbarkeit der Verhaltenseigenschaften von Objekten zur Laufzeit
Wartbare Subsysteme	Bildung von Subsystemen zur Verbesserung der Wart- und Erweiterbarkeit
Portable Klassen und Objekt	Adaption von Klassen und Objekten zur Erleichterung der Wiederverwendung
Beschleunigter Objektzugriff	Effizienter und beschleunigter Zugriff auf Objekte und Komponenten

Tab. 7.1: Kategorien von Entwurfsmustern mit ähnlichen Qualitätseigenschaften

7.1.1 Flexible Klassenstrukturen

Mithilfe dieser Entwurfsmuster kann eine verzweigte und komplexe Klassenhierarchie abgeflacht werden, die bei einer Vielzahl von Objekten mit unterschiedlichen Eigenschaften und differenter Funktionalität häufig entsteht. Sofern die Objekte eines Softwaresystems nicht durch generische Klassen definiert werden können, ist für jedes spezifische Objekt eine eigene Klasse mit abstrakten Oberklassen erforderlich. Bei einer Vielzahl unterschiedlicher Objekte entsteht häufig eine weit verzweigte und komplexe Klassenhierarchie, bei deren Analyse, Erweiterung und Veränderung eine Vielzahl von Unsicherheiten und Risiken verbleiben. Als Beispiel dient ein Softwaresystem eines Fahrzeugherstellers, das einen breit gefächerten Produktkatalog an Fahrzeugen durch Objekte abbildet. Zur Instanziierung der Fahrzeuge als Objekte sind für jeden Fahrzeugtyp entsprechende konkrete Klassen erforderlich. Durch den Einsatz der Entwurfsmuster dieser Kategorie ist nicht für jedes spezifische Objekt eine eigene Klasse mit abstrakten Oberklassen notwendig. Die Erzeugung der Objekte wird einem Konstruktor übertragen; dies kann eine 'Abstrakte Fabrik', eine 'Fabrikmethode', ein 'Erbauer' (sog. 'Builder') oder ein 'Prototyp' sein [Gamm01].

In Abb. 7.2 (siehe S. 140) ist das Beispiel zur Konstruktion verschiedener Fahrzeuge mithilfe eines 'Erbauer'-Musters anhand eines Klassendiagramms dargestellt. Zur Laufzeit greift die Klasse 'Auto' zur Konstruktion spezieller Autos auf einen 'CarBuilder' zurück. Dieser erzeugt mit den Attributen Karosserieform, Farbe und der Ausstattungsvariante (mit oder ohne Alufelgen) entweder ein 'Cabrio' oder einen 'Kombi'. Eine tiefe Klassenhierarchie für die Erzeugung verschiedener Objekt- bzw. Fahrzeugtypen ('Kombi', 'Cabrio' etc.) ist somit nicht notwendig.

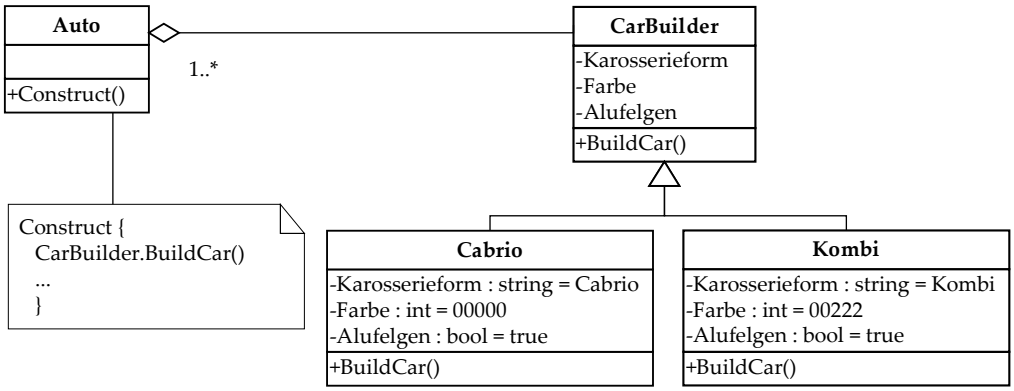


Abb. 7.2: Konstruktion von Fahrzeugen mit dem 'Erbauer'-Muster

Mit der Auslagerung der Objekterzeugung an Konstruktoren kann die Tiefe einer Klassenhierarchie reduziert sowie die Flexibilität bei der Erweiterung und Veränderung der Klassenstruktur erhöht werden. Durch die damit verbundene Reduzierung der Anzahl abstrakter Klassen werden die Verständlichkeit, die Wartbarkeit und die Flexibilität der Klassenstruktur verbessert. Jedoch müssen die durch die Konstruktoren erzeugten Objekte korrekt sein und der Spezifikation entsprechen. Diese Überprüfung erfordert zusätzlichen Test- und Analyseaufwand. Neben der Wartbarkeit wird die Portabilität verbessert, da die Konstruktoren bei einer veränderten Systemumgebung ausgetauscht und angepasste Objekte erzeugt werden können. Durch den Konstruktor werden jedoch zusätzliche Ressourcen verbraucht. Die Objekterzeugung kann sich gerade bei der Konstruktion komplexer Objekte spürbar verlangsamen. Die Problematik wird besonders beim Entwurfsmuster 'Erbauer' sichtbar, bei welchem erst nach Abschluss der Konstruktion das Objekt zurückgegeben wird [Bosc00]. Benötigt der Erbauer viele Einzelschritte zur Konstruktion eines komplexen Objektes, ist die Objekterzeugung langsamer als die reine Instanziierung auf der Grundlage einer vorhandenen spezifischen Klasse. Die Qualitätsmerkmale Sicherheit sowie Zuverlässigkeit werden hingegen nicht direkt beeinflusst. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.2 dargestellt.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	o	o	+	+

Tab. 7.2: Qualitätseigenschaften der Kategorie 'Flexible Klassenstrukturen'

7.1.2 Variable Objekteigenschaften

Das 'Strategie-' und das 'Zustandsmuster' ermöglichen die Nutzung verschiedener Objekt-Implementierungen, um die Objekteigenschaften und das Verhalten zur Laufzeit variieren zu können [Gamm01]. Beim 'Strategiemuster' können die Objekteigenschaften entsprechend einer festgelegten Strategie oder eines Workflows verändert werden. Ein Beispiel ist die Auswahl zwischen verschiedenen Kompressionsalgorithmen, die sich im Hinblick auf die Geschwindigkeit sowie auf die Kompressionsrate unterscheiden. Beim 'Zustandsmuster' können sich die Objekteigenschaften auf der Grundlage des Zustands des Objekts verändern, z. B. ob eine Datei geöffnet oder eine Netzwerkverbindung hergestellt ist. Die unterschiedlichen Objekteigenschaften sind in separaten Klassen implementiert.

Ein Beispiel für ein implementiertes 'Strategiemuster' ist in Abb. 7.3 durch ein Klassendiagramm dargestellt. Das betrachtete Objekt ist ein 'Video', welches komprimiert werden kann. Zur Videokomprimierung stehen ein 'Schneller Codec' und ein 'Experimenteller Codec' zur Verfügung. Beide Codecs unterscheiden sich im Grad der Kompression und in der Anzahl der KeyFrames je Minute. Der Anwender kann zur Laufzeit auswählen, mit welcher der beiden Codec-Implementierungen die Videokomprimierung erfolgen soll.

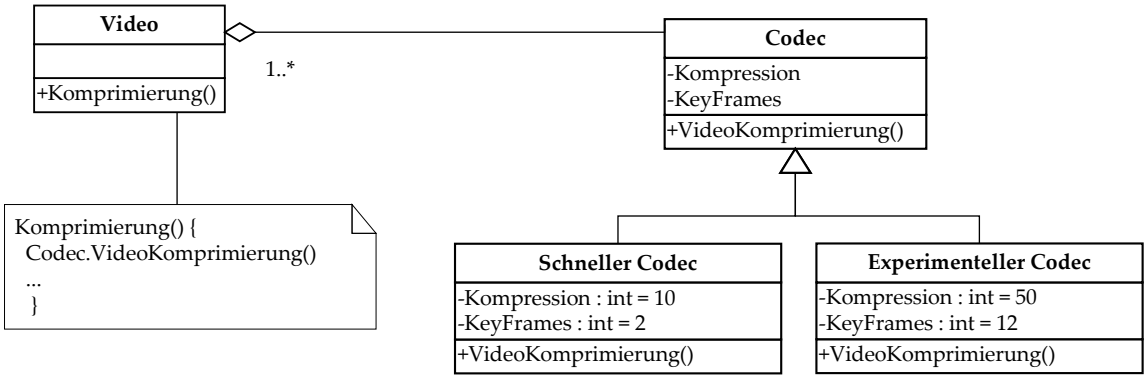


Abb. 7.3: Verwendung von zwei Codec-Implementierungen mit dem 'Strategie'-Muster

Mit dem Einsatz dieser Entwurfsmuster werden zwar die Wartbarkeit und die Variabilität verbessert, da neue Objekt-Implementierungen (z. B. ein 'experimenteller Codec', siehe Abb. 7.3) neben bereits existierenden gefahrlos getestet werden können. Die Erfahrungen aus der Praxis zeigen jedoch, dass einzelne Eigenschaften und Funktionalitäten redundant implementiert werden. Dadurch wird der Test- und Analyseaufwand erhöht, da bei Veränderungen alle redundanten Codeteile angepasst werden müssen. Neben der Wartbarkeit wird auch die Portabilität verbessert, da je nach Systemumgebung verschiedene und angepasste Implementierungen eines Objekts genutzt werden können. Die Verwendung der verschiedenen Implementierungen erzeugt zur Laufzeit jedoch einen Verwaltungs-Overhead, der sich gerade bei zeitkritischen oder Echtzeit-Softwaresystemen negativ auf die Effizienz und die Geschwindigkeit auswirken kann [Bosc00]. Vor allem bei sehr häufigen Zustandsänderungen ist in der Praxis ein deutlich verlangsamtes Laufzeitverhalten zu beobachten. Die Qualitätsmerkmale Sicherheit sowie Zuverlässigkeit werden nicht direkt beeinflusst. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.3 zu finden.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	o	o	+	+

Tab. 7.3: Qualitätseigenschaften der Kategorie 'Variable Objekteigenschaften'

7.1.3 Wartbare Subsysteme

Die Entwurfsmuster dieser Kategorie beschreiben die Bildung und Kapselung von Subsystemen, bei denen der Zugriff ausschließlich über definierte Schnittstellen erfolgt. Zu dieser Kategorie zählen u. a. die Muster 'Kompositum' mit 'Iterator' und 'Besucher', 'Fassade', 'Vermittler', 'Observer' oder 'Controller' [Gamm01] (u. [Bus+05]).

Die Zugriffe auf die gekapselten Objekte werden dabei unterschiedlich behandelt [Gamm01]. Im einfachen Fall der 'Fassade' leitet die Schnittstelle die Aufrufe auf die gekapselten Objekte einfach weiter und gibt den Rückgabewert an das aufrufende Objekt zurück. 'Oberserver' oder 'Vermittler' übernehmen Koordinierungs- und Verwaltungsaufgaben. Ein 'Observer' benachrichtigt bestimmte Objekte, sofern sich ein gekapseltes Objekt des Subsystems verändert. Ein 'Vermittler' ist eine Koordinierungsinstanz zwischen verschiedenen Objekten. Komplexe Objektzugriffe mit einer Vielzahl von Einzelaktivitäten, z. B. zur Formatierung eines Textes, nimmt der 'Vermittler' entgegen, verarbeitet diese und leitet sie an die entsprechenden Objekte weiter. Im Falle des 'Kompositums' werden Objekte so zusammengefasst, dass sie zur Laufzeit als ein einzelnes Objekt benutzt werden können. Die zusammengefassten Objekte sind dadurch Teile eines übergeordneten Objektes. Eng verbunden mit dem Kompositum sind die Muster 'Iterator' und 'Besucher'. Über einen 'Iterator' kann auf die im 'Kompositum' gekapselten Objekte direkt zugegriffen werden. Diese Objekte können durch einen 'Besucher' verändert werden.

Ein Beispiel zur Kapselung von Objekten ist in Abb. 7.4 durch ein Sequenzdiagramm dargestellt. Hierbei wird über einen 'Client', z. B. ein Java-Applet, auf eine Fassade namens 'Videorecorder' zugegriffen. Zur 'Wiedergabe' eines komprimierten Videos ist eine 'VideoDecodierung' notwendig. Der Aufruf des 'Clients' zur Wiedergabe des Videos wird an das gekapselte Objekt 'VideoDecoder' weitergeleitet.

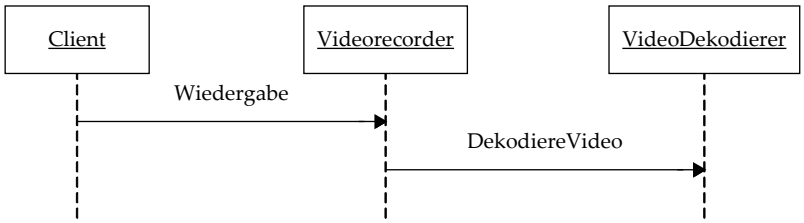


Abb. 7.4: Kapselung von Funktionen zur Videowiedergabe durch eine 'Fassade'

Durch die Kapselung der Objekte wird die Wartbarkeit verbessert, da der Zugriff auf die gekapselten Objekte, im Idealfall ausschließlich, über eine oder mehrere Schnittstellen erfolgt [Bosc00]. Dadurch kann die Fehlersuche systematischer und zielgerichteter innerhalb der jeweiligen Subsysteme erfolgen. Jedoch ist die Herauslösung bereits gekapselter Objekte aus dem Subsystem mit hohem Analyse- und Testaufwand verbunden, da die Objekte des Subsystems untereinander eine enge Bindung aufweisen. Die Schnittstellen fungieren als 'Single-Point-of-Access' in das Subsystem [Bosc00]; Fehler innerhalb des Subsystems können durch entsprechende Ausnahmen auf der Ebene der Schnittstellen abgefangen werden, um die Zuverlässigkeit des Softwaresystems zu erhöhen. Auf der Ebene der Schnittstellen können außerdem Sicherheitsfunktionen implementiert werden, z. B. die Reglementierung des Zugriffs auf die gekapselten Objekte. Da die Zugriffe auf die Objekte über die Schnittstelle des Subsystems erfolgen, kann es bei einem unpassenden Schnittstellendesign zu Verzögerungen beim Zugriff kommen [Bosc00]. Derartige Verzögerungen können zu einem ineffizienten Umgang mit Speicher- und Prozessorressourcen führen. Die Portabilität wird hingegen nicht direkt beeinflusst. Die nachstehende Tabelle Tab. 7.4 gibt eine Übersicht über die Qualitätseigenschaften.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	+	+	+	o

Tab. 7.4: Qualitätseigenschaften der Kategorie 'Wartbare Subsysteme'

7.1.4 Portable Klassen und Objekte

In dieser Kategorie sind jene Entwurfsmuster zusammengefasst, mit denen existierende Objekte und Klassen adaptiert und dadurch angepasst, erweitert und in andere Systemumgebungen portiert werden können. Zu dieser Kategorie zählen insbesondere die Entwurfsmuster 'Adapter' und 'Dekorierer' [Gamm01]. Das 'Adaptermuster' beschreibt eine Struktur, bei welcher inkompatible Schnittstellen zweier Klassen durch einen 'Adapter' überbrückt werden. Mittels Mehrfachvererbung kann der 'Adapter' auf die inkompatiblen Schnittstellen zugreifen und diese überbrücken. Ist beispielsweise ein Viereck zu zeichnen und die relevante Klasse verfügt ausschließlich über die Funktionalität zum Zeichnen von Linien, erbt der 'Adapter' von beiden Klassen die Methoden, z. B. *'ZeichneViereck'* und *'ZeichneLinie'*. Bei Aufruf von *'ZeichneViereck'* setzt der 'Adapter' aus vier Linien das Viereck zusammen. Ein 'Dekorierer' agiert als Wrapper, der ein Objekt kapselt und um die Funktionalität erweitert. Diese Erweiterung ist transparent bzw. unbekannt für das zugrunde liegende Objekt, da das gekapselte Objekt selbst nicht verändert wird.

Das Beispiel einer 'Dekoration' eines Videos ist in Abb. 7.5 anhand eines Klassendiagramms wiedergegeben. Ein 'Video' wird durch zwei 'Dekorierer' erweitert, die entweder 'Untertitel' oder eine ergänzende 'Tonspur' hinzuzufügen. Auf das zugrunde liegende 'Video' kann aber auch ohne die 'Dekorierer' zugegriffen werden.

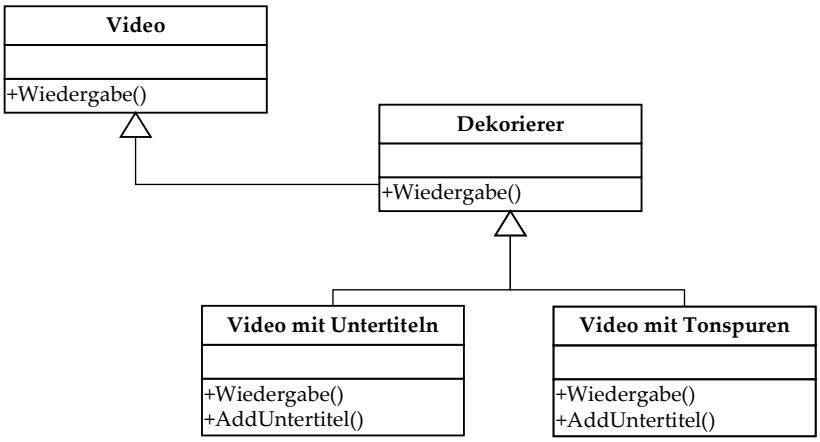


Abb. 7.5: Erweiterung eines Video-Objektes durch 'Dekorierer'

Durch den Einsatz dieser Muster sind positive Auswirkungen auf die Portabilität zu erwarten, da existierende Klassen und Objekte bei veränderten Systemumgebungen adaptiert, angepasst und erweitert werden können. Analog zu den Entwurfsmustern zur Kapselung von Objekten (siehe Abschnitt 7.1.3, S. 141f.) stellen diese Muster einen 'Single-Point-of-Access' zu den adaptierten Klassen und Objekten dar. Somit können Fehlerbehandlungsroutinen oder Sicherungsmechanismen zur Verbesserung der Zuverlässigkeit und Sicherheit implementiert werden. Jedoch sind beide Muster aufgrund ihrer Fehleranfälligkeit und schlechten Testbarkeit umstritten [Klae04]. Das 'Adaptermuster' verwendet die für die Analysier- und Testbarkeit kritische Mehrfachvererbung; der 'Dekorierer' ist mit einer in der Objektorientierung generell unerwünschten Implementierungsvererbung gleichzusetzen. Dies beeinflusst die Wartbarkeit negativ, da die Verständlichkeit und die Testbarkeit vermindert sowie der Analyseaufwand zur Fehlersuche deutlich erhöht werden. Die Praxis zeigt außerdem, dass die 'Adaption' oder 'Dekoration' von Objekten und Klassen im Vergleich zur Neuimplementierung der Funktionalität häufig ein ineffizientes Vorgehen ist. Das wird am Beispiel zum Zeichnen eines Vierecks deutlich, bei dem durch den Adapter vier Schritte anstatt eines einzigen Schrittes erfolgen (siehe Beispiel auf S. 142). Ist dies z. B. eine sehr häufig benutzte Funktion, sind Performance-Engpässe zu erwarten. Eine Übersicht über die Qualitätseigenschaften wird in Tab. 7.5 gegeben.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	+	+	-	+

Tab. 7.5: Qualitätseigenschaften der Kategorie 'Portable Klassen und Objekte'

7.1.5 Beschleunigter Objektzugriff

Die Entwurfsmuster 'Fliegengewicht' und 'Proxy' [Gamm01] ermöglichen einen besonders effizienten und beschleunigten Zugriff auf Objekte. Über das 'Fliegengewicht' wird die Anzahl verschiedener Objekte reduziert, die sich invariante Informationen teilen. Zur Verbesserung der Effizienz und der Ladezeiten werden diese Informationen an ein 'Fliegengewicht'-Objekt ausgelagert. Die im 'Fliegengewicht' enthaltenen Informationen werden wiederum von vielen Objekten gemeinsam benutzt. Ein 'Proxy' ist hingegen ein Zeiger auf ein entferntes Objekt, z. B. ein Objekt auf einem separaten Server. Über den 'Proxy' kann das entfernte Objekt benutzt werden, ohne dass es lokal instanziiert werden muss.

Durch den Einsatz von 'Fliegengewichten' und 'Proxies' werden Zugriffe auf Objekte effizienter. Jedoch steigt der Wartungs- und Analyseaufwand aufgrund dieser komplexen Strukturen an [Bosc00]. Die an ein 'Fliegengewicht' ausgelagerten invarianten Informationen werden in verschiedenen Kontexten verwendet, die bei der Fehlersuche und beim Testen zu berücksichtigen sind. Ein 'Proxy' führt bei der Fehleranalyse oder der Veränderung des Objektes, auf das der 'Proxy' zeigt,

ebenfalls zu einem höheren Analyse- und Testaufwand. Der Einsatz dieser Entwurfsmuster hat hingegen keinen direkten Einfluss auf die Sicherheit, die Portabilität und die Zuverlässigkeit. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.6 dargestellt.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
+	o	o	-	o

Tab. 7.6: Qualitätseigenschaften der Kategorie 'Beschleunigter Objektzugriff'

7.1.6 Zusammenfassung

Die Qualitätseigenschaften der fünf Entwurfsmusterkategorien sind in Tab. 7.7 zusammengefasst.[7]

Kategorie	Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
Flexible Klassenstrukturen	-	o	o	+	+
Variable Objekteigenschaften	-	o	o	+	+
Wartbare Subsysteme	-	+	+	+	o
Portable Klassen und Objekte	-	+	+	-	+
Beschleunigter Objektzugriff	+	o	o	-	o

Tab. 7.7: Überblick über die Qualitätseigenschaften der Entwurfsmusterkategorien

7 Kennzeichnung: - negativer / + positiver / o neutraler Einfluss auf die Qualitätseigenschaften

7.2 Architekturmusterkategorien

Analog zur Kategorisierung der Entwurfsmuster werden im Rahmen der vorliegenden Dissertation vier Kategorien von Architekturmustern vorgeschlagen (siehe Tab. 7.8); die Muster jeder Kategorie besitzen dabei vergleichbare Qualitätseigenschaften. In den nachstehenden Abschnitten erfolgen eine detaillierte Beschreibung der Architekturmusterkategorien und die Analyse der Eigenschaften hinsichtlich der relevanten Merkmale der ISO-Norm 9126 (siehe S. 138).

Architekturmuster-Kategorie	Beschreibung
Effizientes Prozess- und Threadmanagement	Management von Prozessen und Threads zur effizienten Ressourcennutzung
Portable Softwaresysteme	Verbesserung der Portabilität des Softwaresystems oder von Systemteilen
Flexible Komponentenstrukturen	Realisierung einer leicht anpassbaren Komponentenstruktur mit flexiblen Zugriffsmöglichkeiten und variablen Benutzeroberflächen
Ausfallsichere Systemteile	Überwachung von Komponenten-Zuständen und Reaktion auf Fehler

Tab. 7.8: Kategorien von Architekturmustern mit ähnlichen Qualitätseigenschaften

7.2.1 Effizientes Prozess- und Threadmanagement

Mit der Implementierung der Architekturmuster dieser Kategorie wird das Ziel verfolgt, das Prozess- und Threadmanagement auf der Ebene des Betriebssystems zu optimieren und den Ressourcenverbrauch effizienter zu gestalten. Dafür existiert eine Vielzahl an Architekturmustern [Bosc00] (u. [Bus+05]); eines der bekanntesten Muster ist der 'Application-Level-Scheduler'. Ein solcher 'Scheduler' ist eine Anwendung zur effizienten Koordinierung und Verwaltung der Prozesse und Threads des Softwaresystems. Zum Scheduling existieren weitere Muster, z. B. zur nicht-präemptiven Verteilung der Prozessorressourcen an Threads. Im Gegensatz zum rein präemptiven Scheduling werden dem Thread vor dessen Beendigung die Ressourcen nicht entzogen. Der Thread kann dadurch ohne Unterbrechung vollständig abgearbeitet werden. Darüber hinaus existieren Muster zur Strukturierung paralleler Prozessabläufe.

Diese Architekturmuster greifen tief in die Speicher- und Prozessorverwaltung des Betriebssystems ein [Bosc00] (u. [Bus+05]). Die Wartung und Veränderung dieser Muster und deren Implementierungen erfordern einen dementsprechend hohen Analyse- und Testaufwand, da Fehler das gesamte Softwaresystem, welches auf der Betriebssystemebene aufsetzt, beeinträchtigen können. Durch die Optimierung des Prozess- und Threadmanagements, z. B. durch Parallelisierung von Abläufen, kann jedoch ein effizienter Ressourcenverbrauch und ein beschleunigtes Laufzeitverhalten des Softwaresystems ermöglicht werden. Die Sicherheit, die Zuverlässigkeit und die Portabilität werden nicht direkt beeinflusst. Die folgende Tabelle Tab. 7.9 gibt eine Übersicht über die Qualitätseigenschaften.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
+	o	o	-	o

Tab. 7.9: Qualitätseigenschaften der Kategorie 'Effizientes Prozess- und Threadmanagement'

7.2.2 Portable Softwaresysteme

Zu dieser Kategorie zählen u. a. die Architekturmuster 'Microkernel' oder 'Interpreter' [Bus+05], welche die Portabilität eines Softwaresystems verbessern. Der 'Microkernel' stellt eine Plattform zur Ausführung des Softwaresystems dar (siehe Abb. 7.6) und ist vergleichbar mit einem eigenständigen Betriebssystem für ein Softwaresystem. In der Praxis umfasst ein 'Microkernel' meist weniger Funktionen als ein monolithischer Betriebssystemkernel; oftmals sind lediglich grundlegenden Funktionen zur Speicher- und Prozessverwaltung bzw. Prozesskommunikation implementiert. Durch die Verwendung verschiedener, auf entsprechende Hard- und Softwarekonfigurationen (Betriebssysteme, Speicher etc.) abgestimmter 'Microkernel' können Softwaresysteme in den unterschiedlichen Systemumgebungen eingesetzt werden. Eine weitere Möglichkeit zur Verbesserung der Portabilität ist der Einsatz eines 'Interpreters', welcher als virtuelle Maschine eine bestimmte Hard- oder Softwarekonfiguration simuliert und auf der Zielplattform den Bytecode der Software ausführt [Sha+96]. Eine der bekanntesten Implementierungen des 'Interperter'-Musters ist die Java-Runtime-Engine.

In Abb. 7.6 ist eine schematische Darstellung des 'Microkernel'- und des 'Interpreter'-Musters zu sehen. Das Softwaresystem ist durch die beiden Muster vom Betriebssystem, sowie von der Hardware (Speicher, Prozessor) weitgehend entkoppelt. Durch angepasste 'Microkernel' oder 'Interpreter' kann das Softwaresystem mit verschiedenen Hard- und Softwarekonfigurationen eingesetzt werden.

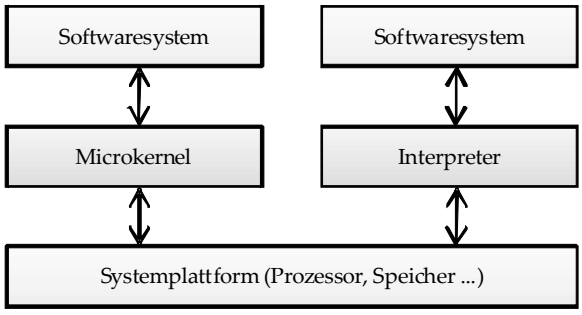


Abb. 7.6: Schematische Darstellung von 'Microkernel' und 'Interpreter'

Die Architekturmuster zur Verbesserung der Portabilität haben die folgenden Qualitätseigenschaften. Durch die Abstraktion der zugrunde liegenden Systemplattform (Speicher, Prozessor etc.) wird die Portabilität des Softwaresystems verbessert. Jedoch werden bei der Transformation und Weiterleitung der Zugriffe auf die Systemplattform (durch den 'Microkernel' oder 'Interpreter') zusätzliche Speicher- und Prozessorressourcen verbraucht [Bus+05]. Außerdem ist der Funktionsumfang zur Speicher- und Prozessorverwaltung reduziert, da z. B. 'Interpreter' keine Register nutzen können. Daher führt der Einsatz dieser Muster in der Praxis häufig zu einem ineffizienten Umgang mit Speicher- und Prozessorressourcen. Analog zu den Mustern im vorangegangenen Abschnitt 7.2.1 (siehe S. 145) sind 'Microkernel' oder 'Interpreter' kritische Strukturen für ein Softwaresystem, da diese als Laufzeitumgebung für das Softwaresystem aufzufassen sind. Bei Wartungen und Veränderungen am 'Microkernel' oder 'Interpreter' ist ein entsprechend hoher Analyse- und Testaufwand erforderlich. Die Zuverlässigkeit und Sicherheit werden durch diese Muster nicht direkt beeinflusst. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.10 dargestellt.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	o	o	-	+

Tab. 7.10: Qualitätseigenschaften der 'Portablen Softwaresysteme'

7.2.3 Flexible Komponentenstrukturen

Aufgrund der Vielzahl an unterschiedlichen Mustern, die eine flexible Komponentenstruktur ermöglichen, wird diese Kategorie in drei Unterkategorien unterteilt. Alle drei Unterkategorien besitzen jedoch identische Qualitätseigenschaften.

Variable Strukturierung von Komponenten

Die Architekturmuster 'Reflection' [Bus+05] oder 'Service-Oriented-Architectures' (SOA) [Pos+04] fokussieren auf eine flexible und einfache Veränderbarkeit der Komponentenstruktur. Durch diese Muster wird das Softwaresystem in zwei Schichten aufgeteilt: die Repräsentations- (oder Services-) und die Implementierungsschicht. Im Falle des Musters 'Reflection' erfolgt auf der übergeordneten Repräsentationsschicht eine Selbstrepräsentation des Softwaresystems über Metaobjekte. Durch die Metaobjekte und deren Beziehungen wird die Architektur des Softwaresystems beschrieben. Den Metaobjekten sind konkrete Objekte der Implementierungsschicht (auch als Basisebene bezeichnet) zugeordnet; die Kommunikation zwischen beiden Schichten erfolgt über ein Metaobjektprotokoll. Veränderungen an der Repräsentation wirken sich somit direkt auf die Objekte der Implementierungsschicht und damit auf die Architektur und das Verhalten des Softwaresystems aus. Im Falle der 'SOA' werden auf der Repräsentationsschicht Dienste bzw. Services definiert, welchen sog. 'Business-Objects' zugeordnet sind. Die 'Business-Objects' umfassen die notwendige Funktionalität, um die Dienste auszuführen.

In Abb. 7.7 ist ein Beispiel für eine SOA wiedergegeben. Zwei Services sind definiert: 'Neukunden werben' und 'Kundendaten aufnehmen'. Das sind in der Praxis typische Schritte eines Akquiseprozesses. Die Services greifen auf entsprechende Business-Objects ('Planung Werbung', 'Daten verarbeiten', 'Statistiken erstellen') zurück. Im zweiten Schritt des Akquiseprozesses werden die Kundendaten ergänzt und verfeinert.

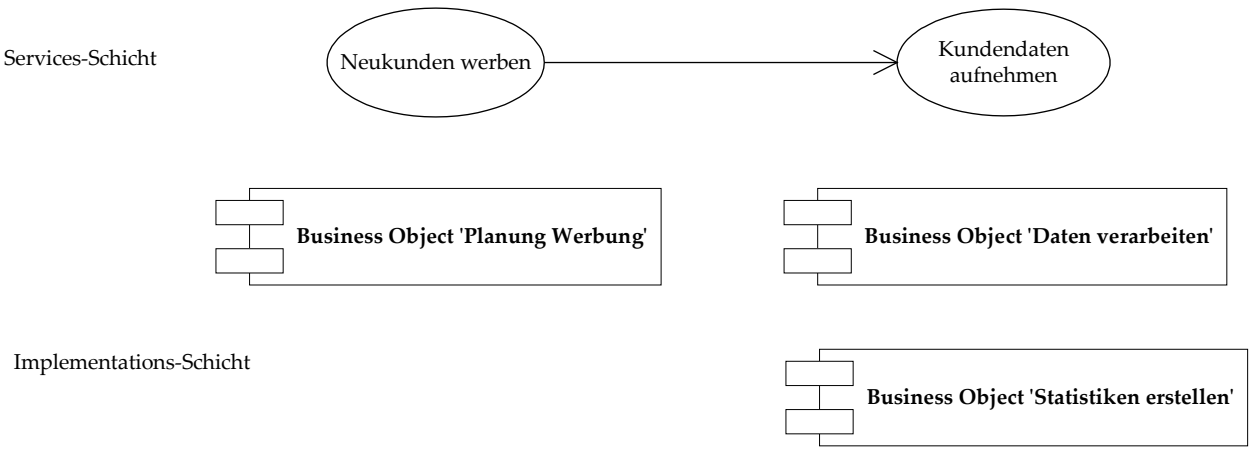


Abb. 7.7: Beispiel für eine 'Service-Oriented-Architecture'

Durch die zusätzliche Repräsentationsschicht werden Workflows oder Prozessmodelle von der zugrunde liegenden Implementierung der Funktionalität getrennt. Dadurch wird eine einfache und flexible Veränderbarkeit des Softwaresystems ermöglicht, wodurch positive Auswirkungen auf die Wartbarkeit zu erwarten sind. Außerdem wird die Portabilität positiv beeinflusst, da die bereits implementierten Objekte in verschiedenen Kontexten wiederverwendet oder ausgetauscht werden können. Die Erfahrung aus der Praxis zeigt jedoch, dass häufig eine enge Kopplung zwischen den Meta-Objekten oder Services sowie den konkreten Objekten der Implementierungsschicht festzustellen ist. In diesen Fällen werden bei Veränderungen auf der Abstraktionsschicht auch Veränderungen an der Implementierungsschicht notwendig, was die Wiederverwendbarkeit der bereits implementierten Objekte begrenzt. Durch die Aufteilung der Architektur in die beiden Schichten werden darüber hinaus zusätzliche Speicher- und Prozessorressourcen verbraucht, was die Effizienz des Softwaresystems vermindert. So müssen spezielle Protokolle implementiert werden, damit eine Kommunikation zwischen den Schichten ermöglicht wird. Die Sicherheit und die Zuverlässigkeit werden von diesen Architekturmustern hingegen nicht direkt beeinflusst. Eine Übersicht über die Qualitätseigenschaften gibt Tab. 7.11.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	o	o	+	+

Tab. 7.11: Qualitätseigenschaften der Unterkategorie 'Variable Strukturierung von Komponenten'

Flexibler Zugriff auf verteilte Komponenten

Die Architekturmuster 'Broker', 'Forwarder-Receiver' oder 'Proxy' ermöglichen einen flexiblen Zugriff auf Komponenten [Bus+05], insbesondere in verteilten Systemumgebungen. Diese Architekturmuster fungieren als Makler zwischen einem Client und der aufgerufenen Komponente, die sich auf einem entfernten Server befindet. Als vermittelnde Instanz leiten sie Nachrichten des Clients, z. B. Funktionsaufrufe, an die entsprechenden Komponenten weiter. Die Art und Weise der Weiterleitung erfolgt z. B. beim 'Broker' unter dem Aspekt eines ausgewogenen Lastenausgleichs. Im Falle des 'Brokers' werden die Funktionsaufrufe des Clients an den Server weitergeleitet, der die Komponente enthält und noch über freie Kapazitäten verfügt. Diese Weiterleitung ist für den Client transparent und daher nicht sichtbar.

In Abb. 7.8 ist anhand eines Sequenzdiagramm dargestellt, wie eine Client-Anwendung 'Videorecorder' über einen 'Broker' auf Videos von verschiedenen Servern zugreift. Die Videos sind einerseits auf 'Server A' gespeichert und können vom 'Videorecorder' aufgerufen und abgespielt werden. Ist 'Server A' ausgelastet, steht mit 'Server B' ein Mirror der gespeicherten Videos zur Verfügung. Je nach Belastung leitet der Broker die Clientzugriffe entweder auf 'Server A' oder 'Server B'.

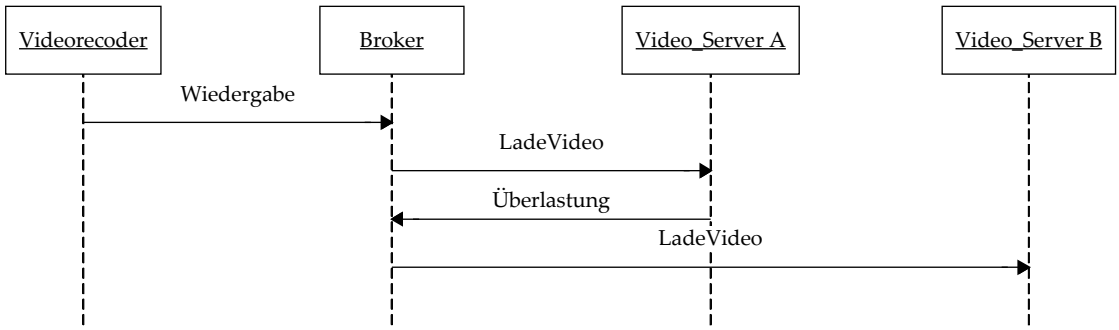


Abb. 7.8: 'Broker'-Muster für den Zugriff auf verteilte Videos

Die Architekturmuster dieser Kategorie ermöglichen einerseits einen flexiblen Zugriff auf verteilte Komponenten. Dies erleichtert die Wartbarkeit, da die Interprozess-Kommunikation direkt über den Makler ('Broker', 'Proxy' etc.) erfolgt und vom Client entkoppelt ist. Darüber hinaus wird die Portabilität verbessert, da je nach Systemumgebung auf angepasste Komponenten zugegriffen werden kann, z. B. auf verschiedene Codecs. Andererseits sind negative Auswirkungen im Hinblick auf das Laufzeitverhalten zu erwarten, da die Kommunikation zwischen Client und den Komponenten über den Makler einen Verwaltungs-Overhead erzeugt. Die Objektzugriffe müssen erst vom Makler empfangen, ausgewertet und entsprechend weitergeleitet werden, was vor allem bei einer Vielzahl an gleichzeitigen Zugriffen zeitkritisch werden kann. Die Sicherheit und die Zuverlässigkeit werden von diesen Architekturmustern hingegen nicht direkt beeinflusst. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.12 gegeben.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	o	o	+	+

Tab. 7.12: Qualitätseigenschaften der Unterkategorie 'Flexibler Zugriff auf verteilte Komponenten'

Variable Benutzeroberflächen

Zur flexiblen Gestaltung von Benutzeroberflächen können die beiden Architekturmuster 'Model-View-Controller' (MVC) und 'Presentation-Abstraction-Control' [Bosc00] (u. [Bus+05]) eingesetzt werden. Das 'MVC'-Muster beschreibt eine Archi-

tektur mit drei Schichten, auf die die zur Verarbeitung der Benutzereingaben notwendige Funktionalität aufgeteilt sind: Views, Controller und Model. Das Model beinhaltet die Daten des Softwaresystems und die Funktionalität, um die auf die Daten zugreifen zu können. Über Views stehen dem Benutzer eine oder mehrere Sichten, z. B. Formulare zur Dateneingabe, auf das Model zur Verfügung. Durch einen Controller zwischen den Views und dem Model werden die Benutzereingaben sowie die zugehörigen Ausgaben empfangen, weitergeleitet und verarbeitet. Beim 'PAC'-Muster stehen Agenten im Vordergrund, die drei Schichten zugeordnet sind: Top-Level-, Intermediate-Level- und Bottom-Level-Agenten. Der Benutzer kommuniziert nur mit den Agenten des Bottom-Level direkt. Bei Bedarf werden Anfragen von dort an die Agenten der höheren Ebenen weitergeleitet. Jeder Agent ist in drei Komponenten aufgeteilt, die der Struktur des MVC-Musters gleichen. Somit hat jeder Agent eine eigene Presentation- und Abstraction-Komponente (View und Model) sowie einen eigenen Controller, der zwischen den beiden Komponenten vermittelt. Darüber hinaus ist der Controller für die Kommunikation zwischen den Agenten verantwortlich.

Mit den Architekturmustern dieser Kategorie wird die Wart- und Erweiterbarkeit der Benutzerschnittstelle verbessert. So können dank des Controllers mehrere Views – auch experimentelle Views – auf ein Model implementiert werden. Es besteht jedoch die Gefahr, dass bei ähnlichen Views redundante Darstellungselemente und die zugehörige Funktionalität implementiert werden, z. B. Listen oder Schaltflächen mit Event-Handling-Funktionalität. Sind diese redundanten Darstellungselemente zu verändern, wie z. B. die Bezeichnung einer Schaltfläche, müssen alle entsprechenden Views angepasst werden. Das erhöht den Wartungsaufwand deutlich. Der Controller selbst agiert als 'Observer' ([Bus+05], siehe S. 141) und erzeugt bei jeder einzelnen Benutzerinteraktion, z. B. Mausbewegungen und -klicks, eine Fülle von Update-Nachrichten. Diese müssen verarbeitet werden, was sich negativ auf das Laufzeitverhalten und damit auf die Effizienz des Softwaresystems auswirken kann, da auch unwesentliche Benutzerinteraktionen, z. B. Fehlklicks, verarbeitet werden müssen. Die Portabilität wird verbessert, da eine komplette View je nach Systemumgebung flexibel ausgetauscht werden kann, wie z. B. durch eine für Mobiltelefone optimierten View. Die Sicherheit und die Zuverlässigkeit werden nicht direkt beeinflusst. In Tabelle Tab. 7.13 ist eine Übersicht über die Qualitätseigenschaften zu finden.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
-	o	o	+	+

Tab. 7.13: Qualitätseigenschaften der Unterkategorie 'Variable Benutzeroberflächen'

7.2.4 Ausfallsichere Systemteile

Zu dieser Kategorie zählen Architekturmuster, welche die Zustände von Komponenten überwachen und dadurch die Ausfallsicherheit sowie Zuverlässigkeit verbessern. Beispiele sind 'Watchdog' [Sil+03] oder das 'Safety-Executive-Pattern' [Doug99]. Ein 'Watchdog' überprüft den Zustand von Komponenten und benachrichtigt das System, wenn eine überwachte Komponente nicht mehr erreichbar ist. Beim Einsatz des 'Safety-Executive-Patterns' erfolgt neben der Überwachung auch eine Fehleranalyse. Ist eine überwachte Komponente aufgrund eines Fehlers nicht mehr erreichbar, werden dem System der Ausfall und zusätzlich die Art des Fehlers gemeldet, wie z. B. eine Null-Pointer-Exception.

Anhand eines Sequenzdiagramms ist in Abb. 7.9 der Einsatz eines 'Safety-Executive-Pattern' dargestellt; dabei wird die Netzwerkverbindung zu einem TV-Stream auf 'Server A' überwacht. Zur Laufzeit nimmt ein 'Videorecorder' diesen Stream auf. Sollte der 'Stream auf Server A' ausfallen, wird dies durch eine 'Kontrollinstanz' erkannt und es wird die Art des Fehlers an den 'Videorecorder' übermittelt. Bei einer Unterbrechung der Netzwerkverbindung kann z. B. versucht werden, eine erneute Verbindung zu 'Server A' herzustellen. Ist der Server jedoch ausgelastet, wird versucht, den alternativen 'Stream auf Server B' zu empfangen und die Aufzeichnung fortsetzen.

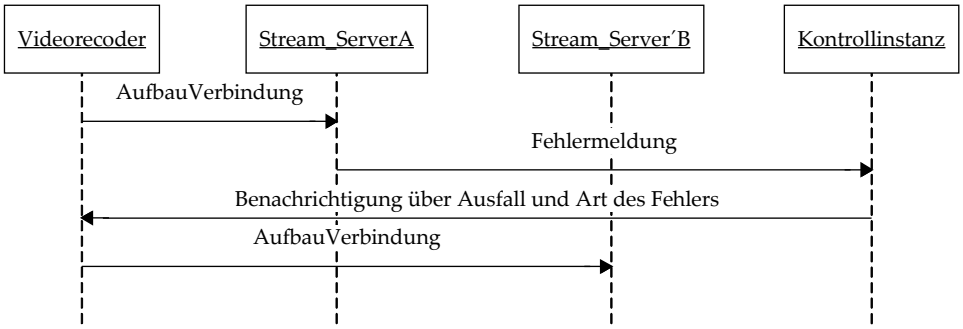


Abb. 7.9: 'Safety Executive Pattern' zur Überwachung der Netzwerkverbindung

Durch den Einsatz dieser Architekturmuster kann die Zuverlässigkeit verbessert werden, da Fehler, Probleme oder Fehlkonfigurationen in den Komponenten frühzeitig – und nicht erst bei Folgeproblemen – erkannt werden. Insbesondere im Fall des Safety-Executive-Patterns ist eine geeignete Reaktion auf erkannte Fehler möglich. Es können spezifische Gegenmaßnahmen eingeleitet werden, da die Art des Fehlers an das System übermittelt wird. Diese Muster agieren passiv im Hintergrund und melden erst im Fall eines Fehlers bei der überwachten Komponente diesen an die angeschlossenen Komponenten, z. B. an den 'Videorecorder' (siehe Abb. 7.9). Daher sind keine direkten Auswirkungen auf die Wartbarkeit, Sicherheit, Portabilität und Effizienz des Softwaresystems zu erwarten. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.14 dargestellt.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
o	o	+	o	o

Tab. 7.14: Überblick über die Qualitätseigenschaften der Kategorie Zuverlässigkeit

7.2.5 Zusammenfassung

Die Qualitätseigenschaften der vier Architekturmusterkategorien sind in Tab. 7.15 zusammenfassend dargestellt.[8]

Kategorie	Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
Effizientes Prozess- und Threadmanagement	+	o	o	-	o
Portable Softwaresysteme	-	o	o	-	+
Flexible Komponentenstrukturen	-	o	o	+	+
Ausfallsichere Systemteile	o	o	+	o	o

Tab. 7.15: Überblick über die Qualitätseigenschaften der Architekturmusterkategorien

8 Kennzeichnung: - negativer / + positiver / o neutraler Einfluss auf die Qualitätseigenschaften

7.3 Architekturstil-Kategorien

Neben der Kategorisierung von Entwurfs- und Architekturmustern (siehe Abschnitt 7.2 und 7.1) werden zur Erleichterung der Vorauswahl von Lösungsansätzen in Abschnitt 5.3.2 (siehe S. 70) die folgenden beiden Architekturstil-Kategorien im Hinblick auf ihre Qualitätseigenschaften untersucht (siehe Tab. 7.16). Die Stile in den jeweiligen Kategorien besitzen vergleichbare Qualitätseigenschaften. Da Architekturstile im Vergleich zu Entwurfs- und Architekturmustern sehr abstrakt sind und kaum konkrete Implementierungsvorgaben beinhalten, können die Qualitätseigenschaften nur sehr grob ermittelt werden; im tatsächlichen Entscheidungsfall ist daher zu überprüfen, ob es Abweichungen gibt.

Architekturstil-Kategorien	Beschreibung
Wartbare Komponentenstrukturen mit klar definierten Zugriffsvarianten	Verbesserung der Wartbarkeit durch Kapselung und Bildung von Subsystemen bei klar definierten Zugriffsvarianten
Flexible, veränderbare und effiziente Komponentenstrukturen	Umsetzung einer hohen Kohäsion und einer losen Kopplung innerhalb der Komponentenstruktur zur Verbesserung der Flexibilität, Veränderbarkeit und Effizienz

Tab. 7.16: Kategorien von Architekturstilen mit ähnlichen Qualitätseigenschaften

7.3.1 Wartbare Komponentenstrukturen mit klar definierten Zugriffsvarianten

Die Architekturstile dieser Kategorie kapseln Komponenten und ermöglichen einen strukturierten Zugriff. Beispiele sind der 'Client-Server'- oder der 'Schichten'-Architekturstil [Sha+96] (u. [Bus+05]). Bei einer 'Client-Server'-Architektur kapselt ein Server ein oder mehrere Komponenten, auf die ein Client über Schnittstellen zugreifen kann. Der Server stellt dem Client dadurch Dienste zur Verfügung. Beispiele für Server sind Mail-, News- oder FTP-Server; die Server können untereinander flexibel zusammenarbeiten.

In Abb. 7.10 ist ein Sequenzdiagramm des Zugriffs auf einen FTP-Server bei einer 'Client-Server'-Architektur abgebildet. Der 'FTP-Server' bietet dem Client verschiedene Dienste an, z. B. die Übertragung von Dateien. Um die Dienste zu nutzen, verbindet sich der 'Client' mit einem 'FTP-Server'. Nachdem der Client die Anmeldedaten übermittelt hat, erfolgt die Authentifizierung der Anmeldedaten, z. B. durch einen 'LDAP-Server' auf der Basis des Lightweight-Directory-Access-Protocol (LDAP). Nach erfolgreicher Authentifizierung kann der Client die Dienste des 'FTP-Servers' nutzen und die 'Datenübertragung' zwischen Client und FTP-Server startet.

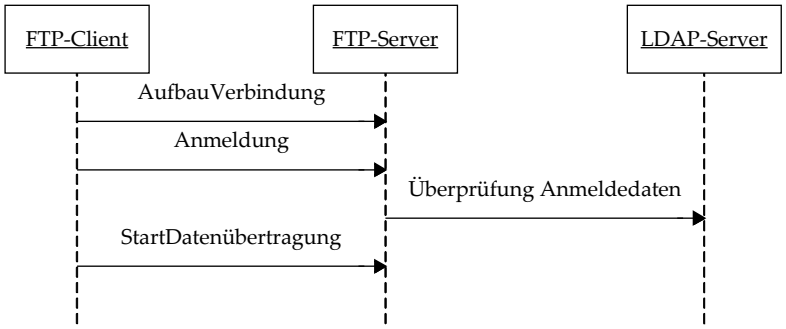


Abb. 7.10: 'Client-Server' Architekturstil am Beispiel einer FTP-Verbindung

Beim 'Schichten'-Architekturstil werden zusammengehörige Komponenten, z. B. die eines Servers, in hierarchischen Schichten angeordnet und gekapselt. Der Zugriff auf Komponenten einer Schicht erfolgt sequenziell von der untersten zur obersten Schicht oder in umgekehrter Reihenfolge. Sprünge über mehrere Schichten (sog. Layer-Bridging) sind zwar nicht gestattet, werden jedoch aus Effizienz- und Geschwindigkeitsgründen häufig implementiert. Jede Schicht stellt dabei eine

eigenständige Funktionalität zur Verfügung, auf der die darüber liegenden Schichten aufbauen. Ein Beispiel ist die Verschlüsselung einer Netzwerkverbindung, auf der andere Dienste bzw. Schichten aufbauen.

In Abb. 7.11 ist anhand eines Sequenzdiagramms dargestellt, wie die Datenübertragung bei einer FTP-Verbindung durch den Einsatz spezieller 'Schichten' verschlüsselt werden kann. 'Ein FTP-Client' stellt eine Datenanfrage an einen 'FTP-Server'. Diese Anfrage wird durch die entsprechenden Schichten ver- und entschlüsselt und an den 'FTP-Server' weitergeleitet. Nach erfolgreicher Authentifizierung durch einen LDAP-Server werden die Daten nicht an den 'FTP-Client' direkt übermittelt, sondern müssen auf Server- und Client-Seite erst ver- und entschlüsselt werden (siehe zur unverschlüsselten FTP-Übertragung Abb. 7.10).

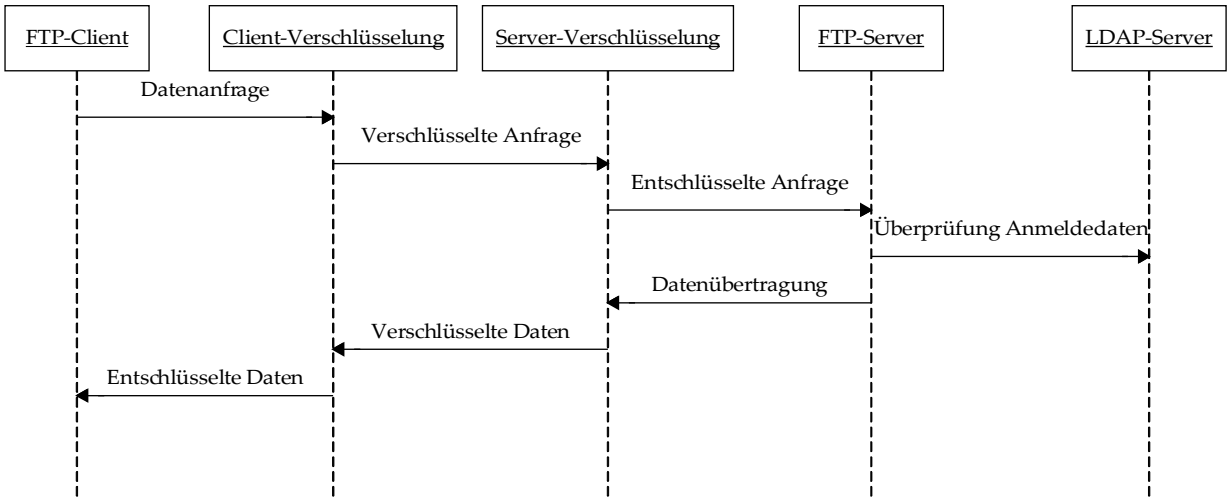


Abb. 7.11: 'Schichten' zur Verschlüsselung der FTP-Datenübertragung

Die Architekturstile dieser Kategorie haben die folgenden Qualitätseigenschaften. Durch die Kapselung zusammengehöriger Komponenten, in Form von Servern oder Schichten, wird ein strukturierter Zugriff auf die gekapselten Komponenten ermöglicht. Dadurch wird zudem die Verständlichkeit des Kontroll- und Datenflusses verbessert und der Aufwand für dessen Analyse reduziert. Beide Architekturstile wirken sich somit positiv auf die Wartbarkeit aus. Aufgrund des strukturierten Kontrollflusses ist es außerdem leichter möglich, geeignete Mechanismen zur Verbesserung der Zuverlässigkeit und Sicherheit zu implementieren. So können spezielle Schichten oder Serverinstanzen implementiert werden, um bei Fehlern in untergeordneten Schichten eine geeignete Fehlerbehandlung durchzuführen. Zudem wird die Portabilität verbessert, da die Systemumgebung auf der Clientseite weitgehend unabhängig von der Systemumgebung auf der Serverseite verändert werden kann. Jedoch ist mit der Kapselung der Komponenten eine Beschränkung der Zugriffsmöglichkeiten verbunden. Beim 'Client-Server'-Stil kann nur auf die Dienste zugegriffen werden, welche die Schnittstellen des Servers bereitstellen. Eine stärkere Beschränkung der Zugriffsmöglichkeiten erfolgt beim Einsatz einer 'Schichten'-Architektur. Aufrufe von der obersten zur untersten Schicht müssen erst die dazwischen liegenden Schichten passieren. Sprünge zwischen Schichten sind zwar nicht gestattet; in der Praxis werden jedoch aus Geschwindigkeitsgründen häufig Brücken über mehrere Schichten implementiert. Diese Beschränkungen der Zugriffe, vor allem bei der Schichten-Architektur, können jedoch zu einem ineffizienten Umgang mit Speicher- und Prozessorressourcen des Softwaresystems führen. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.17 dargestellt.

Effizienz	***Sicherheit***	***Zuverlässigkeit***	***Wartbarkeit***	***Portabilität***
-	+	+	+	+

Tab. 7.17: Übersicht über die Qualitätseigenschaften

7.3.2 Flexible, veränderbare und effiziente Komponentenstrukturen

Die zweite Kategorie umfasst Architekturstile, die durch eine lose Kopplung zwischen den Komponenten eine flexible und leicht veränderbare Komponentenstruktur ermöglichen. Hierzu zählen u. a. die Stile 'Pipes-and-Filters', 'Implicit-Invocation' und 'Blackboard' [Sha+96]. Der Stil 'Pipes-and-Filters' orientiert sich am Datenfluss, der über 'Pipes' zwischen verschiedenen 'Filtern' fließt. Die 'Filter' stellen hierbei Komponenten dar, welche die übertragenen Daten verarbeiten. Der Stil 'Implicit-Invocation' ist ein ereignisorientierter Stil, bei dem die Komponenten nicht über synchrone Botschaften, sondern über asynchrone Ereignisse (Events) kommunizieren. Durch den fehlenden starren Kontrollfluss wird eine flexible Zusammenarbeit der Komponenten ermöglicht. Der Stil 'Blackboard' ist ein datenzentrierter Stil. Ein oder mehrere Daten-Repositories stellen Informationen in Form eines 'Blackboards' zur Verfügung. Die an das 'Blackboard' angeschlossenen Komponenten entnehmen für sie bestimmten Daten, verarbeiten diese und geben die Ergebnisse an das 'Blackboard' zurück.

In Abb. 7.12 ist ein Beispiel für eine 'Pipes-and-Filters'-Architektur zu sehen. Drei Filter werden zur Komprimierung eines Videos nacheinander durchlaufen. Über 'Pipes' erfolgt die Übergabe der Videodaten zwischen den 'Filtern'. Der erste 'Filter' dient zur 'Komprimierung des Videobildes' mithilfe eines Codecs. Der zweite Filter komprimiert die 'Tonspur'. Der letzte 'Schärfefilter' dient zur Verbesserung des Videobildes. Da kein starrer Kontrollfluss festgelegt ist, kann der 'Schärfefilter' auch vor der 'Komprimierung der Tonspur' eingesetzt werden.

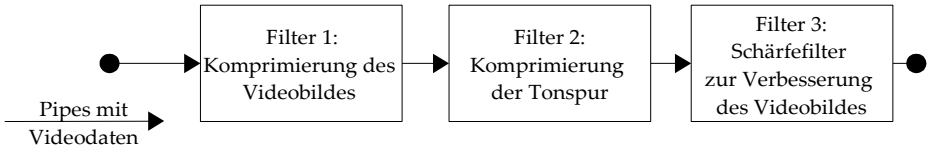


Abb. 7.12: Komprimierung eines Video-Datenstroms mit einer 'Pipes-and-Filters' Architektur

Bei diesen Architekturstilen wird auf die Strukturierung eines Kontroll- und Datenflusses zugunsten der Flexibilität und der Veränderbarkeit verzichtet. Aufgrund der Flexibilität können sehr effiziente Komponentenstrukturen aufgebaut werden. Gerade bei 'Pipes-and-Filters'-Architekturen besteht die Möglichkeit der Parallelisierung der Abläufe. In Bezug auf das vorhergehende Beispiel in Abb. 7.12 können die 'Komprimierung der Tonspur' und die 'Verbesserung der Schärfe' (Filter 2 und 3) auch parallel abgearbeitet werden. Es ist jedoch darauf zu achten, dass diese flexiblen Gestaltungsmöglichkeiten nicht zu einer unstrukturierten Verarbeitung der Daten bei ineffizienter Nutzung von Speicher- und Prozessorressourcen führen. Die Implementierung geeigneter Mechanismen zur Verbesserung der Zuverlässigkeit und Wahrung von Sicherheitsaspekten wird durch die lose Kopplung der Komponenten zusätzlich erschwert, wodurch negative Auswirkungen auf diese Qualitätsmerkmale zu erwarten sind. Durch diese Architekturstile wird die Portabilität jedoch nicht direkt beeinflusst. Eine Übersicht über die Qualitätseigenschaften wird in Tabelle Tab. 7.18 (im folgenden Abschnitt) gegeben.

7.3.3 Zusammenfassung

Die Qualitätseigenschaften der beiden Architekturstilkategorien sind in Tab. 7.18 zusammenfassend dargestellt.

Kategorie	Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
Wartbare Komponentenstrukturen mit klar definierten Zugriffsvarianten	-	+	+	+	+
Flexible, veränderbare und effiziente Komponentenstrukturen	+	-	-	+	o

Tab. 7.18: Überblick über die Qualitätseigenschaften der Architekturstilkategorien

7.4 Softwareprodukte und Technologien

Neben dokumentiertem Erfahrungswissen in Form von Entwurfs- und Architekturmustern sowie Architekturstilen stehen gemäß der Argumentation in Kapitel 2.6 Technologien und Softwareprodukte als mögliche Lösungsansätze für Architekturentscheidungen zur Verfügung. Während gerade bei den Technologien und Softwareprodukten die funktionalen Eigenschaften im Detail ermittelt werden können, existieren große Unsicherheiten über die nicht-funktionalen Qualitätseigenschaften. Um eine systematische Vorauswahl zu ermöglichen (siehe Abschnitt 5.3.2, S. 70), ist daher analog zu den Mustern und Stilen eine Betrachtung der Qualitätseigenschaften erforderlich. Aufgrund der Vielzahl an einzelnen Technologien und Softwareprodukten kann jedoch nur ein grober sowie generischer Überblick über die Qualitätseigenschaften gegeben werden; im tatsächlichen Entscheidungsfall ist daher jeweils zu überprüfen, ob die in den folgenden Abschnitten ermittelten Qualitätseigenschaften abweichen. Entsprechend der Argumentation in Abschnitt 2.7 (ab S. 35ff.) erfolgt die Untersuchung nach den folgenden von Posch aufgeführten Kategorien 'Betriebssysteme und Programmiersprachen', 'Bibliotheken', 'Komponententechnologien' und 'Frameworks' [Pos+04].

7.4.1 Betriebssysteme

Das Betriebssystem stellt die Schnittstelle des Softwaresystems zu den verfügbaren Hard- und Softwareressourcen dar und koordiniert die Prozessor- und Speicherverwaltung [Pos+04]. Zur Auswahl steht eine Vielzahl von Betriebssystemen: 'Microsoft-Windows-Systeme', viele 'Linux'- und 'Unix-Derivate' oder auch 'VxWorks' oder 'QNX' als spezialisierte Echtzeit-Betriebssysteme. Die Wahl des Betriebssystems hängt einerseits von der Zielplattform ab. Bei eingebetteten Systemen ist u. U. gar kein Betriebssystem notwendig. Im Gegensatz dazu können im PC- und im Mainframe-Bereich mehrere Betriebssysteme gleichzeitig einsetzbar sein. Andererseits ist zu beachten, dass von der Auswahl eines Betriebssystems die im Softwareentwicklungsprojekt verwendbaren Technologien und Werkzeuge abhängen. Durch den zunehmenden Einsatz plattformübergreifender Betriebssysteme nimmt dieser Aspekt jedoch eine untergeordnete Bedeutung ein.

In Anlehnung an die Architekturmuster-Kategorie 'Effizientes Prozess- und Threadmanagement' (siehe Kapitel 7.2.1 auf S. 145) ist die Auswahl eines Betriebssystems eine kritische Entscheidung, da das Betriebssystem die Laufzeitumgebung des Softwaresystems zur Verfügung stellt. Fehlkonfigurationen können den Ausfall von Teilen oder des gesamten Softwaresystems zur Folge haben. Durch den Einsatz eines geeigneten Betriebssystems kann jedoch speziell die Zuverlässigkeit, Effizienz, Portabilität oder Sicherheit eines Softwaresystems gezielt beeinflusst werden. Entsprechend den Erfahrungen aus der Praxis stehen die Qualitätseigenschaften jedoch häufig in Konflikt zueinander. Es steht zwar z. B. mit 'QNX' ein effizientes Echtzeit-Betriebssystem zur Verfügung; aufgrund des geringen Verbreitungsgrades wird dadurch jedoch die Portabilität des Softwaresystems vermindert [Sas+95]. Die Eignung bestimmter Betriebssysteme ist daher insbesondere im Rahmen des Rankings der Lösungsansätze (siehe Abschnitt 5.3.3, S. 70ff.) zu konkretisieren und zu verfeinern. Da Aussagen über die Qualitätseigenschaften einzelner Betriebssysteme stark von den Zielen und Rahmenbedingungen des Softwareentwicklungsprojektes abhängen, können keine allgemeingültigen Aussagen über die Qualitätseigenschaften im Hinblick auf die ISO-Norm 9126 getroffen werden.

7.4.2 Programmiersprachen

Programmiersprachen lassen sich nach ihren Eigenschaften klassifizieren; hierzu zählen prozedurale (z. B. 'Cobol'), objektorientierte (z. B. 'Java', 'Smalltalk', 'C++' oder 'C#'), deklarative und funktionale Programmiersprachen [Pos+04]. Darüber hinaus existieren u. a. auch Datenbanksprachen (z. B. 'SQL'). Durch den Einsatz einer bestimmten Programmiersprache können einzelne Qualitätsmerkmale gezielt verbessert werden, wie z. B. die Portabilität durch den Einsatz von 'Java' oder die Zuverlässigkeit durch den Einsatz von 'Cobol'. Gerade bei großen und über Jahre gewachsenen Softwaresystemen (sog. Legacy-System) kommen meist mehrere Programmiersprachen gleichzeitig zum Einsatz. Jedoch ist der Wechsel einer Pro-

grammiersprache bei einem Legacy-System meist nur mit enormem Aufwand möglich, der in der Höhe mit einer völligen Neuimplementierung der Anwendung vergleichbar ist. Außerdem muss berücksichtigt werden, dass der Einsatz einer Programmiersprache andere Qualitätsmerkmale evtl. negativ beeinflusst. Beispielsweise kann Java im Vergleich zu anderen Sprachen nur indirekt über den Interpreter auf das Prozessorregister zugreifen. Damit können Einschränkungen bei der Effizienz verbunden sein. Wie auch bei den Betriebssystemen (siehe vorangegangener Abschnitt 7.4.1) ist die Eignung einzelner Sprachen erst im Zusammenhang mit einer detaillierten Analyse, die in der zweiten Stufe der Vorauswahl erfolgt (siehe Abschnitt 5.3.3, S. 70ff.), abzuschätzen. Allgemeingültige Aussagen über die Qualitätseigenschaften im Hinblick auf die Qualitätsmerkmale der ISO-Norm 9126 können daher nicht getroffen werden.

7.4.3 Bibliotheken, Komponenten und Frameworks

In Bibliotheken sind zusammengehörige Funktionen oder vordefinierte Klassen in einer bestimmten Programmiersprache abgelegt [Pos+04]. Ein Beispiel ist die 'Standard-Template-Library' (STL), welche u. a. Iteratorklassen für die Sprache 'C++' zur Verfügung stellt. Bibliotheken sind daher Container, um häufig verwendete Standardfunktionen strukturiert zusammenzufassen und wiederzuverwenden. Erneute Implementierungen dieser Standard-Funktionen können somit vermieden werden. Dies erleichtert die Fehleranalyse bei Wartungs- und Veränderungsaktivitäten.

Neben Bibliotheken können Komponententechnologien eingesetzt werden, um die Funktionalität wiederzuverwenden. Komponenten sind flexibel einsetzbare Softwareprodukte, die eine spezielle Laufzeitumgebung erfordern [Pos+04]. Als Beispiele sind das 'Component-Object-Model' (COM), das 'Corba'- oder das '.NET'-Komponentenmodell zu nennen. Im Gegensatz zu einer Bibliothek ist die Wiederverwendung von Komponenten nicht von der Programmiersprache abhängig, in der die Funktionalität der Komponente implementiert wurde. Mit fast jeder verfügbaren Programmiersprache können Komponenten angesprochen und deren Funktionalität benutzt werden.

Die dritte Möglichkeit zur Wiederverwendung von Funktionalität stellen Frameworks dar, bei denen zwischen fachlichen und technischen Frameworks unterschieden wird [Pos+04]. Für Architekturentscheidungen sind speziell die technischen Frameworks relevant, welche u. a. Funktionen zur Kommunikation zwischen Objekten oder Komponenten, zur persistenten Datenspeicherung oder zur Verwaltung von Benutzeroberflächen beinhalten. Ein häufig genutztes technisches Framework stellt 'Hibernate' dar, welches einerseits einen transparenten Datenbankzugriff für die Programmiersprache 'Java' bietet und andererseits als Mapper zwischen Objekten und der relationalen Datenbankstruktur eingesetzt werden kann. Die Funktionalität zur Generierung von 'SQL-Statements', zur Abfrage von Datenbanken und das Verfahren zum Mapping der unterschiedlichen Datenstrukturen können somit leicht wiederverwendet werden.

Durch den Einsatz von Bibliotheken, Komponenten und Frameworks wird die Wiederverwendung von Funktionalität erleichtert. Dies verbessert zum einen die Wartbarkeit, da fehleranfällige Individualentwicklungen vermieden werden. Zum anderen wird die Portabilität erhöht, da spezielle Bibliotheken, Komponenten oder Frameworks entwickelt und je nach Systemumgebungen eingesetzt werden können. Die Effizienz, Sicherheit und Zuverlässigkeit werden nicht direkt beeinflusst; im tatsächlichen Entscheidungsfall ist die Eignung bestimmter Bibliotheken, Komponenten und Frameworks jedoch insbesondere im Rahmen des Rankings der Lösungsansätze (siehe Abschnitt 5.3.3, S. 70ff.) zu konkretisieren und zu verfeinern. Eine Übersicht über die Qualitätseigenschaften ist in Tab. 7.19 abgebildet.

Effizienz	Sicherheit	Zuverlässigkeit	Wartbarkeit	Portabilität
o	o	o	+	+

Tab. 7.19: Qualitätseigenschaften der Kategorie Bibliotheken, Komponenten und Frameworks

Kapitel 8

Evaluation des entwickelten Entscheidungsprozesses

8.1 Reduzierung von Komplexität, Risiken und Unsicherheiten

Komplexitätsreduzierung durch Dekomposition des Entscheidungsproblems

Zur Reduzierung der kombinatorischen Komplexität sowie zur Erreichung einer systematischen und nachvollziehbaren Entscheidungsfindung erfolgt eine Dekomposition des Entscheidungsproblems in seine Bestandteile. Gerade bei Architekturentscheidungen bei großen und über Jahre gewachsenen Softwaresystemen (sog. Legacy-Systeme) existieren vielzählige sowie zum Teil unentdeckte Abhängigkeiten zwischen den Zielen, Bedingungen und Lösungsansätzen. Um die daraus resultierende kombinatorische Komplexität zu reduzieren und eine systematische Entscheidungsfindung zu ermöglichen, ist es erforderlich, die Bestandteile der Architekturentscheidung zunächst isoliert zu untersuchen. So werden in den ersten beiden Phasen des Entscheidungsprozesses (siehe Abb. 3.1 auf S. 44) zunächst die Entscheidungsgrundlagen Ziele, Rahmenbedingungen sowie Architekturmodell isoliert betrachtet und untersucht. Erst im Anschluss werden geeignete alternative Lösungsansätze ermittelt, ausgewählt, verfeinert und konkretisiert (siehe die Darstellung in Abb. 8.1). Die Dekomposition des Entscheidungsproblems ist eine Weiterentwicklung des generischen Vorgehens aus der Entscheidungstheorie (siehe Abschnitt 2.1, S. 23f.), welches um spezifisch für Architekturentscheidungen benötigte Methoden und Konzepte ergänzt wurde.

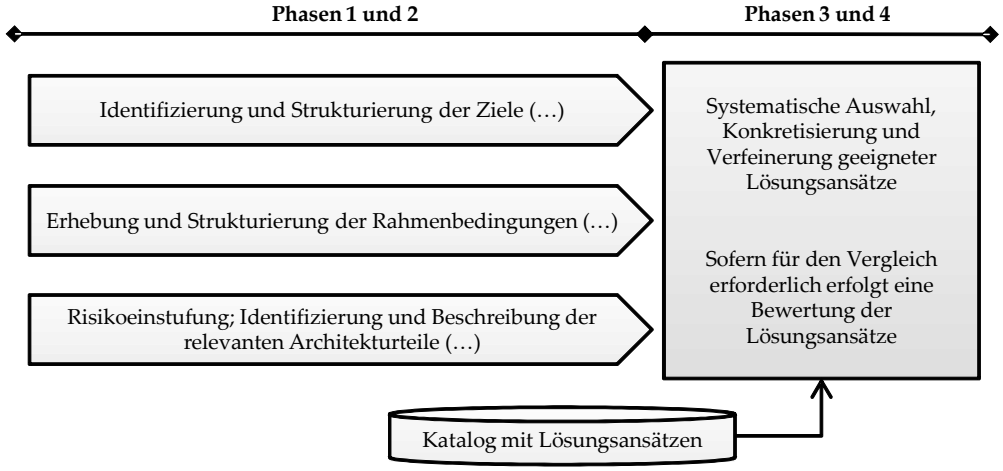


Abb. 8.1: Dekomposition des Entscheidungsproblems zur systematischen Entscheidungsfindung

Durch die Dekomposition und die sequenzielle Durchführung der Entscheidungsfindung wird es möglich, Abhängigkeitsbeziehungen systematisch zu identifizieren, zu berücksichtigen und dadurch Risiken sowie Unsicherheiten zu reduzieren. So können u. a. Ziel-Mittel-Beziehungen und Ziel-Konflikte können unabhängig von den Rahmenbedingungen oder Einschränkungen aus der existierenden Architektur identifiziert und aufgelöst werden; auch Konflikte zwischen den Rahmenbedingungen können ohne Rücksicht auf die Ziele aufgelöst werden. Erst bei der Auswahl, Verfeinerung und Konkretisierung der Lösungsansätze werden die zunächst isolierten Bestandteile in Kombination betrachtet. So wird beispielsweise im Rahmen der Typo3-Architekturentscheidung klar und eindeutig beschrieben, warum die Portabilität und die Zugriffszeiten die wichtigsten Ziele der Architekturentscheidung sind und welche Instrumentalziele dafür zu erfüllen sind (siehe Abb. 6.3, S. 94). Abhängigkeiten zu Terminen und zeitlichen Vorgaben sowie zu technischen Einschränkungen, beispiels-

weise aufgrund der Vielzahl an Typo3-Extensions, die nicht verändert werden können, werden bei der Aufstellung des Ziel-Wirkungsmodells zur Reduzierung der Komplexität ausgeblendet.

Die isolierte Betrachtung der Bestandteile der Entscheidung kann jedoch auch zur Folge haben, dass die Ziele mit den verfügbaren organisatorischen und technischen Rahmenbedingungen nicht erfüllt werden können. Andererseits können drängende prekäre Architekturprobleme dazu führen, dass zuerst die kritischen Probleme behoben werden müssen, bevor mit der Umsetzung der Ziele begonnen werden kann. In diesen Fällen ist ein Rücksprung mit entsprechender Anpassung der betreffenden Phase des Entscheidungsprozesses erforderlich. Mit diesen nachträglichen Anpassungen und Korrekturen ist ein Mehraufwand verbunden. Die Erfahrungen aus der Praxis zeigen jedoch, dass gravierende nachträgliche Korrekturen eher selten sind.

Reduzierung von Unsicherheiten bei der Auswahl und Verfeinerung von Lösungsansätzen

Entsprechend der Ausführungen im Rahmen der Analysen zum Stand der Technik (siehe insbesondere Abschnitt 2.1, S. 23f.) ist das generische Vorgehen aus der Entscheidungstheorie um Konzepte und Methoden zu ergänzen, die eine Analyse und Berücksichtigung der softwaretechnischen Details der Architektur und der Lösungsansätze in der erforderlichen Tiefe ermöglichen. Gerade bei der Auswahl, der Verfeinerung und der Konkretisierung der Lösungsalternativen sind die technischen Eigenschaften der Architektur und der Lösungsansätze zu berücksichtigen, jedoch ist aus Aufwandsgründen nur bei sehr riskanten Entscheidungen (siehe für die Risikoeinstufung Abschnitt 5.2.1, S. 63) eine prototypische Implementierung gerechtfertigt. Um die Unsicherheiten bei der Auswahl und Verfeinerung der Lösungsansätze auch ohne eine prototypische Implementierung reduzieren zu können, wurde das generische Vorgehen aus der Entscheidungstheorie daher um die folgenden Beiträge ergänzt.

Zur Unterstützung der Identifizierung der geeigneten Lösungsansätze wurde ein Katalog entwickelt, der die wichtigsten Entwurfs- und Architekturmuster, Stile sowie Technologien und Softwareprodukte beinhaltet und diese anhand ähnlicher Qualitätseigenschaften kategorisiert. Der Katalog ermöglicht dem Entscheidungsträger eine systematische Auswahlentscheidung auch bei nicht-funktionalen Qualitätszielen, bei denen in der Praxis oftmals Unsicherheiten und Widersprüche bei der Zerlegung und Verfeinerung auftreten (siehe hierzu die Argumentation in Abschnitt 2.3, S. 26ff.). Da der Katalog die Entwurfs- und Architekturmuster, Stile sowie Technologien und Softwareprodukte ausschließlich nach Qualitätseigenschaften klassifiziert, sind bei der Auswahlentscheidung weitere Untersuchungen erforderlich, um die organisatorischen und technischen Rahmenbedingungen zu berücksichtigen. Die Eignung bestimmter Lösungsansätze ist daher insbesondere im Rahmen des Rankings der Lösungsansätze (siehe Abschnitt 5.3.3, S. 70ff.) zu konkretisieren und zu verfeinern.

Da aus Aufwandsgründen nur bei sehr riskanten Architekturentscheidungen der Klasse 2 (siehe für die Risikoeinstufung S. 63) prototypische Implementierungen gerechtfertigt sind, erfolgen die Analyse und Bewertung von Lösungsansätzen anhand von Architekturprototypen. Dabei wird ausgehend vom Modell der existierenden Architektur ein Modell der zukünftigen Architektur entwickelt und mittels szenariobasierter Analysetechniken untersucht. Auf dieser Grundlage kann grob abgeschätzt werden, ob der Lösungsansatz die Schwachstellen und Problembereiche behebt, welche Abhängigkeiten dabei berücksichtigt werden müssen und ob die Rahmenbedingungen eingehalten werden. Somit können die Eigenschaften von Lösungsansätzen sowie Seiteneffekte und Wechselwirkungen mit – im Vergleich zur prototypischen Implementierung – geringem Aufwand analysiert werden. Dies zeigt sich u. a. bei der Typo3-Architekturveränderung, bei der schon bei der Erstellung der Architekturprototypen für die ADOdb-Alternative erkannt wurde, dass sich die Zugriffszeiten verschlechtern würden (siehe Abschnitt 6.1.3 auf S. 107f.). Die Qualität der szenariobasierten Analysen hängt jedoch stark von der Qualität der Szenarien ab. Es ist ein großes Maß an Erfahrung erforderlich, um genau jene Szenarien auswählen zu können, die objektive und valide Aussagen über die Eigenschaften der Lösungsansätze ermöglichen. Zwar steht zur Erleichterung dieser Auswahl mit der Analysemethode ALMA ein Katalog mit Veränderungsszenarien zur Verfügung (siehe Abschnitt 2.4 auf S. 29 und Tab. 10.3 auf S. 175f.); trotzdem verbleibt ein Risiko bei der Auswahl der *richtigen* Szenarien.

Senkung des Risikos von Fehleinschätzungen durch Einhaltung der Rationalitätskriterien

Ein wichtiger Beitrag zur Senkung des Risikos von Fehleinschätzungen und -entscheidungen sowie den daraus resultierenden negativen Konsequenzen ist die Einhaltung der Rationalitätskriterien bei jeder der vier Phasen des Entscheidungsprozesses. Die Rationalitätskriterien (siehe Abschnitt 2.1, S. 22) schreiben ein hohes Maß an Objektivität und Validität bei der Entscheidungsfindung vor. So müssen die Entscheidungsgrundlagen (Ziele, Rahmenbedingungen etc.) vollständig und konsistent identifiziert und erhoben sein. Bei der Durchführung der Entscheidung (Auswahl, Konkretisierung und Bewertung der Lösungsansätze) ist die prozedurale Rationalität der Entscheidungsfindung einzuhalten, indem sich das Vorgehen an sachlichen Kriterien (Beitrag zur Zielerreichung, Höhe des Implementierungsaufwandes, Kompatibilität etc.) orientiert. Gerade im Hinblick auf die Einhaltung der Rationalität ist jedoch das Verhältnis zwischen Aufwand und Nutzen zu betrachten. Nicht jede Entscheidung rechtfertigt eine vollständige Einhaltung der Rationalität, da beispielsweise das mit der Entscheidung verbundene Risiko nicht hoch genug und ein gewisses Maß an Unsicherheit aus Aufwandsgründen gerechtfertigt ist. Wie ein ökonomisch sinnvolles Verhältnis zwischen Rationalität und Nutzen bei der Entscheidungsfindung erreicht werden kann, ist Gegenstand des folgenden Abschnitts 8.2 (S. 159ff.).

Zur vollständigen und widerspruchsfreien Identifizierung und Strukturierung der Entscheidungsgrundlagen sind die nachstehenden Maßnahmen durchzuführen:

- *Ziele:* Sofern die Ziele noch nicht vollständig bekannt sind, wird zur Identifizierung eine szenariobasierte Analyse vorgeschlagen (siehe die Ausführungen in Abschnitt 5.1.1, S. 56). Die Ziele sind zur Beschreibung von Ziel-Mittel-Beziehungen und zur Auflösung von Konflikten durch die Klassifizierung nach Instrumental- und Fundamentalzielen sowie durch die Erstellung des Ziel-Wirkungsmodells zu strukturieren.
- *Rahmenbedingungen:* Aufgrund der Vielzahl an potenziellen Konflikten und Widersprüchen zwischen den Rahmenbedingungen erfolgt die Erhebung anhand einer systematischen Checkliste. Widersprüche zwischen den Rahmenbedingungen aufgrund von Abhängigkeitsbeziehungen werden dadurch aufgelöst, indem die vom Entscheidungsträger beeinflussbaren Rahmenbedingungen entsprechend angepasst werden.
- *Modell der Architektur:* Das im Rahmen des Entscheidungsprozesses erstellte Modell der Architektur umfasst aus Aufwandsgründen nur einen Ausschnitt der real existierenden Teile des Softwaresystems. Dieser Ausschnitt muss zur Wahrung der Vollständigkeit alle entscheidungsrelevanten Architekturteile beinhalten und frei von Widersprüchen sein. Da zum Zeitpunkt der Architekturbeschreibung die Lösungsansätze noch nicht ausgewählt wurden, muss das Modell ergänzt werden, wenn bei der Auswahl und Konkretisierung der Lösungsansätze Lücken oder Inkonsistenzen erkannt werden. Die Rationalitätskriterien werden daher nur in Ausnahmefällen gleich bei der Beschreibung des Architekturmodells vollständig erfüllt; in der Regel steht erst nach Abschluss der dritten Phase, der Auswahl und Konkretisierung der Lösungsansätze, ein – für die aktuell betrachtete Entscheidung – vollständiges und konsistentes Architekturmodell zur Verfügung.

Nachdem die Entscheidungsgrundlagen vollständig und konsistent beschrieben wurden, dienen sie als Kriterien für die Auswahl, Verfeinerung und Konkretisierung der Lösungsansätze. Um auf dieser Entscheidungsgrundlage die sachlich-richtigen Einschätzungen über die Lösungsansätze vornehmen zu können, kann ein szenariobasiertes Analyseverfahren zur Verfeinerung und Konkretisierung der Lösungsansätze durchgeführt werden; die Grundlage für die Analyse ist die Erstellung von Architekturprototypen. Werden dabei Lücken oder Inkonsistenzen im bereits erstellten Modell der Architektur (siehe vorangegangenen Absatz) festgestellt, ist das Modell zu ergänzen oder zu korrigieren. Sofern erforderlich erfolgt im Anschluss eine Bewertung der Lösungsansätze, die sich ausschließlich am Grad der Zielerreichung der Lösungsansätze orientiert. Methodische Grundlage der Bewertung ist das aus der präskriptiven Entscheidungstheorie adaptierte Bewertungsverfahren für unsichere Entscheidungen bei mehreren Zielen (siehe hierzu Abschnitt 2.9, S. 38ff.). Somit erfüllt der Entscheidungsprozess die Anforderungen an die Rationalitätskriterien aus Abschnitt 2.1 (S. 22) .

In Abb. 8.2 ist dieser Zusammenhang dargestellt. Die zunächst isoliert voneinander vollständig und konsistent ermittelten Entscheidungsgrundlagen (Phase 1 und 2) dienen als sachliche Kriterien für die Auswahl, Konkretisierung und Bewertung der Lösungsansätze (Phase 3 und 4).

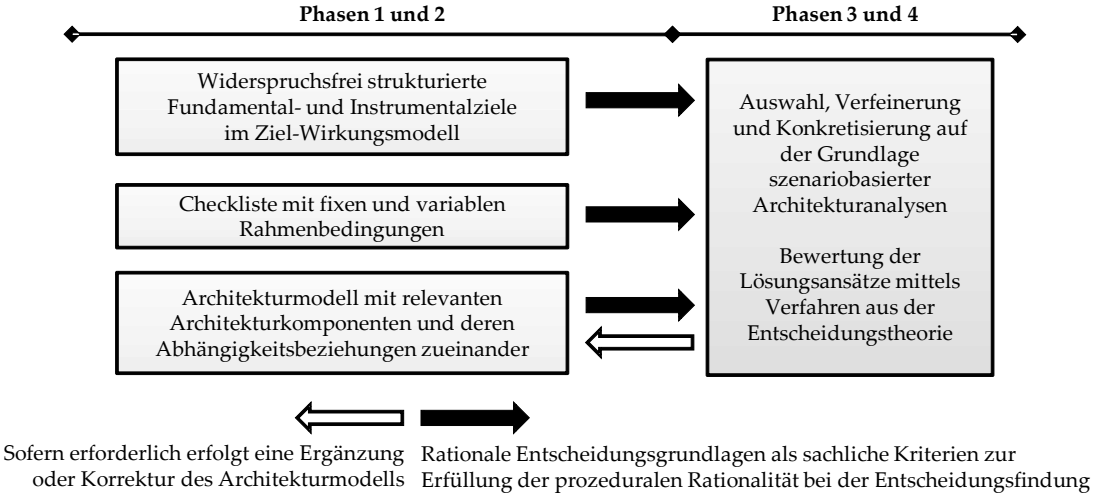


Abb. 8.2: Einhaltung der Rationalitätskriterien zur Senkung des Risikos von Fehlentscheidungen

Dass durch die Einhaltung der Rationalitätskriterien das Risiko von Fehleinschätzung gesenkt wird, kann anhand der Typo3-Architekturveränderung wie auch der 'Sicherheitsverbesserung des E-Banking-Systems' verdeutlicht werden. Bei der Typo3-Architekturveränderung konnte bereits während der Konkretisierung der Lösungsansätze aufgezeigt werden, dass die Performance bei der ADOdb-Alternative aufgrund der Transformationsschritte ein kritischer Punkt ist (siehe Abschnitt 6.1.3, S. 107f.); auf der anderen Seite konnte gezeigt werden, dass gerade die Transformationen ein wesentlicher Grund für die Flexibilität und die daraus resultierenden hohen Portabilitätsverbesserungen ist. Genau diese prognostizierten Konsequenzen, die auf der Grundlage einer szenariobasierten Analyse der Eigenschaften der Lösungsansätze, konnten unter realen Bedingungen bestätigt werden (siehe Abschnitt 6.1.5, S. 116f.). Analog dazu konnten die im Rahmen des E-Banking-Beispiels aufgezeigten negativen Auswirkungen der 'clientbasierten Generierung' der Transaktionsnummern mithilfe eines separaten Gerätes sowie die hohen Kosten im Vergleich zur serverbasierten Variante und die geringe Akzeptanz auf der Kundenseite anhand der realen Erfahrungen bei der BW-Bank vorhergesagt werden (siehe Abschnitt 6.2.5, S. 137).

Jedoch bleibt ein subjektiver Faktor bei der Ermittlung der Eigenschaften der Lösungsansätze, da mit der Auswertung der szenariobasierten Analyseergebnisse eine Vielzahl von Schätzungen verbunden sind. Nur eine prototypische Implementierung würde eine detaillierte Analyse und ausführliche Tests, gerade im Hinblick auf die funktionalen Eigenschaften, ermöglichen. Aus Aufwandsgründen ist die prototypische Implementierung jedoch nur bei sehr riskanten Entscheidungen gerechtfertigt (siehe Abschnitt 5.3.4.3 auf S. 77).

Beide Beispiele sind eher kleinere, prototypische Architekturentscheidungen, bei denen der Umfang der Architekturveränderung und die Höhe der Risiken überschaubar sind. Gerade die 'Sicherheitsverbesserung des E-Banking-Systems' wurde zudem mit geringer Detailtiefe beschrieben. Es ist daher eine Notwendigkeit, dass der Entscheidungsprozess in verschiedenen Domänen mit unterschiedlichen Komplexitäts- und Risikostufen eingesetzt wird, um den Entscheidungsprozess für Architekturentscheidungen weiterzuentwickeln.

8.2 Ökonomisch sinnvolles Aufwand-Nutzen-Verhältnis

Die Reduzierung von Komplexität, Unsicherheiten und Risiken muss in einem ökonomisch sinnvollen Verhältnis zum Aufwand stehen, der mit der Entscheidungsfindung verbunden ist. So ist es zwar aus Gründen der Nachvollzieh- und Validierbarkeit auch bei risikoarmen Architekturentscheidungen erforderlich, den Entscheidungsprozess anzuwenden. Dies gilt insbesondere dann, wenn die Architekturentscheidung in eine Kette aufeinander aufbauender Entscheidungen eingebunden ist. Bei risikoarmen Architekturentscheidungen wird jedoch nur ein Mindestmaß an Analyse-, Design- und Bewer-

tungstätigkeiten benötigt; umfangreiche und detaillierte Analysen oder eine prototypische Implementierung sind aufgrund des geringen Risikos nicht gerechtfertigt.

Um den Entscheidungsprozess an die Komplexität und das Risiko der Entscheidungssituation flexibel anpassen zu können, sind die Analyse-, Design- und Bewertungstätigkeiten jeder Phase in obligatorische und optionale Tätigkeiten unterteilt. Die 'Identifizierung und widerspruchsfreie Strukturierung der Ziele' ist beispielsweise eine obligatorische Tätigkeit, die bei jeder Entscheidung durchzuführen ist, da ohne sie insbesondere die Nachvollzieh- und Validierbarkeit der Entscheidung nicht mehr möglich ist (siehe Abschnitt 5.1.2 auf S. 57). Eine 'prototypische Modellveränderung' und die aufwendigen 'szenariobasierten Architekturanalysen' sind jedoch bei risikoarmen Entscheidungen der Klasse 0 auf ein Minimum zu reduzieren (siehe Abschnitt 5.3.3 auf S. 71f., siehe für die Risikoeinstufung S. 63). In Abschnitt 3.3 (siehe S. 48) wird eine Tabelle eingeführt, die alle relevanten Tätigkeiten zur Entscheidungsfindung umfasst und in obligatorische und optionale Tätigkeiten unterteilt (siehe für die vollständige Tabelle Tab. 10.2 auf S. 171ff.). In dieser ist jeweils aufgeführt und begründet, warum eine Tätigkeit obligatorisch ist und anhand welcher Kriterien ermittelt werden kann, wann die Anwendung einer optionalen Tätigkeit gerechtfertigt ist. Dadurch wird eine Anpassbarkeit des Entscheidungsprozesses an komplexe und riskante, aber auch an risikoarme Entscheidungssituation ermöglicht. In Tab. 8.1 sind die wenigen obligatorischen Tätigkeiten aufgezeigt, die ein Mindestmaß an rationaler Entscheidungsfindung auch bei Entscheidungen mit geringer Unsicherheit, Komplexität sowie geringen Risiken gewährleisten. Diese obligatorischen Tätigkeiten können um die in Tab. 10.2 aufgeführten optionalen Tätigkeiten ergänzt werden.

Nummer	Bezeichnung der obligatorischen Tätigkeiten
Phase 1	
5.1.1	**Identifizierung der Ziele auf der Grundlage von Schwachstellen und Problembereichen**
5.1.3	**Erhebung der organisatorischen und technischen Rahmenbedingungen**
Phase 2	
5.2.1	**Einstufung des Risikos der Entscheidung**
5.2.2	**Beschreibung der Architektur**
Phase 3	
5.3.4	**Konkretisierung und Anpassung der Lösungsansätze**
5.3.4.1	**Aufstellung und Verfeinerung der Maßnahmenfolge**
5.3.4.2	**Auswertung der Abhängigkeitsbeziehungen**
5.3.4.3	**Ermittlung der Eigenschaften der Lösungsansätze**
Phase 4	
5.4.2	**Treffen der Entscheidung**

Tab. 8.1: Mindestumfang des Entscheidungsprozesses

Die Abwägung, welche der Kriterien erfüllt sind, die die Durchführung einer optionalen Tätigkeit rechtfertigen, obliegt dem Entscheidungsträger und dessen Erfahrung mit vergleichbaren Architekturentscheidungen. Das wichtigste Kriterium ist dabei das Risiko, welches aus verschiedenen Blickwinkeln betrachtet werden kann und jeweils zu unterschiedlichen Interpretationen führt. Um eine falsche Einschätzung des mit der Entscheidung verbundenen Risikos zu vermeiden und die Risikoeinschätzung transparenter sowie nachvollziehbarer zu machen, erfolgt zu Beginn der dritten Phase eine Klassifizierung des Risikos der Entscheidung. Das Risiko wird hierbei unter dem Gesichtspunkt der Kritikalität einer Entscheidung für ein geschäftskritisches Softwaresystem betrachtet; je nachdem ob eine Entscheidung sich auf ein geschäftskritisches Softwaresystem bezieht und ob ein Rücksprung oder Wechsel zu einem lauffähigen System kurzfristig möglich ist. Anhand dieser Kriterien kann der Entscheidungsträger transparent und nachvollziehbar bestimmen, welches Risiko mit der aktuell betrachteten Entscheidung verbunden ist und welche der obligatorischen Tätigkeiten angewendet werden sollten.

Jedoch obliegt die Entscheidung, welche optionalen Tätigkeiten angewendet werden, dem Entscheidungsträger. Sofern dieser die Komplexität und das Risiko der Architekturentscheidung aufgrund mangelnder Erfahrungen falsch einschätzt, wird ein ökonomisch sinnvolles Verhältnis zwischen dem Aufwand zur Entscheidungsfindung und dem daraus resultierenden Nutzen nicht erreicht.

8.3 Dokumenation und Nachvollziehbarkeit

Der Entscheidungsprozess sowie die Art und Weise der Entscheidungsfindung müssen dokumentiert werden, um nachvollziehen zu können, unter welchen Bedingungen und Annahmen die Architekturentscheidung getroffen wurde. Die Erfahrungen aus der Praxis zeigen, dass sich die Bedingungen und Annahmen im Laufe der Implementierung verändern und der Entscheidungsträger einem immer stärkeren Rechfertigungsdruck ausgesetzt ist. Neben den Aspekten der Rechtfertigung stellt die Dokumentation die Grundlage für die Validierbarkeit der Entscheidung dar, um im Anschluss an die Implementierung einer Lösungsalternative ermitteln zu können, ob und in welchem Umfang die während der Entscheidungsfindung ermittelten Eigenschaften, Seiteneffekte etc. mit den tatsächlich eingetretenen Konsequenzen übereinstimmen. Dadurch kann die Gefahr von Schätz- und Analysefehlern bei zukünftigen Entscheidungen reduziert werden.

Ein zentrales Element zur Dokumentation der Entscheidungsfindung und zur Erfüllung der Nachvollziehbarkeit ist der Entscheidungsbaum, der im Rahmen der Verfeinerung und Konkretisierung der alternativen Lösungsansätze aufgestellt wird und die wichtigsten Annahmen sowie entscheidungsrelevanten Informationen umfasst:

- Der Entscheidungsbaum beinhaltet eine detaillierte Maßnahmenfolge, die den Prozess der Implementierung beschreibt. Da die Beschreibung in Form eines standardisierten UML-Aktivitätsdiagramms erfolgt, ist die Nachvollziehbarkeit leichter möglich als bei einer groben und informalen Beschreibung.
- Die Eigenschaftswerte der im Detail betrachteten Lösungsansätze werden in Form von ordinal oder kardinal skalierten Werten am Ende jedes Astes des Entscheidungsbaumes dargestellt. Zu den Eigenschaften zählen insbesondere der Grad der Zielerreichung, der Implementierungsaufwand sowie die Einhaltung der organisatorischen und technischen Rahmenbedingungen.
- Sind bei der Verfeinerung und Konkretisierung der Lösungsansätze Varianten mit unterschiedlichen Wahrscheinlichkeiten zu berücksichtigen, werden diese als separate Äste im Entscheidungsbaum abgebildet. Die Verfeinerung und Konkretisierung der ADOdb-Alternative im Rahmen der Typo3-Architekturveränderung umfasst zwei Varianten 'mit und ohne Probleme bei den Zugriffszeiten', die im Entscheidungsbaum entsprechend abgebildet wurden (siehe Abschnitt 6.1.3, Abb. 6.11 auf S. 107). Die Typo3-Architekturveränderung zeigt ebenfalls, dass die im Entscheidungsbaum abgebildeten Informationen und kausalen Zusammenhänge dazu genutzt werden können, die im Rahmen der Entscheidungsfindung getroffenen Annahmen und Bewertungen anhand der in der Praxis tatsächlich realisierten Ergebnisse überprüfen und validieren zu können. Gerade im Hinblick auf die 'ADOdb-Alternative' decken sich die im Rahmen des Entscheidungsprozesses eingesetzten Annahmen, Schlussfolgerungen und Bewertungen mit den Erfahrungen aus der Praxis (siehe Abschnitt 6.1.5, S. 116f.).

Neben dem Entscheidungsbaum sind das Ziel-Wirkungsmodell, die Architekturprototypen und die Ergebnisse der Szenarioanalysen wichtige Dokumente für die Nachvollziehbarkeit und Validierbarkeit der Entscheidungsfindung. Dabei muss jedoch berücksichtigt werden, dass die Erstellung von Architekturprototypen und deren szenariobasierte Analyse gerade bei risikoarmen Entscheidungen aus Aufwand-Nutzen-Gründen nur selten gerechtfertigt ist.

Die ersten Anwendungen des Entscheidungsprozesses sowie Diskussionen mit Architekten und Projektleitern von Großprojekten zeigten, dass die Dokumentation und Nachvollziehbarkeit für den Entscheidungsträger oftmals noch wichtiger sind, als die Reduzierung von Risiken, Komplexität und Unsicherheiten. Die Ziele und vor allem die organisatorischen und technischen Rahmenbedingungen verändern sich im Laufe der Zeit; damit verändert sich auch der Grad der Zielerreichung und der Implementierungsaufwand der im Rahmen der Entscheidungsfindung betrachteten Lösungsansätze. Es hat sich

gezeigt, dass der Entscheidungsträger einem immer stärkeren Rechtfertigungsdruck ausgesetzt ist und dass die im Rahmen des Entscheidungsverlaufs erstellten Dokumente eine wichtige Grundlage für die Begründung und Nachvollziehbarkeit einer Architekturentscheidung sind.

8.4 Eignung für architekturorientiertes Refactoring

Das Ziel des Entscheidungsprozesses ist die Unterstützung von Architekturentscheidungen bei der Neuentwicklung eines Softwaresystems sowie bei der Architekturveränderung existierender Systeme. Das Refactoring stellt als spezielle Architekturveränderung jedoch spezifische Anforderungen, die im Entscheidungsprozess zu berücksichtigen sind. Die wichtigste Anforderung ist die Vermeidung funktionaler Veränderungen an den Eigenschaften des Softwaresystems. Nach einem durchgeführten Refactoring sollen mit identischen Eingangswerten dieselben Ergebnisse erzielt werden können [Fowl05], um u. a. den Testaufwand verringern sowie Wechselwirkungen mit unveränderten Komponenten des Softwaresystems vermeiden zu können. Eine weitere spezifische Anforderung ist die Forderung nach kleinen Refactoringschritten. Komplexe architekturorientierte Refactorings, die weite Teile des Softwaresystems betreffen, sind in klar abgegrenzten kleineren Teilschritten durchzuführen. Das Ziel ist die Ermöglichung eines Rücksprunges (ein sog. Undo) auf eine unveränderte Version des Softwaresystems, um z. B. im Falle unerwarteter Veränderungen am funktionalen Verhalten einer zentralen und kritischen Komponente diese zeitnah austauschen zu können.

Im Falle eines architekturorientierten Refactorings erfolgt im Rahmen der Verfeinerung und Konkretisierung der Lösungsansätze eine Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften (siehe Abschnitt 5.3.4.4, S. 79f.). Dabei werden die folgenden Stufen durchlaufen:

- Zunächst wird anhand eines Architekturprototyps überprüft, ob und in welchem Umfang die funktionalen Anforderungen erfüllt werden können, die von der Architekturveränderung betroffen sind. Zum Einsatz kommen dabei szenariobasierte Architekturanalysen.
- Da das prototypische Modell nur einen Ausschnitt der zukünftigen Architektur beschreibt, ist bei Unklarheiten im Rahmen der Analyse der Betrachtungsraum zu erweitern.
- Können auch auf dieser Basis keine sicheren Erkenntnisse über die Erfüllung der relevanten funktionalen Anforderungen getroffen werden, ist aus Risikoerwägungen heraus eine prototypische Implementierung der betreffenden Lösungsansätze notwendig, die durch die Anwendung geeigneter Testfälle hinsichtlich des funktionalen Verhaltens überprüft wird. Gerade bei diesem Schritt ist jedoch der Mehraufwand zur prototypischen Implementierung im Hinblick auf den Nutzen kritisch zu hinterfragen; es wird daher empfohlen, die prototypische Implementierung nur bei sehr riskanten Entscheidungen der Klasse 2 durchzuführen. So wird beispielsweise im Rahmen der Typo3-Architekturveränderung (Risikoklasse 1) nur anhand der prototypischen Architekturmodelle (Abb. 6.7 und Abb. 6.8, siehe S. 102ff.) überprüft, ob und welche Veränderungen am funktionalen Verhalten entstehen.

Im Falle von erkannten Veränderungen stehen dem Entscheidungsträger zwei Möglichkeiten der Reaktion zur Verfügung. Für kleinere Veränderungen, die durch geeignete Gegenmaßnahmen im Rahmen der Implementierung des Lösungsansatzes abgefangen werden können, z. B. indem die betreffende Schnittstelle speziellen Tests unterworfen wird, sind die entsprechenden Konzepte, Testfälle oder Gegenmaßnahmen zu erstellen. Bei größeren Veränderungen, die strukturelle Ursachen haben, ist hingegen eine Ergänzung oder Korrektur der Maßnahmenfolge erforderlich.

Wird erkannt, dass das geplante architekturorientierte Refactoring in einem komplexen und weit reichenden Refactoringschritt mündet, stehen dem Entscheidungsträger zwei Möglichkeiten zur Verfügung, um das Refactoring dennoch in klar abgegrenzten kleineren Teilschritten durchzuführen. Zunächst kann eine sinnvolle Einschränkung der Ziele auf zwei bis drei Fundamentalziele zu einer Reduzierung der kombinatorischen Komplexität führen und damit kleinere, klar abgegrenzte Refactoringschritte ermöglichen (siehe Abschnitt 5.1.2 auf S. 59). Die zweite Möglichkeit ist die Aufteilung der Architekturentscheidung in mehrere Stufen. Dabei wird auf der Grundlage von Lösungsszenarien ermittelt, in welcher Art und Weise die Architektur weiterentwickelt und verändert wird und in wie viele Stufen die aktuell betrachtete Entschei-

dung aufgeteilt werden sollte. Auf jeder Stufe erfolgt eine Auswahl und Verfeinerung von Lösungsansätzen, bei der jedoch Abhängigkeiten zu den anderen Stufen berücksichtigt werden müssen. Dass ein komplexes Architekturrefactoring in kleinere Einzelteile zerlegt werden kann, zeigt das Beispiel der 'Sicherheitsverbesserungen des E-Banking-Systems'. In drei klar voneinander abgegrenzten Stufen werden die relevanten Lösungsansätze ausgewählt, verfeinert und es erfolgt eine grobe Überprüfung der Veränderungen am funktionalen Verhalten.

Die Ausführungen zeigen, dass mit dem entwickelten Entscheidungsprozess architekturorientierte Refactorings zwar prinzipiell geplant und durchgeführt werden können. Gerade bei architekturorientiertem Refactoring tritt jedoch der Konflikt zwischen Aufwand und Nutzen offen zutage. Für eine detaillierte Überprüfung auf Veränderungen am funktionalen Verhalten ist eine prototypische Implementierung erforderlich. Gerade bei risikoarmen Entscheidungen steht der Aufwand jedoch nicht immer in einem ökonomisch sinnvollen Verhältnis zum Nutzen. Zudem zeigen die Erfahrungen in der Praxis, dass aus Zeitknappheit oftmals keine Ressourcen für die Implementierung eines Prototypen zur Verfügung stehen. Im Rahmen der Entwicklung des Entscheidungsprozesses wurde mit der szenariobasierten Analyse der Architekturprotypen ein Kompromiss vorgeschlagen. Es verbleiben jedoch Unsicherheiten über Veränderungen am funktionalen Verhalten, da die Szenarien keine detaillierten Untersuchungen der funktionalen Anforderungen ermöglichen. Sofern identische funktionale Systemeigenschaften notwendig sind, ist kritisch zu überprüfen, in welchem Umfang eine prototypische Implementierung durchgeführt werden kann.

Kapitel 9

Schlussfolgerungen

9.1 Beitrag

Da die existierenden Methoden zur Entscheidungsunterstützung entweder zu feingranular oder – wie im Falle der Entscheidungstheorie – zu generisch sind, wurde ein Entscheidungsprozess entwickelt, der an die spezifischen Eigenschaften von Architekturentscheidungen angepasst ist. Die Grundlage für die Entwicklung stellt das generische Vorgehen aus der Entscheidungstheorie dar, welches insbesondere um Methoden und Konzepte ergänzt wurde, die eine detaillierte Untersuchung der softwaretechnischen Eigenschaften von Architekturen und alternativen Lösungsansätzen ermöglicht. Der entwickelte Entscheidungsprozess beinhaltet vier Phasen, um ausgehend von den Zielen, Rahmenbedingungen und dem Modell der existierenden Architektur systematisch alternative Lösungsansätze zur Architekturentwicklung und -veränderung auswählen, vergleichen und bewerten zu können. Der Beitrag des Entscheidungsprozesses ist die systematische Strukturierung der Entscheidungsfindung in Form eines Prozesses mit aufeinander folgenden Phasen. Der strukturierte Prozess ist eine wichtige Grundlage, um ein einheitliches Verständnis über den Verlauf der Entscheidungsfindung zu gewährleisten und eine Gruppendiskussion über die Ziele, Rahmenbedingungen und Lösungsansätze zu ermöglichen. Die Erfahrungen aus der Praxis zeigen, dass fehlende Klarheit über das Vorgehen bei Architekturentscheidungen häufig dazu führt, dass zu früh über alternative Lösungsansätze diskutiert wird, ohne die Ziele, Rahmenbedingungen und die existierende Architektur ausreichend detailliert betrachtet zu haben.

Entsprechend der Ausführungen in den vorangegangenen Kapiteln tragen ausgewählte Methoden und Konzepte aus der Entscheidungstheorie in Kombination mit den Methoden und Konzepten zur detaillierten Analyse der softwaretechnischen Eigenschaften von Architekturen zu einer Reduzierung von Komplexität, Risiken und Unsicherheit von Architekturentscheidungen bei. Insbesondere die Identifizierung und Berücksichtigung der Abhängigkeitsbeziehungen zwischen den Zielen, den Maßnahmen zur Architekturentwicklung und den Architekturkomponenten trägt dazu bei, Unsicherheiten über Abhängigkeiten und die daraus resultierenden Risiken zu reduzieren. Durch die kontinuierliche Identifizierung und Berücksichtigung der Abhängigkeitsbeziehungen in allen Phasen des Entscheidungsprozesses, die Klarheit über die softwaretechnischen Eigenschaften der Architektur sowie das systematische Vorgehen zur Entscheidungsfindung kann das Risiko einer Fehlentscheidung gesenkt und die Qualität der Entscheidungsfindung erhöht werden. Zudem wird dadurch die Wiederverwendung der Ergebnisse der Entscheidungsfindung bei einer vergleichbaren Folgeentscheidung erleichtert.

Wichtigster Teil der Weiterentwicklung des generischen Vorgehens aus der Entscheidungstheorie ist das strukturierte Vorgehen zur Auswahl, Verfeinerung und Konkretisierung alternativer Lösungsansätze. Die Erfahrungen aus der Praxis zeigen, dass zwar die Ziele, Rahmenbedingungen und Eigenschaften der existierenden Architektur im Detail identifiziert, ausgewertet und dokumentiert werden können. Gerade die Auswahl und die detaillierte Untersuchung der Lösungsansätze setzt jedoch ein großes Maß an Erfahrung voraus und ist gerade bei Entscheidungen unter hohem Zeit- und Termindruck eine der Hauptursachen für Fehlentscheidungen. Einerseits ermöglicht der entwickelte Katalog mit Kategorien von Lösungsansätzen und deren Qualitätseigenschaften ein systematisches Auswahlverfahren. Andererseits tragen die anschließende Verfeinerung und Konkretisierung der ausgewählten Lösungsansätze, welche anhand von szenariobasierten Analysen von Architekturprototypen durchgeführt werden, dazu bei, mit angemessenem Analyseaufwand die wichtigsten Eigenschaften aber auch Risiken aus Seiteneffekten und Wechselwirkungen ermitteln zu können.

Da im Entscheidungsprozess jedoch nicht alle Unsicherheiten – gerade über Abhängigkeiten und deren Seiteneffekte – aufgelöst werden können, ist die Berücksichtigung von wahrscheinlichen und unwahrscheinlichen Varianten ein

wichtiger Beitrag, um Architekturentscheidungen auch unter verbleibenden Unsicherheiten durchführen zu können. Mittels geeigneter Verfahren aus der Entscheidungstheorie kann die Wahrscheinlichkeit des Eintritts eines positiven oder negativen Ereignisses gezielt quantifiziert werden. Die aufgestellten Varianten werden durch spezielle Bewertungsmethoden methodisch aufgelöst und können damit im systematischen Entscheidungsprozess berücksichtigt werden. Trotzdem die Berücksichtigung von Wahrscheinlichkeiten ein zusätzliches Maß an Unsicherheit für die Entscheidungsfindung darstellen kann, ist dies ein wesentlicher Beitrag, um eine Entscheidungssituation realistisch abzubilden.

Aus Aufwandsgründen ist es erforderlich, die Notwendigkeit der Durchführung der Analyse-, Design- und Bewertungstätigkeiten des Entscheidungsprozesses im Hinblick auf das Risiko und die Komplexität der Entscheidung kritisch zu hinterfragen. Durch die Aufteilung der Tätigkeiten zur Entscheidungsfindung in obligatorische und optionale Tätigkeiten wird die Grundlage geschaffen, um eine systematische Entscheidungsfindung auch bei hohem Termin- und Zeitdruck zu ermöglichen. Die obligatorischen Tätigkeiten stellen dabei den Mindestumfang einer Entscheidungsfindung nach rationalen Gesichtspunkten dar; speziell im Hinblick auf die Dokumentation und zur Erfüllung der Nachvollziehbarkeit der Entscheidung. Je nach Risiko und Komplexität der Entscheidung ist die Durchführung ergänzender optionaler Analyse-, Design- und Bewertungstätigkeiten erforderlich und gerechtfertigt. Kriterium ist hierbei die Einhaltung eines ökonomisch sinnvollen Aufwand-Nutzen-Verhältnisses. Die Anpassbarkeit an Risiko und Komplexität stellt einen wichtigen Beitrag zur Akzeptanz des Entscheidungsprozesses in der Praxis auch unter hohem Zeit- und Termindruck dar.

Die Anwendung des Entscheidungsprozesses ermöglicht darüber hinaus eine einheitliche Dokumentation des Entscheidungsverlaufs und stellt damit die Grundlage für die Nachvollziehbarkeit der Entscheidung dar. Die Erfahrungen in der Praxis zeigen, dass sich insbesondere die Rahmenbedingungen und Ziele einer Architekturentscheidung oft verändern. Dadurch steigt die Wahrscheinlichkeit, dass die Implementierung des ausgewählten Lösungsansatzes nicht im erforderlichen Umfang zur Zielerreichung beiträgt oder Seiteneffekte bzw. Wechselwirkungen auftreten, wodurch der Entscheidungsträger einem immer stärkeren Rechtfertigungsdruck ausgesetzt ist. Die im Rahmen des Entscheidungsverlaufs erstellten Dokumente ermöglichen eine ausführliche sowie detaillierte Begründung und Nachvollziehbarkeit, unter welchen Voraussetzungen und mit welchen Annahmen eine Architekturentscheidung getroffen wurde.

9.2 Ausblick

Der im Rahmen dieser Dissertation entwickelte Entscheidungsprozess zeigt, wie durch eine systematische Entscheidungsfindung nach rationalen Gesichtspunkten Komplexität, Risiken und Unsicherheiten von Architekturentscheidungen reduziert werden können. Damit können jedoch nicht alle offenen Fragen und Lücken bei Architekturentscheidungen gelöst und geschlossen werden. Es verbleibt Raum für zukünftige Entwicklungen in diesem Themenumfeld.

Ein wichtiger Ansatzpunkt für zukünftige Arbeiten ist die Optimierung der Entwicklungsschritte zur Implementierung eines Lösungsansatzes, die im Rahmen der vorliegenden Dissertation als Maßnahmen zur Architekturentwicklung definiert werden. In der dritten Phase des Entscheidungsprozesses wird eine detaillierte Maßnahmenfolge erarbeitet, die den Prozess der Implementierung eines Lösungsansatzes beschreibt. Im Rahmen der Verfeinerung und Konkretisierung erfolgt zwar eine Ergänzung oder Korrektur der Maßnahmenfolge, sofern die Ziele nicht im erforderlichen Umfang erfüllt werden oder Wechselwirkungen sowie Seiteneffekte auftreten können. Das Vorgehen zur Verfeinerung und Konkretisierung ist im Hinblick auf eine Optimierung der Maßnahmenfolge zu ergänzen und weiterzuentwickeln. Gerade bei mehrstufigen Architekturentscheidungen, die verschiedene Kombinationen von Lösungsansätzen umfassen, kann ein hohes Optimierungspotenzial erzielt werden, durch das ein oder mehrere Optimierungsschritte als integrale Bestandteile des Entscheidungsprozesses auch im Hinblick auf das Aufwand-Nutzen-Verhältnis gerechtfertigt sein können.

Der im Rahmen der vorliegenden Dissertation entwickelte Entscheidungsprozess ist als generischer Ansatz konzipiert, der als eigenständiger Teil in die etablierten Entwicklungsprozesse integriert werden kann, um durch die systematische Entscheidungsfindung die Komplexität, die Risiken und die Unsicherheiten der Architekturentwicklung insgesamt reduzieren zu können. Um den Entscheidungsprozess jedoch vollständig in einen Entwicklungsprozess, wie zum Beispiel das Was-

serfallmodell oder das Spiralmodell [Balz00], zu integrieren, sind weitere Verfeinerungen und Anpassungen an den Analyse-, Design- und Bewertungstätigkeiten der einzelnen Phasen erforderlich. Die Erfahrungen bei der Anwendung des Entscheidungsprozesses innerhalb verschiedener Entwicklungsprozesse zeigen, dass die Analyse-, Design- und Bewertungstätigkeiten je nach Prozessmodell spezifisch angepasst und erweitert werden müssen. Ein wichtiger Punkt ist hierbei die Anwendung des Entscheidungsprozesses in der Praxis und die kontinuierliche Weiterentwicklung der Tätigkeiten für verschiedene Anwendungsdomänen.

Ein weiterer wichtiger Ansatzpunkt für zukünftige Arbeiten ist die Entwicklung einer geeigneten Werkzeugunterstützung. Für den im Rahmen der vorliegenden Dissertation entwickelten Entscheidungsprozess wird eine Werkzeugunterstützung zwar nicht zwingend benötigt. Insbesondere bei der Beschreibung des Architekturmodells, bei der Ermittlung der Abhängigkeitsbeziehungen sowie bei der Bewertung der einzelnen Lösungsansätze mit ihren Varianten kann eine Werkzeugunterstützung jedoch den Aufwand zur Entscheidungsfindung senken und die Qualität der Analysen erhöhen. Durch ein teilautomatisiertes Vorgehen können z. B. Berechnungsfehler vermieden und Inkonsistenzen, beispielsweise bei der Modellierung oder der Ermittlung der Abhängigkeitsbeziehungen, durch geeignete Prüfverfahren zuverlässig erkannt werden. Bei der Gestaltung einer geeigneten Werkzeugumgebung ist jedoch zu berücksichtigen, dass das Verfahren zur Entscheidungsfindung eine Vielzahl kognitiver und kreativer Tätigkeiten umfasst, die nur bedingt automatisiert oder durch Werkzeuge unterstützt werden können. Es ist bei der Entwicklung einer Werkzeugunterstützung daher immer abzuwägen, welche Routinetätigkeiten leicht automatisiert werden können und bei welchen Tätigkeiten eine Werkzeugunterstützung aufgrund der Variantenvielfalt nicht sinnvoll ist.

Kapitel 10

Anhang

10.1 Verfahren zur objektiven und subjektiven Messung von Wahrscheinlichkeiten

Methoden zur subjektiven Messung von Wahrscheinlichkeiten

Bei der Messung von Wahrscheinlichkeiten ist zwischen direkter und indirekter Messung zu unterschieden. In der ersten Variante, der direkten Messung, wird in einer Befragung direkt nach der Wahrscheinlichkeit einer Variante gefragt. Die indirekte Messung basiert auf Vergleichen; der Befragte soll anhand von Referenzsituationen Vergleiche und damit Rückschlüsse auf die aktuelle Entscheidungssituation ziehen, um die Wahrscheinlichkeiten indirekt zu ermitteln. Dabei ist zu beachten, dass der Befrager und der Befragte auch dieselbe natürliche Person sein kann, sofern diese mit dem Verfahren zur Messung von Wahrscheinlichkeiten vertraut ist. In Tab. 10.1 ist eine Übersicht dargestellt, mit welchen Vorgehensweisen subjektive Wahrscheinlichkeiten gemessen werden können.

Werte / Wahrscheinlichkeiten	***Direkte Messung***	***Indirekte Messung***
Frage nach Wahrscheinlichkeiten	Wahrscheinlichkeiten diskreter Ereignisse, Wahrscheinlichkeitsdichte kontinuierlicher Variablen, Verteilungsfunktion	
Frage nach Werten einer unsicheren Variable	Verteilungsfunktion	

Tab. 10.1: Subjektive Messmethoden von Wahrscheinlichkeiten

Zur direkten oder indirekten Messung der Wahrscheinlichkeiten wird – im Groben – wie folgt vorgegangen [EiWe03]:

- *Direkte Wahrscheinlichkeitsabfrage:* Dabei wird der Befragte direkt gefragt, wie hoch er die Wahrscheinlichkeit des Eintrittes von bestimmten Ereignissen einschätzt. Bei allen Verfahren wird auf der Grundlage der Antworten eine Wahrscheinlichkeits- oder Dichtefunktion aufgestellt und es wird eine Verteilungsfunktion abgeleitet. Ein Beispiel ist die Frage, mit welcher Wahrscheinlichkeit durch die Implementierung eines bestimmten Architekturmusters die Performance oder die Sicherheit erhöht wird.
- *Direkte Wertabfrage:* Dem Befragten werden bekannte Wahrscheinlichkeiten eines Ereignisses vorgegeben und es wird abgefragt, welche Wahrscheinlichkeit der Eintritt eines vergleichbaren Ereignisses hat.
- *Indirekte Wahrscheinlichkeitsabfrage:* Hierbei wird nach dem Lotterieverfahren vorgegangen. Neben einer unsicheren Ereignisbeschreibung wird dem Befragten z. B. ein Korb mit farblich unterschiedlichen Bällen zur Verfügung gestellt. Um zu gewinnen, soll der Befragte nun entscheiden, ob eher die das Ereignis eintritt, oder eher ein Ball mit einer bestimmten Farbe gezogen wird. Ausschlaggebend ist dabei das Verhältnis der Bälle mit unterschiedlichen Farben. Bei einem Korb mit 9 weißen und einem gelben Ball, ist die Wahrscheinlichkeit den gelben Ball zu ziehen 10 %. Das Verhältnis der Bälle wird so lange verändert, bis der Befragte indifferent zwischen der Ballziehung und dem Eintritt der unsicheren Situation ist. Sind dann beispielsweise 6 weiße und 4 gelbe Bälle im Korb, liegt die Wahrscheinlichkeit für den Eintritt des unsicheren Ereignisses bei 40 %. Neben dem Korb mit Bällen kann auch ein Wahrscheinlichkeitsrad eingesetzt werden, bei dem ein Tortenstück in einer anderen Farbe dargestellt ist.
- *Indirekte Wertabfrage:* Wie bei der indirekten Wahrscheinlichkeitsabfrage wird nach dem Lotterieverfahren vorgegangen, z. B. mit einem Korb mit weißen und gelben Bällen in einem fest vorgegebenen Verhältnis. Der Befragte wird aber

nicht mit einem jeweils unterschiedlichen Ballverhältnis konfrontiert, sondern die Parameter des Ereignisses werden verändert, z. B. der Umfang von Restrukturierungsmaßnahmen. Der Befragte wird dann z. B. danach gefragt, ob eher ein weißer Ball gezogen wird oder die Restrukturierung eine bestimmte Schwachstelle oder Problembereich beheben.

Die vier grundsätzlichen Arten zur Messung von Wahrscheinlichkeiten zeigen, dass eine Wahrscheinlichkeit nicht ausschließlich auf der Grundlage von empirischen Daten gemessen werden kann, sondern auch in Form eines Interviews. Dabei ist jedoch auf die Qualität der gemessenen Wahrscheinlichkeiten zu achten. Die ermittelten Werte sollten konsistent sein und nur geringfügig um die Verteilungsfunktion streuen.

Methoden zur objektiven Messung von Wahrscheinlichkeiten

Gegenstand der Messung sind einzelne Ereignisse, Zustände oder Szenarien; die Messung erfolgt durch die Ermittlung von Wahrscheinlichkeitsverteilungen von numerischen Größen (sog. Zufallsvariablen) [EiWe03]. Die Verteilung von Zufallsvariablen kann durch Wahrscheinlichkeitsfunktionen (auch als Dichtefunktionen bezeichnet) oder durch Verteilungsfunktionen beschrieben werden. Dabei sind diskrete und kontinuierlich (stetig) steigende Zufallsvariablen zu unterscheiden:

- *Diskrete Zufallsvariablen:* Jede mögliche Ausprägung x_i besitzt eine positive Wahrscheinlichkeit p_i mit der Wahrscheinlichkeitsfunktion $p(x_i)$. Die Verteilungsfunktion $P(x)$ (siehe Abb. 10.1) beschreibt die Wahrscheinlichkeit, mit der die Zufallsvariable einen Wert kleiner oder gleich x annimmt. Eine Verteilungsfunktion wird benötigt, um Fehler bei der Messung von Wahrscheinlichkeiten zu erkennen und die Konsistenz der Schätzungen sicherzustellen.

$$P(x) = \sum_{x_{(i)} \le x} p_i \quad .$$

Abb. 10.1: Verteilungsfunktion diskreter Zufallsvariablen

- *Kontinuierlich steigende Zufallsvariablen:* Hierbei werden nicht einzelne Wahrscheinlichkeiten sondern Intervalle betrachtet. Die Wahrscheinlichkeit wird hierbei mit einer (Wahrscheinlichkeits-) Dichtefunktion angegeben. Erstellt wird eine solche Dichtefunktion mit einem Säulendiagramm (Histogramm), bei der die Fläche jeder Säule das Maß für Wahrscheinlichkeit beschreibt, dass die Zufallsvariable in das Intervall fällt. So kann eine Architekturveränderung mit einer Wahrscheinlichkeit von 90 % z. B. zwei bis drei Personentage für Reviews und Tests benötigen (siehe Abb. 10.2).

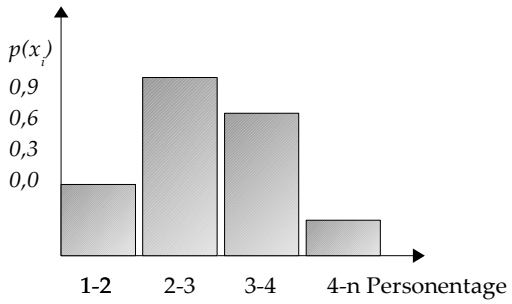


Abb. 10.2: Dichtefunktion bei kontinuierlich steigenden Zufallszahlen für die Dauer von Tests

Theorem von Bayes

Das Theorem von Bayes ist ein Verfahren zur Ermittlung von Wahrscheinlichkeiten auf der Grundlage von anderen Wahrscheinlichkeiten [Harn03] (u. [EiWe03]). Damit kann eine zunächst angenommene Wahrscheinlichkeit (sog. a priori-Wahrscheinlichkeit) im Licht neuer Daten revidiert werden. Die Voraussetzung für die Einsetzbarkeit dieses Ansatzes ist jedoch die Verfügbarkeit von Wahrscheinlichkeiten zum aktuell betrachteten Entscheidungsproblem.

Zunächst werden mit einem der in den vorangegangenen Abschnitten aufgeführten Verfahren die Wahrscheinlichkeiten $p(s_1)$, $p(s_2)$, ..., $p(s_n)$ für unsichere Umweltzustände $s_1, s_2, ..., s_n$ ermittelt. Das Theorem zieht eine Informationsquelle mit in die Berechnung ein, welche genau eine Information der Informationsmenge $Y = \{y_1, y_2, ..., y_n\}$ liefert. Ein Beispiel ist eine Übersicht über fehleranfällige Komponenten oder über Komponenten, bei denen selbst kleine Veränderungen langwierige Tests erfordern. Darüber hinaus sind die bedingten Wahrscheinlichkeiten $p(y_j|s_i)$ zu ermitteln, dass aufgrund der Information y_j ein bestimmter Zustand s_i eintritt. So kann die Veränderung einer komplexen uns fehleranfälligen Komponente langwierige Tests erfordern. Es kann aber auch sein, dass nur eine kürzere Testdauer erforderlich ist. Dieses Beispiel für bedingte Wahrscheinlichkeiten ist in Abb. 10.3 dargestellt. Die bedingten Wahrscheinlichkeiten werden auch als Likelihoods bezeichnet.

Wahrscheinlichkeiten für unsichere Zustände	p(Testdauer 1-3 Personentage) = 0,6 p(Testdauer >4 PT) = 0,4
Bedingte Wahrscheinlichkeiten der Zustände	p(Komponente fehleranfällig | Testdauer 1-3 PT) = 0,3 p(Komponente nicht fehleranfällig | Testdauer 1-3 PT) = 0,7 p(Komponente fehleranfällig | Testdauer >4 PT) = 0,8 p(Komponente nicht fehleranfällig | Testdauer >4 PT) = 0,2

Abb. 10.3: Beispiel für bedingte Wahrscheinlichkeiten (PT – Personentage)

Mit der in Abb. 10.4 dargestellten Formel des Theorems von Bayes kann die Wahrscheinlichkeit unsicherer Zustände $p(s_i \mid y_j)$ unter Betrachtung der Informationsmenge Y errechnet werden.

$$p(s_i|y_j) = \frac{p(s_i) \cdot p(y_j|s_i)}{\sum p(s_i) \cdot p(y_j|s_i)} \quad .$$

Abb. 10.4: Theorem von Bayes

Für die in Abb. 10.3 dargestellten Beispiele können die folgenden Wahrscheinlichkeiten (siehe Abb. 10.5) ermittelt, werden, dass die Veränderung einer fehleranfälligen Komponente langwierige Tests erfordert.

$$p(Testdauer\ 1-3\ PT|Komponente\ fehleranfällig) = \frac{0,3 \cdot 0,6}{0,3 \cdot 0,6 + 0,8 \cdot 0,4} = 0,36$$

$$p(Testdauer\ 1-3\ PT|Komponente\ nicht\ fehleranfällig) = \frac{0,7 \cdot 0,6}{0,7 \cdot 0,6 + 0,2 \cdot 0,4} = 0,84$$

Abb. 10.5: Auswertung bedingter Wahrscheinlichkeiten mit dem Theorem von Bayes

Das Theorem von Bayes stellt ein Verfahren dar, um bedingte Wahrscheinlichkeiten systematisch auszuwerten und so realistischere Wahrscheinlichkeiten zu ermitteln. Die Anwendung dieses Verfahrens kann daher zur Reduzierung der Unsicherheiten beitragen, sofern die Informationsmenge Y genügend Informationen über die tatsächliche Entscheidungssituation beinhaltet.

10.2 Obligatorische und optionale Tätigkeiten im Entscheidungsprozess

Die folgende Tabelle Tab. 10.2 umfasst alle optionalen und obligatorischen Tätigkeiten des Entscheidungsprozesses (siehe für die Phasen Abb. 3.1 auf S. 44). Alle optionalen Tätigkeiten, die nur in Kombination mit einer anderen Tätigkeit durchgeführt werden, sind durch kursive Schrift hervorgehoben; alle obligatorischen Tätigkeiten sind durch Fettdruck hervorgehoben.

Kapitel	Bezeichnung der Tätigkeit	Obligatorisch / Optional	Begründung für obligatorische Tätigkeiten Kriterien für optionale Tätigkeiten
Phase 1 – Identifizierung und Strukturierung der Ziele und Rahmenbedingungen			
5.1.1	**Identifizierung der Ziele auf der Grundlage von Schwachstellen und Problembereichen**	**Obligatorisch**	**Die Ziele sind die wichtigsten Kriterien der Entscheidung und müssen in detaillierter Form identifiziert und dokumentiert werden.**
	Einsatz von Architekturanalysen zur Identifizierung der Ziele	*Optional (bei 5.1.1)*	*Sofern die Schwachstellen und Problembereiche noch nicht in ausreichendem Maß bekannt sind und ermittelt wurden, ist eine Analyse der Architektur durchzuführen. Dafür eignen sich insbesondere szenariobasierte Analysemethoden, z. B. ATAM oder ALMA (siehe S. 29ff.).*
	Einsatz der ISO-Norm 9126 zur Identifizierung und Strukturierung der Qualitätsziele	*Optional (bei 5.1.1)*	*Werden im vorliegenden Entscheidungsfall die Qualitätsziele unterschiedlich interpretiert und existieren Unsicherheiten über Abhängigkeitsbeziehungen, kann für eine einheitliche Formulierung der Ziele und zur Unterstützung der Identifizierung von Abhängigkeitsbeziehungen die ISO-Norm 9126 eingesetzt werden (siehe S.56f.). Die Formulierung der Qualitätsziele mittels der ISO-Norm ist außerdem eine wichtige Grundlage für die Nachvollziehbarkeit der Entscheidung.*
5.1.2	Erkennung und Auflösung von Abhängigkeitsbeziehungen und widerspruchsfreie Ziel-Strukturierung	Optional	Erforderlich bei komplexen Ursache-Wirkungs- und Konkurrenzbeziehungen zwischen den Zielen, die eine klare und eindeutige Auswahl von Lösungsansätzen nicht ermöglichen.
	Ziel-Einschränkungen	Optional	Kann die Zahl der identifizierten Fundamentalziele nicht auf weniger als zwei bis drei Ziele reduziert werden, ist aufgrund der hohen kombinatorischen Komplexität sowie der damit verbundenen Unsicherheiten und Risiken eine Einschränkung auf die wichtigsten Ziele erforderlich. Ist beispielsweise eine Entscheidung zwischen fünf Lösungsansätzen zu treffen, die fünf Ziele erfüllen müssen, sind in der Summe 125 Kriterien zu überprüfen. Aufgrund der Vielzahl an Analysen und Einzelbewertungen können die zentralen Ziele des Entscheidungsprozesses, die Senkung der Komplexität, der Unsicherheiten und der Risiken, nicht im vollen Umfang erfüllt werden. Darüber hinaus sind derart komplexe Entscheidungsabläufe nur bedingt nachvollziehbar.
5.1.3	**Erhebung der organisatorischen und technischen Rahmenbedingungen**	**Obligatorisch**	**Die Aufstellung und Abarbeitung der Checkliste der Rahmenbedingungen ist obligatorisch, da die Rahmenbedingungen neben den Zielen die wichtigsten Entscheidungskriterien darstellen.**

Kapitel	Bezeichnung der Tätigkeit	Obligatorisch / Optional	Begründung für obligatorische Tätigkeiten Kriterien für optionale Tätigkeiten
5.1.4	Erkennung und Auflösung von Abhängigkeitsbeziehungen zwischen Rahmenbedingungen	Optional	Sofern konkurrierende Abhängigkeitsbeziehungen zwischen den Rahmenbedingungen existieren, z. B. aufgrund knapper Budgets oder enger Terminpläne, sind diese aufzulösen. Dies ist jedoch nur in begrenztem Maße möglich, da nicht alle Rahmenbedingungen in ausreichendem Maße vom Entscheidungsträger beeinflusst werden können.
Phase 2 – Architekturbeschreibung und Risikoeinstufung			
5.2.1	**Einstufung des Risikos der Entscheidung**	**Obligatorisch**	**Um den Umfang und den Detaillierungsgrad der Analyse- und Bewertungstätigkeiten für die Entscheidungsfindung festzulegen, ist das mit der Entscheidung verbundene Risiko zu bestimmen.**
5.2.2	**Beschreibung der relevanten Teile der Architektur**	**Obligatorisch**	**Eine Beschreibung der Architektur ist bei jeder Architekturentscheidung erforderlich, da auf dieser Grundlage die Auswahl, Verfeinerung und Konkretisierung der Lösungsansätze erfolgt. Je nach Risikoklasse der Entscheidung kann es erforderlich werden, den Betrachtungsraum des Modells zu erweitern, indem Um- und Drittsysteme mit in die Architekturbeschreibung aufgenommen werden.**
	Einsatz formaler oder semi-formaler Architektur- oder Modellbeschreibungssprachen sowie Sichten	*Optional (bei 5.2.2)*	*Ausschließlich riskante und sehr riskante Entscheidungen rechtfertigen den Einsatz semi-formaler/formaler Beschreibungssprachen oder den Einsatz von Sichten, da dadurch die Komplexität reduziert und die Qualität der Architekturanalysen erhöht werden kann.*
5.2.3	Ermittlung unbekannter Abhängigkeitsbeziehungen zwischen Komponenten	Optional	Dieser Schritt ist vor allem dann erforderlich, wenn aus der Dokumentation und der späteren Auswertung der Beziehungen ein Nutzen für die Vorauswahl und die Konkretisierung in Form einer Reduzierung der Unsicherheiten, Risiken und Komplexität ersichtlich wird. Ein Indikator für die Notwendigkeit sind Komponenten mit einer hohen Komplexität; dieser Wert ist z. B. mit dem Distanzmaß (S. 64f.) bestimmbar.
Phase 3 – Vorauswahl und Konkretisierung alternativer Lösungsansätze			
5.3.1	Aufstellung eines Grobplanes	Optional	Zur Reduzierung der kombinatorischen Komplexität speziell bei mehrstufigen Architekturentscheidungen erforderlich
	Skizzierung von Lösungsszenarien	*Optional (bei 5.3.1)*	*Die Skizzierung von Lösungsszenarien ist nur dann erforderlich, wenn bei mehrstufigen Architekturentscheidungen noch keine Klarheit über die zur Verfügung stehenden Lösungsansätze und über die möglichen Wege der Zielerreichung existiert.*
	Auflösung von Abhängigkeitsbeziehungen, Priorisierung und Aufstellung des Grobplanes	*Obligatorisch (bei 5.3.1)*	*Sofern ein Grobplan aufgestellt wird, sind die Abhängigkeitsbeziehungen zwischen den groben Veränderungsschritten zu ermitteln und es ist eine Reihenfolgeplanung durchzuführen.*
5.3.2	Vorauswahl auf Kategorieebene	Optional	Diese Form der Vorauswahl ist nur dann erforderlich, sofern noch kein ausreichender Überblick über die geeigneten Lösungsansätze zur Erfüllung der Ziele existiert. Dieser Schritt ist insbesondere bei der Auswahl von Lösungsansätzen zur Erfüllung nicht-funktionaler, qualitativer Ziele erforderlich, da die positiven und negativen Qualitätseigenschaften der Lösungsansätze oftmals nur durch spezielle Analysen ermittelt werden können.

Kapitel	Bezeichnung der Tätigkeit	Obligatorisch / Optional	Begründung für obligatorische Tätigkeiten Kriterien für optionale Tätigkeiten
5.3.3	Analyse und Ranking der Lösungsansätze	Optional	Ein Ranking der Lösungsansätze ist erforderlich, um eine große Menge an geeigneten Lösungsansätzen systematisch zu reduzieren. Die Beschränkung auf die wichtigsten zwei bis drei Lösungsansätzen ist wichtig, da mit der Verfeinerung und Konkretisierung der Lösungsansätze ein erheblicher Aufwand verbunden ist. Zudem steigt mit der Anzahl der betrachteten Lösungsansätze die kombinatorische Komplexität der Entscheidungsfindung an.
	Szenariobasierte Architekturanalyse mit prototypischer Architekturveränderung	*Optional (bei 5.3.3)*	*Für das Ranking der Lösungsansätze ist bereits ein grober Überblick über die Eigenschaften der Lösungsansätze erforderlich,* u. a. *über den Grad der Zielerreichung und die Einhaltung der Rahmenbedingungen. Um die Eigenschaften zu ermitteln, kann es insbesondere bei riskanten oder sehr riskanten Entscheidungen (siehe Tätigkeit 5.2.1) erforderlich sein, eine prototypische Architekturveränderung vorzunehmen und diesen Architekturprototypen mittels geeigneter Szenarien zu untersuchen. Dafür eignen sich insbesondere szenariobasierte Analysemethoden, z. B. ATAM oder ALMA (siehe S. 29ff.).*
5.3.4	**Konkretisierung und Anpassung der Lösungsansätze**	**Obligatorisch**	**Einer der wichtigsten Schritte im Entscheidungsprozess ist die Verfeinerung und Konkretisierung der Lösungsansätze. Dieser Schritt ist obligatorisch, da nur auf der Grundlage der im Detail ermittelten Eigenschaften der Lösungsansätze ein objektiver und systematischer Vergleich erfolgen kann.**
5.3.4.1	**Aufstellung und Verfeinerung der Maßnahmenfolge**	**Obligatorisch**	**Eine konkrete, verfeinerte und detaillierte Maßnahmenfolge, die den Prozess der Implementierung beschreibt, ist die Grundlage, um die Eigenschaften der Lösungsansätze und mögliche Wechselwirkungen und Seiteneffekte bestimmen zu können.**
5.3.4.2	**Auswertung der Abhängigkeitsbeziehungen**	**Obligatorisch**	**Dieser Schritt ist erforderlich, um die Abhängigkeitsbeziehungen zwischen den Komponenten aufzulösen. Ziel ist es, dass im Anschluss an die Verfeinerung die betreffenden Komponenten eine 'positive Auswirkung' auf die Ziele haben.**
5.3.4.3	**Ermittlung der Eigenschaften der Lösungsansätze**	**Obligatorisch**	**Die systematische Ermittlung der Eigenschaften der Lösungsansätze, u. a. den Grad der Erfüllung der Ziele, die Einhaltung der Rahmenbedingungen, ist eine wesentliche Voraussetzung, um einen objektiven und systematischen Vergleich der Lösungsansätze durchführen zu können.**
	Szenariobasierte Architekturanalyse	*Optional (bei 5.3.4.3)*	*Gerade bei riskanten und sehr riskanten Entscheidungen, der Klasse 1 und höher (siehe Tätigkeit 5.2.1), kann es erforderlich sein, die Eigenschaften auf einer methodisch-fundierten Analysegrundlage abzuschätzen. Da bei diesen Entscheidungen eine Fehlentscheidung zu massiven negativen Folgen führen kann, ist der zusätzliche Analyseaufwand gerechtfertigt. Für die Analyse eignen sich insbesondere szenariobasierte Analyseverfahren, z. B. ATAM oder ALMA*

Kapitel	Bezeichnung der Tätigkeit	Obligatorisch / Optional	Begründung für obligatorische Tätigkeiten Kriterien für optionale Tätigkeiten
5.3.4.4	Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften	Optional	Vor allem bei der Durchführung eines architekturorientierten Refactoring ist es erforderlich, dass die funktionalen Eigenschaften des Softwaresystems nicht verändert werden. Daher ist zu überprüfen, welche Veränderungen auftreten und welche Gegenmaßnahmen ergriffen werden können.
Phase 4 – Rationale Bewertung und Entscheidung über alternative Lösungsansätze			
5.4.1	Bewertung der Lösungsansätze	Optional	Eine Bewertung der Lösungsansätze dann erforderlich, wenn der Wert eines Lösungsansatzes nicht allein durch Addition der einzelnen Eigenschaftswerte ermittelt werden kann. Das ist insbesondere dann der Fall, wenn Wahrscheinlichkeiten zu berücksichtigen sind.
5.4.2	**Treffen der Entscheidung**	**Obligatorisch**	**Die Entscheidungsfindung ist mit der Interpretation und der Entscheidung zugunsten des besten Lösungsansatzes beendet.**
	Interpretation des Ergebnisses der Entscheidungsfindung	Optional (bei 5.4.2)	Gerade bei minimalen Abweichungen zwischen den Lösungsansätzen ist zu prüfen, ob weitere Ziele, Faktoren oder Eigenschaften existieren, die bislang noch nicht in ausreichendem Maße in die Entscheidungsfindung mit einbezogen worden sind.

Tab. 10.2: Optionale und Obligatorische Tätigkeiten im Entscheidungsprozess

10.3 Übersicht über Veränderungsszenarien

Bengtsson et. al. schlagen die in Tab. 10.3 dargestellten Kategorien von Veränderungsszenarien vor [Ben+00]. Diese sind einerseits Gegenstand der Architecture-level modifiability analysis (ALMA). Diese dienen andererseits als Grundlage für die Entwicklung von Szenarien zur Ermittlung der Eigenschaften der Lösungsansätze (siehe Kapitel 5.3.4, S. 73ff.).

	Changes in the functional specification	**Changes in the requirements**	**Changes in the technical environment**	**Other sources**
Change scenarios that require adaptations to the system and these adaptations have external effects	Functions of the system under analysis are used by other systems. One of them is modified and, as a result, other systems have to be modified as well.	A requirement of the system is changed (for example a higher performance is required). To realize this, systems that are used by the system under analysis have to be modified as well.	Part of the technical environment of the system is modified. Because this part of the technical environment is sieheared with other systems, these systems have to be modified as well.	For reasons, other than the aforementioned three, part of the system has to be adapted. As a result of these changes, other systems have to be adapted as well.
Change scenarios that require adaptations to the environment of the system and these adaptations affect the system	The system uses functions of other systems. One of these functions is modified and, as a result, the system under analysis has to be modified as well.	A requirement of another system is changed. To realize this, the system under analysis has to be modified as well.	Part of the technical environment is modified for another system. Because this part of the technical environment is used by the system under analysis, it has to be modified as well.	For reasons, other than the aforementioned three, a system has to be adapted. As a result of these changes, other systems have to be adapted as well.
Change scenarios that require adaptations to the macro architecture	Because of changes in the functional specification, the structure of systems has to be modified.	Changes in the requirements cannot be realized in the current macro architecture and, therefore, the structure of systems has to be modified.	Changes in the requirements cannot be realized in the current macro architecture and, therefore, the structure of systems has to be modified.	Because of other changes, the structure of systems has to be modified.
Change scenarios that require adaptations to the micro architecture	Because of changes in the functional specification, the internal structure of the system under analysis has to be modified.	Changes in the requirements cannot be realized in the current micro architecture and, therefore, the internal structure of the system under analysis has to be modified.	Necessary changes in the technical environment require the structure of the system to be modified.	Because of other changes, the structure of the system has to be modified.
Change scenarios that introduce version conflicts	Change scenarios that introduce version conflicts	Changes in the requirements introduce different versions of the same component. However, these different versions cannot coexist because of conflicts	Changes in the technical environment introduce different versions of the same component. However, these different versions cannot coexist because of conflicts	Other changes introduce different versions of the same component. However, these different versions cannot coexist because of conflicts

Tab. 10.3: Kategorien von Veränderungsszenarien [Ben+00]

10.4 Datenbankfunktionen und Pfadangaben des Typo3-Beispiels

Datenbankfunktion INSERTquery

```
function INSERTquery($table,$fields_values)    {
   // Table and fieldnames should be "SQL-injection-safe"
   // when supplied to this function (contrary to values
   // in the arrays which may be insecure).
   if (is_array($fields_values) && count($fields_values))  {

   // Add slashes old-school:
   foreach($fields_values as $k => $v)    {
   $fields_values[$k] = $this->fullQuoteStr($fields_values[$k],
                                                          $table);
    }
      // Build query:
      $query = 'INSERT INTO '.$table.'
            (
            '.implode(',
            ',array_keys($fields_values)).'
                  ) VALUES (
            '.implode(',
            ',$fields_values).'
            )';

            // Return query:
            if ($this->debugOutput || $this->store_lastBuiltQuery)
                $this->debug_lastBuiltQuery = $query;
            return $query;
            }
      }
```

Listing 10.1: T3lib-Funktion INSERTquery

Pfadangaben der Bibliotheken und Skripte von Typo3

Die folgenden Bibliotheken, Skripte und Funktionen wurden im Rahmen der Typo3-Architekturveränderung verwendet (siehe Tab. 10.4, Tab. 10.5 und Tab. 10.6). Die Pfadangaben beziehen sich auf die Standard-Installation von Typo3 (Version 3.8.1).

Bibliothek	Beschreibung	Pfad
T3lib_db	Funktionen Datenbankzugriff	/typo3/t3lib/class.t3lib_db.php
T3lib_tcemain	Core-Engine	/typo3/t3lib/class.t3lib_tcemain.php
T3lib_sqlengine	Generierung SQL-Statements	/typo3/t3lib/class.t3lib_sqlengine.php
T3lib_sqlparser	Generierung SQL-Statements	/typo3/t3lib/class.t3lib_sqlparser.php
T3lib_tceforms	Generierung der Backend-Eingabefelder	/typo3/t3lib/class.t3lib_tceforms.php
T3lib_befunc	Backend-Funktionen	/typo3/t3lib/class.t3lib_befunc.php
T3lib_tsparser_ext	TypoScript Interpreter	/typo3/t3lib/class.t3lib_tsparser_ext.php
T3lib_tstemplate	TypoScript Interpreter	/typo3/t3lib/class.t3lib_tstemplate.php
T3lib_tsstyleconfig	Konfiguration von TypoScript	/typo3/t3lib/class.t3lib_tsstyleconfig.php
T3lib_extmgm	Extension-API	/typo3/t3lib/class.t3lib_extmgm.php
T3lib_tsparser	TypoScript Interpreter	/typo3/t3lib/class.t3lib_tsparser.php

Tab. 10.4: Verzeichnis der Typo3-Bibliotheken

PHP-Skript	Beschreibung	Pfad
index_ts.php	Frontend	/index.php mit Verknüpfung auf /typo3/sysext/cms/tslib/index_ts.php
index.php	Backend	/typo3/index.php
tce_db.php	Instanz der T3lib_tcemain	/typo3/tce_db.php

Tab. 10.5: Verzeichnis der Typo3-Skripte

Funktionen	Implementiert in	Zeile
exec_INSERTquery()	T3lib_db	168
exec_INSERTquery()	T3lib_db	168
exec_UPDATEquery()	T3lib_db	184
exec_DELETEquery()	T3lib_db	198
exec_SELECTquery()	T3lib_db	217
exec_SELECT_mm_query()	T3lib_db	242
exec_SELECT_queryArray()	T3lib_db	265
exec_SELECTgetRows()	T3lib_db	288
INSERTquery()	T3lib_db	333

Funktionen	Implementiert in	Zeile
process_datamap()	T3lib_tcemain	435

Tab. 10.6: Verzeichnis der Funktionen der Typo3-Komponenten

10.5 Erhöhung der Sicherheit eines E-Banking-Systems

10.5.1 Verfeinerung und Konkretisierung der Lösungsansätze des E-Banking-Beispiels

Stufe 1 – Lösungsansätze für TAN-Generierung und Übertragungsmedium

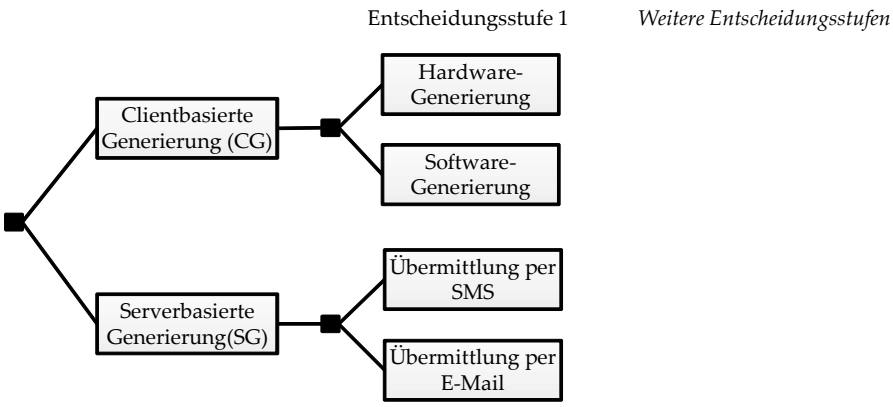


Abb. 10.6: Erste Stufe der Architekturentscheidung

Im Falle der 'clientbasierten Generierung' stehen die folgenden beiden Lösungsansätze zur Verfügung (siehe Abb. 10.6):

- *Hardware-Generator:* Ein Hardware-Generator ist eine Hardwarekomponente, z. B. ein Security-Token (siehe für ein Beispiel z.B. [Reim07] und [Bwba08]), welche den Algorithmus zur Generierung einer TAN beinhaltet. Um die Generierung zu starten, ist zusätzlich ein Codewort einzugeben, z. B. eine Ziffernfolge, welches dem Bankkunden über das E-Banking-Portal mitgeteilt wird. Als Prototyp wird dazu eine Java-Anwendung für ein Mobiltelefon entwickelt, mit der Test-TANs generiert werden können. Test-TANs sind gültige TANs mit denen zwar keine Aufträge authentifiziert aber die notwendigen Tests durchgeführt werden können.
  - *Verfeinerung der Maßnahmen:* Zu entwickeln ist eine mobile Java-Anwendung auf der Grundlage des J2ME-Frameworks mit den Teilmaßnahmen: Entwicklung der Benutzeroberfläche zur Eingabe des Codewortes, Entwicklung des Test-Algorithmus und Entwicklung der Anwendung zur Generierung der Test-TANs.
  - *Berücksichtigung von Abhängigkeitsbeziehungen und Ermittlung der Eigenschaften der Lösungsansätze:* Da die einzige Verbindung zwischen dem Hardware-Generator und den restlichen Teilen des E-Banking-Systems die manuelle Übergabe des Codewortes und der generierten TANs (beim Prototyp die Test-TANs) ist, kann von einer ausreichenden losen Kopplung ausgegangen werden (Zwischenzielerreichung 100 %). Um zu überprüfen, ob das Verfahren sicherer als das existierende TAN-Listen Verfahren ist, ist eine szenariobasierte Analyse erforderlich. Dabei wird das bereits skizzierte Lösungsszenario der 'clientbasierten Generierung' (siehe Abb. 6.24 auf S. 126) mit den folgenden Szenarien untersucht: 'missbräuchliche Nutzung nach einem Diebstahl' sowie 'Ausführung auf einem unsicheren und z. B. von Virenbefall gefährdeten Client-Rechner'. Mit dem gestohlenen Hardware-Generator können nur dann gültige TANs erzeugt werden, wenn ein gültiges Codewort eingegeben wurde. Da für die Übermittlung des Codewortes eine Anmeldung am E-Banking-Portal zwingend erforderlich ist, können mit dem gestohlenen Gerät allein keine gültigen TANs erzeugt werden. Außerdem ist der Hardware-Generator vom Rechner des Bankkunden physisch getrennt; somit haben Sicherheitsmängel, beispielsweise eine fehlende Firewall oder ein fehlendes Virenschutzprogramm keine direkten Auswirkungen auf die Sicherheit (Zwischenzielerreichung 100 %). Das Verfahren ist außerdem als sicher einzustufen, da die Hardware-Generatoren nur an berechtigte Personen ausgegeben werden und im Gegensatz zu einem Softwareprodukt schwerer zu kopieren sind. Nur durch Diebstahl oder Verlust kann der Hardware-Generator an unbefugte Personen gelangen (Zwischenzielerreichung 100 %). Der Implementierungsaufwand wird

auf 10 Personentage geschätzt. Zudem ist die Herstellung derartiger Geräte aufgrund fehlender Standards mit einem erheblichen finanziellen Zusatzaufwand verbunden ist.

- *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften*: Da der ursprüngliche Algorithmus zur TAN-Generierung nicht verändert wird, sind keine Veränderungen an den funktionalen Eigenschaften des E-Banking-Portals zu erwarten.
- *Dokumentation des Entscheidungsverlaufs*: Die Ergebnisse werden in Form des in Abschnitt 5.3.5.3 vorgestellten Datenblattes dokumentiert (siehe S. 84). Außerdem werden jene Besonderheiten dokumentiert, die Auswirkungen auf die Lösungsansätze der zweiten und dritten Stufe der Entscheidung haben können. So wird für den Prototyp der Hardware-Generierung die Programmiersprache Java mit dem Framework J2ME verwendet. Schon der prototypische Einsatz des Mobiltelefons zeigt, dass nur eingeschränkte Speicherressourcen und Eingabemöglichkeiten (mehrfach belegte Ziffern von 0-9) bestehen (siehe für ein Beispiel Abb. 10.9, S. 187).

- *Software-Generator:* Dies ist eine eigenständige und betriebssystemunabhängige Software-Anwendung, die auf dem Rechner des Clients installiert und ausgeführt wird. Analog zum Hardware-Generator beinhaltet die Software den Algorithmus zur TAN-Generierung; um die Generierung zu starten, muss der Bankkunde das ihm über das E-Banking-Portal mitgeteilte Codewort eingeben.
  - *Verfeinerung der Maßnahmen:* Entwicklung einer stationären Java-Anwendung mit den identischen Teilmaßnahmen, die auch bei der Entwicklung des Prototypen zur Hardware-Generierung notwendig sind.
  - *Berücksichtigung von Abhängigkeitsbeziehungen und Ermittlung der Eigenschaften der Lösungsansätze:* Wie auch bei dem Hardware-Generator ist die einzige Verbindung zwischen der Software und dem restlichen E-Banking-System die manuelle Übergabe des Codewortes und der generierten TANs. Daher kann auch hier von einer ausreichenden losen Kopplung ausgegangen werden (Zwischenzielerreichung 100 %). Bei der Überprüfung des Software-Generators hinsichtlich der Sicherheit werden dieselben Szenarien angewendet, wie bei der Überprüfung des Hardware-Generators. Da auch beim Software-Generator ein Codewort benötigt wird, ist bei Diebstahl oder Verlust des Rechners keine missbräuchliche Nutzung möglich. Gefährlicher ist der Einsatz der Software-Generierung in einer unsicheren Umgebung, da die Eingaben des Bankkunden genau protokolliert werden können, z. B. durch einen Trojaner oder Keylogger. Die Szenarioanalyse zeigt, dass der Hardware-Generator sicherer als der Software-Generator ist. Die Software-Generierung kann zudem durch Kopieren leicht vervielfältigt und damit an unbefugte Personen verteilt werden kann (Zwischenzielerreichung 70 %). Der Implementierungsaufwand wird ebenfalls auf 10 Personentage geschätzt.
  - *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Da der Algorithmus zur TAN-Generierung nicht verändert wird, sind keine Veränderungen an den funktionalen Eigenschaften des E-Banking-Systems zu erwarten.
  - *Dokumentation des Entscheidungsverlaufs*: Auch diese Ergebnisse werden in Form eines Datenblattes dokumentiert. Im Gegensatz zur Hardware-Generierung ist in den folgenden zwei Entscheidungsstufen zu beachten, dass für den Prototyp die betriebssystemunabhängige Programmiersprache Java verwendet wird.

Bei der 'serverbasierten Generierung' stehen die folgenden beiden Lösungsansätze zur Verfügung:

- *Übermittlung per SMS:* Dabei wird die generierte TAN über ein Gateway zu einem Mobilfunkbetreiber übermittelt, welcher die TAN an das Mobiltelefon des betreffenden Bankkunden schickt.
  - *Verfeinerung der Maßnahmen:* Zunächst ist ein Gateway zu implementieren, damit der Applikationsserver des E-Banking-Systems die TANs per Kurznachricht (SMS) an das Mobiltelefon des Bankkunden schicken kann. In einem prototypischen Stadium kann dies durch die Anbindung eines einzelnen Mobiltelefons an das E-Banking-System realisiert werden. Des Weiteren ist eine Anwendung zu entwickeln, mit welcher Test-TANs generiert werden können.
  - *Berücksichtigung von Abhängigkeitsbeziehungen und Ermittlung der Eigenschaften*: Da die generierten TANs zunächst an ein Gateway übertragen und von dort aus an das Mobiltelefon des Bankkunden übermittelt werden, entsteht eine

stärkere Kopplung mit dem E-Banking-System. Das ist beispielsweise daran erkennbar, dass eine Vielzahl von Vermittlungsstellen involviert sind, um die Kurznachricht zu übertragen und einen Sendebericht an das E-Banking-System zurückzusenden. Alle diese Vermittlungsstellen müssen bei der Fehlerbehandlung und beim Testen berücksichtigt werden (Zwischenzielerreichung 80 %). Die Sicherheit des Verfahrens ist jedoch sehr hoch, da das Mobilfunkgerät des Bankkunden selbst durch einen PIN abgesichert ist und das gesamte Verfahren zur Generierung der TANs auf der Seite des E-Banking-Systems erfolgt (Zwischenzielerreichung 100 %). Der Implementierungsaufwand wird auf 5 Personentage geschätzt.

- *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Da die Übermittlung der TAN per Kurznachricht eine isolierte Funktion des E-Banking-Systems darstellt, sind keine Veränderungen an den funktionalen Eigenschaften des E-Banking-Systems zu erwarten.
- *Dokumentation des Entscheidungsverlaufs*: Auf dem Datenblatt wird neben diesen Ergebnissen dokumentiert, dass die Übertragung der SMS über die Netz-Infrastruktur eines Mobilfunkbetreibers erfolgt, was u. a. zusätzlichen Analyse- und Testaufwand erfordert.

- *Übermittlung an eine E-Mail-Adresse:* Beim zweiten Lösungsansatz der 'serverseitigen TAN-Generierung' wird die generierte TAN über einen Mail-Server an die E-Mail-Adresse des betreffenden Bankkunden geschickt. Die E-Mail ist dabei verschlüsselt und nur der betreffende Bankkunde verfügt über den notwendigen Schüssel, um den Inhalt der E-Mail im Klartext lesen zu können.
  - *Verfeinerung der Maßnahmen:* Zur Übermittlung der E-Mails wird ein Mailserver benötigt, wobei zum Betrieb des Prototypen auch ein bereits existierender Server zu Testzwecken eingesetzt werden kann. Der Applikationsserver ist so weit zu ergänzen, dass die generierte TAN in einer verschlüsselten E-Mail verpackt und an den Mailserver gesendet werden kann. Des weiteren ist eine Anwendung zu entwickeln, welche Test-TANs generiert.
  - *Einflussfaktoren*: Aus der Checkliste der Rahmenbedingungen sind keine Einflussfaktoren erkennbar.
  - *Ermittlung der Eigenschaften:* Da die generierten TANs zunächst in einer verschlüsselten E-Mail und an einen Mailserver übertragen und von dort aus versendet werden, entsteht eine stärkere Kopplung mit dem E-Banking-System. Die Kopplung ist jedoch geringer als bei der Übermittlung der TANs per Kurznachricht, da nur zwei Vermittlungsstellen benötigt werden: die Mailserver auf der Server- und der Clientseite (Zwischenzielerreichung 90 %). Da die E-Mail während der Übermittlung leicht abgefangen werden kann, existiert bei diesem Verfahren eine potenzielle Sicherheitslücke (Zwischenzielerreichung 90 %). Der Implementierungsaufwand auf 10 Personentage geschätzt, da die Verschlüsselung der E-Mails zusätzlichen Aufwand erfordert.
  - *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Da die Übermittlung der TAN per verschlüsselter E-Mail eine isolierte Funktion des E-Banking-Systems darstellt, sind keine Veränderungen an den funktionalen Eigenschaften zu erwarten.
  - *Dokumentation des Entscheidungsverlaufs*: Da dieser Lösungsansatz keine besonderen Eigenschaften beinhaltet, umfasst das Datenblatt nur die bereits genannten Analyseergebnisse.

Stufe 2 – Kapselung des Algorithmus zur TAN-Generierung

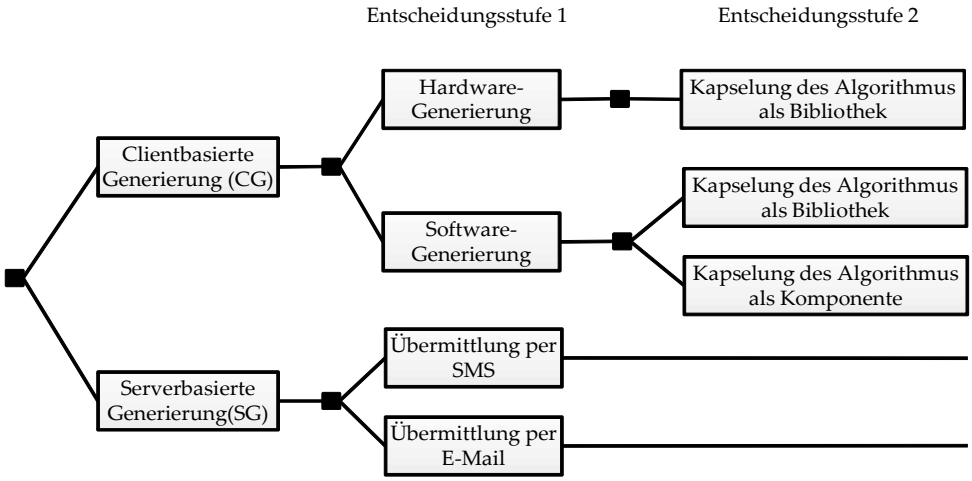


Abb. 10.7: Zweite Stufe der Architekturentscheidung

Im Falle der 'clientbasierten Generierung' sind nach dem in Kapitel 7 aufgeführten Katalog von Lösungsansätzen (siehe S. 139) die folgenden Ansätze für die Algorithmus-Kapselung in Betracht zu ziehen; dabei sind die Abhängigkeiten zu den Endgeräte-Alternativen der Hardware- und Software-Generierung zu beachten(siehe Abb. 10.7):

- *Bibliothek:* Entsprechend den Ausführungen in Abschnitt 7.4.3 (S. 155) ist eine Bibliothek ein Container, um Funktionalität in einer bestimmten Programmiersprache wiederverwendbar abzulegen. Hierbei wird der Algorithmus zur Generierung der TANs, auf der Grundlage benutzerspezifischer Parameter und des Codewortes, in einer Bibliothek implementiert.
  - *Verfeinerung der Maßnahmen:* Da die Durchführung der Kapselung der jeweiligen Bank obliegt, werden die Maßnahmen hierzu nicht im Detail betrachtet.
  - *Berücksichtigung von Abhängigkeitsbeziehungen:* Bei der Hardware-Generierung nur ein Codewort mit einer Ziffernfolge verwendet; dies muss bei der Kapselung des Algorithmus beachtet werden. Wird die Bibliothek im Zusammenhang mit der Hardware-Generierung eingesetzt, sind die eingeschränkten Hardwareressourcen, zu beachten, insbesondere im Hinblick auf den zur Verfügung stehenden Speicher.
  - *Ermittlung der Eigenschaften:* Zur Generierung der TANs werden die in der Bibliothek gekapselten Funktionen aufgerufen und es wird das Codewort als Parameter übergeben. Sofern das Schnittstellendesign der Bibliothek einen direkten Zugriff auf alle relevanten Funktionen ermöglicht, ist von einer ausreichenden losen Kopplung auszugehen. Jedoch ist der Zugriff auf eine Bibliothek grundsätzlich nur in derselben oder einer kompatiblen Programmiersprache möglich, in der auch die Funktionen in der Bibliothek implementiert sind (Zielerreichung 90 %). Die Abschätzung des Implementierungsaufwandes ist nicht erforderlich, da die Durchführung der Algorithmus-Kapselung der jeweiligen Bank überlassen ist.
  - *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Bei korrekter Portierung des bereits existierenden Algorithmus in eine gekapselte Form sind keine Veränderungen an den funktionalen Systemeigenschaften zu erwarten.
  - *Dokumentation des Entscheidungsverlaufs:* Da dieser Lösungsansatz keine spezifischen Eigenschaften beinhaltet, umfasst das Datenblatt nur die bereits genannten Analyseergebnisse.
- *Komponente:* Als Alternative zu einer Bibliothek kann der Algorithmus auch in Form einer Komponente gekapselt werden. Dies hat den Vorteil, dass der Zugriff auf die Komponente nicht von der Programmiersprache abhängig ist, in der die Funktionalität der Komponente implementiert ist (siehe Abschnitt 7.4.3 auf S. 155).

- *Verfeinerung der Maßnahmen:* Da die Durchführung der Kapselung der jeweiligen Bank obliegt, werden die Maßnahmen hierzu nicht im Detail betrachtet.
- *Berücksichtigung von Abhängigkeitsbeziehungen:* Für den Bereich der eingebetteten Systeme steht kein geeignetes Komponentenmodell zur Verfügung. Daher kann dieser Lösungsansatz nur im Zusammenhang mit der Software-Generierung verwendet werden.
- *Ermittlung der Eigenschaften:* Zur Generierung der TANs wird auf die Schnittstellen der Komponente zugegriffen, die den Zugriff auf die benötigte Funktionalität realisieren; das Codewort wird dabei als Parameter übergeben. Sofern das Schnittstellendesign der Komponente einen geeigneten Zugriff die relevanten Funktionen ermöglicht, ist von einer ausreichenden losen Kopplung auszugehen (Zielerreichung 100 %). Die Abschätzung des Implementierungsaufwandes ist nicht erforderlich, da die Durchführung der Kapselung der jeweiligen Bank überlassen ist.
- *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Bei korrekter Portierung des bereits existierenden Algorithmus in eine gekapselte Form sind keine Veränderungen an den funktionalen Systemeigenschaften zu erwarten.
- *Dokumentation des Entscheidungsverlaufs:* Da dieser Lösungsansatz keine spezifischen Eigenschaften beinhaltet, umfasst das Datenblatt nur die bereits genannten Analyseergebnisse.

Bei der 'serverbasierten Generierung' kann auf die Auswahl und Verfeinerung von alternativen Lösungsansätzen verzichtet werden. In diesem Fall ist durch die Bank zu gewährleisten, das Transaktionssystem um eine geeignete Schnittstelle erweitert wird, die den Zugriff auf den Algorithmus zur Generierung der TANs gewährleistet. Sofern die Schnittstelle einen direkten Zugriff auf die relevanten Funktionen ermöglicht, ist von einer ausreichend losen Kopplung auszugehen.

Stufe 3 – Lösungsansätze zur Codewort-Generierung und Gateway-Integration

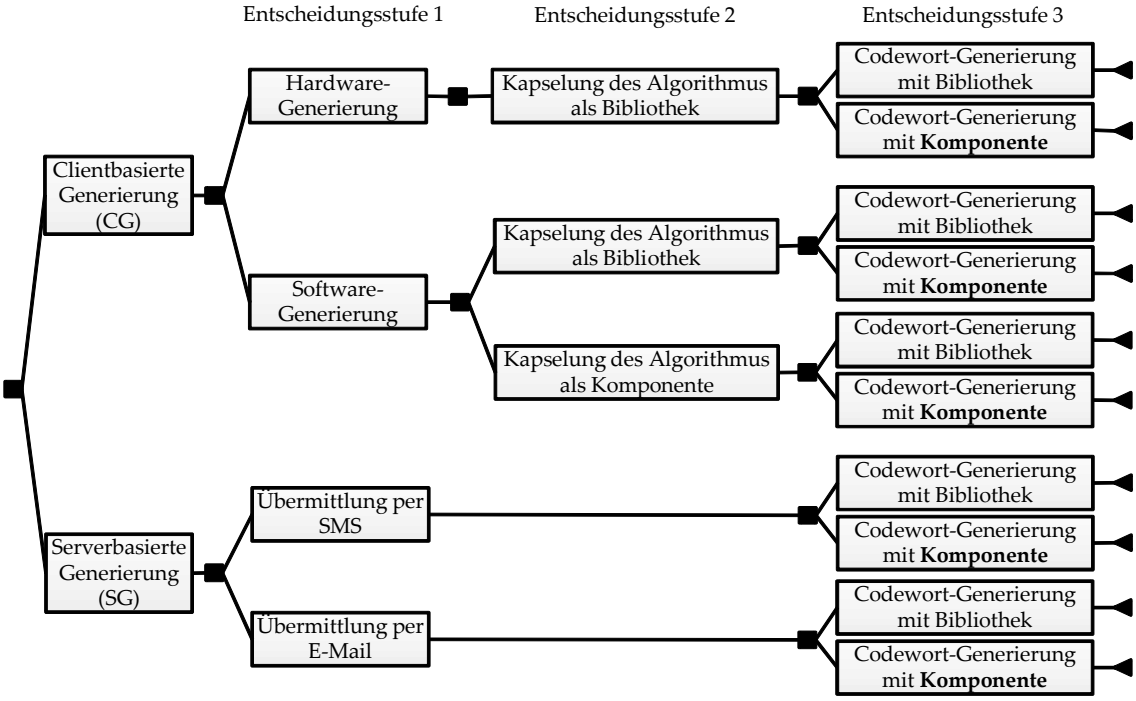


Abb. 10.8: Dritte und letzte Stufe der Architekturentscheidung

Im Falle der 'clientbasierten Generierung' sind – neben der abschließenden Entwicklung des Endgerätes – jene Lösungsansätze auszuwählen, mit denen die Generierung des Codewortes realisiert werden kann. Ein solches Codewort wird dann erzeugt, wenn der Bankkunde über das E-Banking-Portal den Auftrag eingegeben hat, z. B. die relevanten Daten für eine

Überweisung. Für die Erweiterung des Applikationsservers um diese Funktionalität stehen die Lösungsansätze 'Bibliotheken' oder 'Komponenten' zur Verfügung (siehe Abb. 10.8 auf vorangegangener Seite):

- *Bibliothek:* Bei diesem Lösungsansatz wird die Funktionalität zur Generierung eines Codewortes als zusätzliche Funktionsbibliothek implementiert (siehe für die bereits existierenden Funktionsbibliotheken Abb. 6.21, S. 123).
  - *Verfeinerung der Maßnahmen:* Um das Codewort zu generieren, wird bei diesem Lösungsansatz zum Applikationsserver eine zusätzliche Funktionsbibliothek hinzugefügt. Die Benutzeroberfläche des E-Banking-Portals ist um die dafür notwendigen Ein- und Ausgabefelder sowie Schaltflächen zu ergänzen.
  - *Berücksichtigung von Abhängigkeitsbeziehungen:* Bei diesem Lösungsansatz ist im Falle der Hardware-Generierung zu berücksichtigen, dass die Generierung der TAN in einem Endgerät erfolgt. Das Codewort ist auf eine Ziffernfolge beschränkt, da nur eingeschränkte Eingabemöglichkeiten in Form mehrfachbelegter Tasten von 0-9 zur Verfügung stehen. Bei der Implementierung dieser Stufe der Architekturentscheidung wird außerdem das Endgerät in einer finalen Version entwickelt. Dabei variiert der Implementierungsaufwand, da die Entwicklung des Hardware-Generators mehr Zeit in Anspruch nimmt, als die Entwicklung des Software-Generators.
  - *Ermittlung der Eigenschaften:* Mit der Implementierung der relevanten Funktionen zur Generierung des Codewortes mittels einer Bibliothek kann eine lose Kopplung zwischen dem Applikationsserver und der Codewort-Generierung erreicht werden (Zielerreichung 90 %). Grundsätzlich werden dabei die Durchführungszeiten je Transaktion erhöht, da zunächst das Codewort generiert und durch den Bankkunden in den Software- oder Hardware-Generator eingegeben werden muss, bevor die TANs generiert werden können (Zielerreichung aufgrund der Veränderung an den Transaktionszeiten 20 %). Der Implementierungsaufwand wird im Falle der Hardware-Generierung auf 110 Personentage geschätzt, da neben der Entwicklung der Bibliothek auch der Hardware-Generator zu konzipieren und zu entwickeln ist; für die Entwicklung des Software-Generators wird der Aufwand auf 80 Personentage geschätzt.
  - *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Da das existierende TAN-Listenverfahren während der Pilotphase parallel zum neuen TAN-Generierungsverfahren eingesetzt werden kann, ist nach Abschluss der Architekturveränderung explizit zu testen, ob beide Authentifizierungsverfahren fehlerfrei eingesetzt werden können.
  - *Dokumentation des Entscheidungsverlaufs:* Die Ergebnisse und vor allem die Hinweise bezüglich der Tests, die im Rahmen der Implementierung des Lösungsansatzes beachtet werden müssen, werden in Form des Alternativen-Datenblattes dokumentiert (siehe Abb. 10.9, S. 187).
- *Komponente*: Ein anderer Lösungsansatz ist die Entwicklung einer Komponente, die die relevanten Funktionen zur Codewort-Generierung umfasst. Der Vorteil dieses Lösungsansatzes liegt darin, dass für die Entwicklung der Komponente auch eine andere und eventuell effizientere Programmiersprache genutzt werden kann, als jene, in der der Applikationsserver und die bereits existierenden Funktionsbibliotheken implementiert sind.
  - *Verfeinerung der Maßnahmen:* Um das Codewort zu generieren, wird bei diesem Lösungsansatz das E-Banking-System um eine Komponente ergänzt; der Applikationsserver ist anzupassen, dass er auf die Funktionen der Komponente zur Codewort-Generierung zugreifen kann. Wie auch im vorangegangenen Lösungsansatz (siehe S. 184) muss die Benutzeroberfläche des E-Banking-Portals entsprechend angepasst.
  - *Berücksichtigung von Abhängigkeitsbeziehungen:* Es sind die identischen Abhängigkeiten zu berücksichtigen, wie im vorangegangenen Lösungsansatz (siehe S. 184).
  - *Ermittlung der Eigenschaften:* Auch bei diesem Lösungsansatz kann eine ausreichende lose Kopplung zwischen dem Applikationsserver und der Funktionalität zur Generierung des Codewortes erreicht werden (Zielerreichung 100 %). Analog zum vorangegangenen Lösungsansatz Bibliothek (siehe S. 184) werden aufgrund der zusätzlichen Generierungs-Schritte die Transaktionszeiten erhöht (Zielerreichung 20 %). Der Implementierungsaufwand wird im Falle der Hardware-Generierung auf 100 Personentage geschätzt, da neben der Entwicklung der Bibliothek auch der Hardware-Generator zu konzipieren und zu entwickeln ist. Im Falle des Software-Generators wird der Implementierungsaufwand auf 70 Personentage geschätzt.

- *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Analog zum vorangegangenen Lösungsansatz muss das existierende TAN-Listenverfahren während der Pilotphase parallel zum neuen TAN-Generierungsverfahren eingesetzt werden. Daher sind nach Abschluss der Architekturveränderung beide Authentifizierungsverfahren zu testen.
- *Dokumentation des Entscheidungsverlaufs:* Die Ergebnisse und vor allem die Hinweise bezüglich der Tests, die im Rahmen der Implementierung des Lösungsansatzes beachtet werden müssen, werden in Form des Datenblattes dokumentiert.

Neben der 'clientbasierten Generierung' kann die Generierung der TANs auch serverseitig erfolgen. Die Übertragung der TANs an den Bankkunden kann entweder über ein Mobilfunk-Gateway oder in verschlüsselter Form an per E-Mail erfolgen. Als Gateway kann entsprechend der Ausführungen in Kapitel 7 (S. 138ff.) ein Broker oder ein Vermittler eingesetzt werden:

- *Broker:* Die Verbindung zwischen Applikationsserver und dem Mobilfunk-Gateway oder dem Mailserver kann über einen Broker erfolgen. Der Broker agiert hierbei als Makler und überträgt die Nachrichten vom Applikationsserver aus kommend an die relevanten Stellen. Der Fokus des Brokers liegt dabei auf einem ausgewogenen Lastenausgleich. Ist beispielsweise ein Mailserver überlastet und kann daher die E-Mail nicht direkt versendet werden, wählt der Broker automatisch einen Mailserver mit noch freien Kapazitäten.
  - *Verfeinerung der Maßnahmen:* Der Broker wird als Komponente hinzugefügt, die die notwendige Funktionalität umfasst, um die vom Applikationsserver gesendete Nachricht an die entsprechenden Stellen weiterzuleiten. Im Falle der Nutzung des Mobilfunk-Gateways ist der Applikationsserver um eine Komponente zur Erzeugung einer Kurznachricht (SMS) zu ergänzen, welche die erstellte Kurznachricht an den Broker übermittelt. Im Falle der Nutzung der E-Mail als Übertragungsmedium wird der Applikationsserver um eine Komponente zur Erstellung und Verschlüsselung der E-Mail ergänzt, welche die erstellte Kurznachricht ebenfalls an den Broker übermittelt.
  - *Berücksichtigung von Abhängigkeitsbeziehungen:* Im Gegensatz zur 'clientbasierten Generierung' erfolgt die Generierung der TAN gekapselt im Transaktionssystem. Der Aufruf zur Generierung wird vom Bankkunden über das E-Banking-Portal des E-Banking-Systems initialisiert. Dem Applikationsserver obliegt die Aufgabe, die generierte TAN über eine der beiden Übermittlungswege an den Bankkunden zu übermitteln. Im Falle der Übermittlung per E-Mail ist eine zusätzliche Verschlüsselung erforderlich.
  - *Ermittlung der Eigenschaften:* Die zur Übermittlung der TANs notwendige Funktionalität ist zwar als Komponente im Applikationsserver integriert. Jedoch wird der Broker in der Praxis meist getrennt vom Applikationsserver auf einem eigenständigen Server implementiert. Dadurch wird die lose Kopplung nur zum Teil erfüllt (Zielerreichung 50 %). Da eine Codewort-Überprüfung nicht erforderlich ist, kann von einer identischen Durchführungszeit der Transaktionen ausgegangen werden (Zielerreichung 100 %). Der Implementierungsaufwand wird im Falle der Nutzung des Mobilfunk-Gateways auf 56 Personentage geschätzt, da keine weiteren Endgeräte zu entwickelt werden. Im Falle der Übermittlung per E-Mail wird der Implementierungsaufwand auf 66 Personentage geschätzt, da außerdem ein Verschlüsselungsverfahren zu implementieren ist.
  - *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Da die Funktionalität zur TAN-Generierung im Transaktionssystem der Bank gekapselt ist, beschränken sich die Veränderungen am E-Banking-System auf die Ergänzung des Applikationsservers, das Hinzufügen des Brokers die Einbindung der Gateways oder Mailserver. Es sind keine Veränderungen am funktionalen Verhalten zu erwarten, da das ursprüngliche Authentifizierungsverfahren nur minimal angepasst wird. Es ist für die Authentifizierung unerheblich, ob die TANs als Liste zur Verfügung stehen oder erst zur Laufzeit generiert werden.
  - *Dokumentation des Entscheidungsverlaufs:* Die Ergebnisse der Konkretisierung und Verfeinerung werden in der bekannten Form eines Datenblattes dokumentiert.
- *Vermittler:* Die Verbindung zwischen Applikationsserver und dem Mobilfunk-Gateway oder dem Mailserver kann auch über einen Vermittler erfolgen. Der Vermittler kann neben der vom Broker bekannten Funktion des Maklers

zusätzliche Funktionalität beinhalten. Daher wird bei diesem Lösungsansatz die Erzeugung der Kurznachricht bzw. die Erzeugung und Verschlüsselung der E-Mail durch den Vermittler realisiert. Die erzeugte Kurznachricht oder E-Mail wird im Anschluss an ein freies Mobilfunk-Gateway oder einen noch freien Mailserver gesendet.

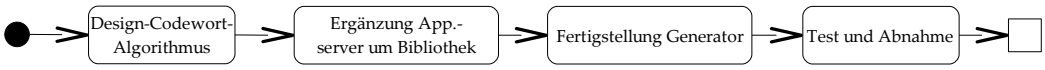
- *Verfeinerung der Maßnahmen:* Der Vermittler wird als neue Komponente zum E-Banking-System hinzugefügt. Er umfasst im Falle der Nutzung des Mobilfunk-Gateways die relevanten Funktionalitäten zur Generierung einer Kurznachricht. Im anderen Fall, bei dem eine E-Mail als Übertragungsmedium eingesetzt wird, umfasst der Vermittler die relevante Funktionalität um die E-Mail zu erstellen und zu verschlüsseln. In beiden Fällen erhält der Vermittler als Parameter vom Transaktionssystem die TAN. Daher ist auch der Applikationsserver so anzupassen, dass die vom Transaktionssystem empfangene TAN an den Vermittler weitergeleitet werden kann.
- *Berücksichtigung von Abhängigkeitsbeziehungen:* Es gelten die identischen Abhängigkeiten zum vorangegangenen Lösungsansatz Broker (siehe S. 185f.).
- *Ermittlung der Eigenschaften:* Da die Funktionalität zur Übertragung der TANs, entweder per E-Mail oder per Kurznachricht, vollständig im Vermittler implementiert ist, wird die erforderliche lose Kopplung zwischen dem Applikationsserver und dem Vermittler erreicht (Zielerreichung 100 %). Wie im vorangegangenen Lösungsansatz kann von einer identischen Durchführungszeit der Transaktionen ausgegangen werden (Zielerreichung 100 %). Der Implementierungsaufwand wird im Falle der Nutzung des Mobilfunk-Gateways auf 76 Personentage geschätzt. Im Falle der Übermittlung per E-Mail wird der Implementierungsaufwand auf 86 Personentage geschätzt, da beim Vermittler zusätzlich ein Verschlüsselungsverfahren zu implementieren ist.
- *Überprüfung und Reaktion auf Veränderungen funktionaler Systemeigenschaften:* Es sind die identischen Hinweise im Lösungsansatz Broker zu beachten (siehe S. 185f.).
- *Dokumentation des Entscheidungsverlaufs:* Die Ergebnisse werden wiederum in Form eines Datenblattes dokumentiert.

10.5.2 Beispiel für ein Datenblatt eines Lösungsansatzes

Lösungsansatz: Implementierung der Funktionalität zur Codewort-Generierung mittels Bibliothek

Hintergrund: Erfordert von Hardware/Software-Generator zur Erhöhung der Sicherheit. Im Falle der Hardware-Generierung ist zu berücksichtigen, dass die Generierung der TAN in einem Endgerät erfolgt. Das Codewort ist auf eine Ziffernfolge beschränkt, da nur eingeschränkte Eingabemöglichkeiten in Form mehrfachbelegter Tasten von 0-9 zur Verfügung stehen.

Maßnahmenfolge:



Eigenschaften des Lösungsansatzes:

Grad der Zielerreichung	Einhaltung loser Kopplung, erfüllt zu 90 % Beibehaltung Durchführungszeiten je Transaktion, erfüllt zu 20 %
Implementierungsaufwand	Hardware-Generierung: 110 Personentage Software-Generierung: 80 Personentage
Rahmenbedingungen	Erfüllt, Überprüfung bei Implementierung erforderlich
Veränderungen am funktionalen Verhalten	Möglich, Überwachung während Parallelbetrieb des alten und neuen Authentifizierungsverfahrens in Pilotphase

Abb. 10.9: Beispiel eines Datenblattes eines Lösungsansatzes

Literaturverzeichnis

[ADOD06] o.A.: *Databases Supported*, URL: 'http://phplens.com/adodb/supported.databases.html', Abruf 2006-07-10.

[Arc+06] Mehdi Archour, Friedhelm Betz, Antony Dovga: *PHP Handbuch*, URL: 'http://www.dynamicwebpages.de/php/index.php', Abruf 2006-06-15.

[Balz00] Helmut Balzert: *Lehrbuch der Software-Technik*, Spektrum, Heidelberg u.a., 2000.

[Bar+03] Mario R. Barbacci, Robert Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, William G. Wood: *Quality Attribute Workshops (QAWs), 2nd Edition*, URL: 'http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr019.pdf', Abruf 2007-09-20.

[Bas+02] Victor R. Basili, Gianluigi Caldiera und H. Dieter Rombach: *The Goal Question Metric Approach*. In: J.J. Marciniak: 'Encyclopedia of Software Engineering, Band 2', Wiley & Sons, New York (NY, USA) u.a., 1994, 528 - 532.

[BeBe07] Petra Becker-Pechau, Marcel Bennicke: *Concepts of Modeling Architectural Modules Views for Compliance Checks Based on Architectural Styles*, In: 'Proceedings of the 11th IASTED International Conference on Software Engineering and Applications (SEA 2007)', Acta Press, Cambridge (MA, USA), 2007, Seiten: 103-133.

[Beeg07] Robert F. Beeger: *Hibernate : Persistenz in Java-Systemen mit Hibernate und der Java Persistence API*, dpunkt, Heidelberg, 2007.

[Ben+00] PerOlof Bengtsson, Nico Lassing, Jan Bosch, Hans van Vliet: *Analyzing Software Architectures for Modifiability*, URL: 'http://www.bth.se/fou/forskinfo.nsf/7172434ef4f6e8bcc1256f5f00488045/45f87bb6d72ed63bc12568dd00506989/$FILE/Research%20report%2000-11.pdf', Abruf 2007-10-02.

[Ben+04] PerOlof Bengtsson, Nico Lassing, Jan Bosch, Hans van Vliet: *Architecture-level modifiability analysis (ALMA)*, Journal of Systems and Software - Volume 69 - Nummer 1-2 - Januar 2004, Seiten 129 - 147.

[Benc04] o.A.: *Benchmark results for the TYPO3 DBAL extension*, URL: 'http://projects.fishfarm.de/dbal/DBAL-Benchmark.pdf', Abruf 2007-08-04.

[Beva99] Nigel Bevan: *Quality in use: Meeting user needs for quality*, Journal of Systems and Software - Volume 49 - Nummer 1 - Dezember 1999, Seiten 89 - 96.

[Boeh78] Barry W. Boehm: *Characteristics of Software Quality*, North-Holland, Amsterdam (The Netherlands), 1978.

[Boo+06] Grady Booch, James Rumbaugh, Ivar Jacobson: *Das UML Benutzerhandbuch : aktuell zur Version 2.0*, Addison-Wesley, München u.a., 2006.

[Bosc00] Jan Bosch: *Design and use of software architectures : adopting and evolving a product-line approach*, Addison-Wesley, Harlow u.a., 2000.

[Bus+05] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad: *Pattern-Oriented Software Architecture - A System of Patterns, v. 1.*, Wiley & Sons, Chichester (USA), 2005.

[Bwba08] o.A.: *TAN-Generator der BW-Bank zertifiziert -Erfolg für sicheres Onlinebanking*, URL: 'http://www.bw-bank.de/bwbankde/1000005999-s1463-de.html', Abruf 2008-01-05.

[Cle+01] Robert T. Clemen, Terence Reilly: *Making hard decisions with Decision Tools Suite*, Duxbury Thomson Learning, Pacific Grove (CA), 2001.

[Cle+03] Paul Clements (Hrsg.), Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford: *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, Boston (MA, USA), 2003.

[Damb06a] Karsten Dambekalns: *TYPO3 4.0: DBAL Datenbankzugriff mal anders*, URL: 'http://www.yeebase.com/fileadmin/t3n/archiv/05-02/t3n_05-02_typo3_40_dbal.pdf', Abruf 2006-09-27.

[Damb06b] Karsten Dambekalns: *Database Abstraction Layer*, URL: 'http://typo3.org/documentation/document-library/extension-manuals/dbal/current/view/', Abruf 2006-09-10.

[Das+01] Eric M. Dashofy, André Van der Hoek, Richard N. Taylor: *A Highly-Extensible, XML-Based Architecture Description Language*, In: 'Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)', IEEE Computer Society, Washington (DC, USA), 2001, Seiten: 103 - 112.

[Doug99] Bruce Powel Douglass: *Doing Hard Time*, Addison-Wesley, Reading (Mass., USA), 1999.

[Ecke05] Claudia Eckert: *IT-Sicherheit Studienausgabe. Konzepte - Verfahren - Protokolle*, Oldenbourg, München, 2005.

[Eis+01] Franz Eisenführ, Thomas Langer, Martin Weber: *Fallstudien zu rationalem Entscheiden*, Springer, Berlin, 2001.

[Eis+07] Horst A. Eiselt, C.-L. Sandblom: *Decision Analysis, Location Models, and Scheduling Problems*, Springer, Berlin, 2007.

[EiWe03] Franz Eisenführ, Martin Weber: *Rationales Entscheiden : mit 59 Tabellen*, Springer, Berlin u.a., 2003.

[Fen+97] Norman E. Fenton, Shari Lawrence Pfleeger: *Software metrics : a rigorous and practical approach, 2nd Edition*, International Thomson Computer Press, London (UK), 1997.

[Fiel00] Roy Thomas Fielding: *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation vorgelegt an der University of California (Irvine, USA), 2000.

[Fish67] Peter C. Fishburn: *Methods of Estimating Additive Utilities*, Management Science - Volume 13 - Nummer 7 - Serie A - März 1967, Seiten 435 - 453.

[Fowl05] Martin Fowler: *Refactoring : Wie Sie das Design vorhandener Software verbessern*, Addison-Wesley, München u.a., 2005.

[Fowl06] Martin Fowler: *Alpha list of refactorings*, URL: 'http://www.refactoring.com/catalog/index.html', Abruf 2006-10-01.

[Gamm01] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Entwurfsmuster - Elemente wiederverwendbarerer objektorientierter Software*, Addison-Wesley, München u.a., 2001.

[Gar+00] David Garlan, Robert T. Monroe und David Wile: *Acme: Architectural Description of Component-Based Systems*. In: Gary T. Leavens und Murali Sitaraman: 'Foundations of component-based systems', Cambridge University Press, New York (NY, USA), 2000, 47 - 67.

[Garv84] David A. Garvin: *What does Product quality really mean?*, Sloan Management Review - Volume 4 - 1984, Seiten 25 - 43.

[Gie+06] Simon Giesecke, Wilhelm Hasselbring, Matthias Riebisch: *Classifying Architectural Constraints as a Basis for Software Quality Assessment*, Advanced Engineering Informatics - Volume Volume 21 - Issue 2 - April 2007, Seiten 169-179.

[Gur+03] Thorsten Gurzki, Henning Hinderer: *Eine Referenzarchitektur für Portalsoftware*, URL: 'http://www.swe.uni-linz.ac.at/teaching/lva/ws04-05/seminar/WM2003_Gurzki%20Referenzarchitektur%20Portalsoftware.pdf', Abruf 2007-01-04.

[Harn03] H. L. Harney: *Bayesian Inference: Parameter Estimation and Decisions (Advanced Texts in Physics)*, Springer, Berlin, 2003.

[Hof+00] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied software architecture*, Addison-Wesley, Harlow, Reading (Mass., USA), 2000.

[Horv06] Peter Horváth: *Controlling*, Vahlen, München, 2006.

[IEE610] Institute of Electrical and Electronics Engineers (IEEE): Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990, IEEE Computer Society, Piscataway (NJ, USA), 1983.

[IS8402] DIN EN ISO 8402: Qualitätsmanagement und Qualitätssicherung, Beuth Verlag Berlin, Augsburg, 1995.

[IS9126] DIN ISO 9126-2000: Informationstechnik - Beurteilen von Software-Produkten, Qualitätsmerkmale und Leitfaden zu deren Verwendung, Beuth Verlag, Berlin - Augsburg, 2000.

[Kaz+00] Rick Kazman, Mark Klein, Paul Clements: *ATAM: Method for Architecture Evaluation*, URL: 'http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf', Abruf 2007-10-01.

[Kaz+95] Rick Kazman, Len Bass, Mike Webb, Gregory Abowd: *SAAM: A Method for Analyzing the Properties of Software Architectures*, In: 'ICSE '94: Proceedings of the 16th international conference on Software engineering', IEEE Computer Society, Los Alamitos (CA, USA), 1995, Seiten: 81 - 90.

[Keri04] Joshua Kerievsky: *Refactoring to patterns*, Addison-Wesley, Boston (NY, USA) u.a., 2004.

[Klae04] Herbert Klaeren: *Konzepte höherer Programmiersprachen : Abschnitt koobj*, URL: 'http://www-pu.informatik.uni-tuebingen.de/users/klaeren/koobj.pdf', Abruf 2007-01-23.

[Kol+03] Elena Kolodizki, Markus Pizka, Martin Ober: *Refactoring in Langzeitprojekten - Fallstudie*, URL: 'http://www4.in.tum.de/%7Epizka/mp03i.pdf', Abruf 2007-10-15.

[Kru+06] Philippe Kruchten, Patricia Lago und Hans van Vliet: *Building Up and Reasoning About Architectural Knowledge*, 2nd International Conference on the Quality of Software Architectures (QoSA 2006) - Volume 4214 - 2006, Seiten 43-58.

[Kruc01] Philippe Kruchten: *The rational unified process - an introduction, 2nd Edition*, Addison-Wesley, Boston (MA, USA) - London (UK), 2001.

[Labo06] Kai Laborenz: *TYPO3 4.0 : das Handbuch für Entwickler*, Galileo-Press, Bonn, 2006.

[Lan+97] Filippo Lanubile, Giuseppe Visaggio: *Extracting reusable functions by flow graph based program slicing*, Software Engineering, IEEE Transactions on - Volume 23 - Nummer 4 - April 1997, Seiten 246 - 259.

[Lim04] John Lim: *ADOdb Database Abstraction Library for PHP (and Python).*, URL: 'http://adodb.sourceforge.net/', Abruf 2007-10-02.

[Lin+03] Mikael Lindvall, Roseanne Tesoriero Tvedt, Patricia Costa: *An Empirically-Based Process for Software Architecture Evaluation*, Empirical Software Engineering - Volume 8 - Nummer 1 - March 2003, Seiten 83 - 108.

[McC+77] Jim A. McCall, Paul K. Richards, Gene F. Walters: *Factors in software quality*, Rome Air Development Center, Springfield (VA, USA), 1977.

[Men+02] Tom Mens, Serge Demeyer, Dirk Janssens: *Formalising Behaviour Preserving Program Transformations*, In: 'Graph Transformation: First International Conference, ICGT 2002, Barcelona, Spain, October 7-12, 2002. Proceedings', Springer, Berlin - Heidelberg, 2002, Seiten: 286 - 301.

[Mrtm07] o.A.: *Hotelkette Maritim Homepage*, URL: 'http://www.maritim.de/typo3/', Abruf 2007-07-22.

[Omni08] o.A.: *MultiCash@Online*, URL: 'http://www.multicash.fr/ProduktSeiten/show_PDF.php?Land=DE&Bereich=MC%20Online', Abruf 2008-01-04.

[Opdy92] William Opdyke: *Refactoring Object-Oriented Frameworks*, Dissertation vorgelegt an der University of Illinois (Urbana-Champaign, USA), 1992.

[Pom+04] Gustav Pomberger, Wolfgang Pree: *Software Engineering. Architektur-Design und Prozessorientierung, Auflage: 3., vollst. überarb. Aufl.*, Hanser Fachbuchverlag, München, 2004.

[Pos+04] Torsten Posch, Klaus Birken Michael Gerdom: *Basiswissen Softwarearchitektur : verstehen, entwerfen, bewerten und dokumentieren*, dpunkt-Verlag, Heidelberg, 2004.

[Rap+97] Rapide Design Team: *Guide to the Rapide 1.0 - Language Reference Manuals*, URL: 'http://pavg.stanford.edu/rapide/lrms/overview.ps', Abruf 2007-09-10.

[Reim07] Helmut Reimer: *Resource Secured*, Datenschutz und Datensicherheit - DuD - Volume Volume 30 - Number 6 - Juni 2006, Seiten 381-389.

[Rip+07] Franz Ripfel, Melanie Meyer, Irene Höppner: *Das TYPO3-Profihandbuch. Der Leitfaden für Entwickler und Administratoren zu Version 4.1.*, Addison-Wesley, München, 2007.

[RoLi04] Stefan Roock und Martin Lippert: *Refactorings in großen Softwareprojekten : komplexe Restrukturierungen erfolgreich durchführen*, dPunkt-Verlag, Heidelberg, 2004.

[Romh98] Kai Romhardt: *Die Organisation aus der Wissensperspektive : Möglichkeiten und Grenzen der Intervention*, Gabler, Wiesbaden, 1998.

[Saat01] Thomas L. Saaty: *Decision making for leaders : the analytic hierarchy process for decisions in a complex world, New ed., 3. ed., 4. printing.*, RWS Publications, Pittsburgh (Pa, USA), 2001.

[Sas+95] David C. Sastry, Mettin Demirci: *The QNX Operating System*, Computer - Volume 28 - Nummer 11 - November 1995, Seiten 75 - 77.

[Sha+96] M. Shaw und D. Garlan: *Software architecture : perspectives on an emerging discipline*, Prentice Hall, Upper Saddle River (NJ , USA), 1996.

[She+01] Mark Shereshevsky, Habib Ammari, Nicholay Gradetsky, Ali Mili, Hany H. Ammar: *Information Theoretic Metrics for Software Architectures*, In: '25th Annual International Computer Software and Applications Conference (COMPSAC'01)', IEEE Computer Society, Los Alamitos (CA, USA), 2001, Seiten: 151 - 160.

[Sil+03] Silvia Rothen, Bernhard Angerer, Max Kleiner (Hrsg.): *Patterns konkret : Patterns in modellgetriebenen Projekten durchgängig anwenden*, Software & Support, Frankfurt am Main, 2003.

[Sim+99] Frank Simon, Silvio Löffler, Claus Lewerentz: *Metrics Based Refactoring*, In: 'Proceedings of the Fifth European Conference on Software Maintenance and Reengineering', IEEE Computer Society, Los Alamitos (CA, USA), 1999, Seiten: 30 - 38.

[Simo78] Herbert A. Simon: *Rationality as Process and as Product of Thought*, American Economic Review - Volume 68 - Nummer 2 - Mai 1978, Seiten 1 - 16.

[Skrh06] Kasper Skrhj: *Inside TYPO3*, URL: 'http://typo3.org/documentation/document-library/core-documentation/doc_core_inside/current/view/', Abruf 2006-10-10.

[Skrh06b] Kasper Skrhj: *TYPO3 Coding Guidelines*, URL: 'http://typo3.org/documentation/document-library/core-documentation/doc_core_cgl/current/view/', Abruf 2006-10-25.

[Spr+04] Günter Springer, Katharina Trippler: *Webauftritt aus einem Guss*, Ilmenauer Uni-Nachrichten - Volume 47 - Nummer 2 - März/April 2004, Seiten 5 - 6.

[Ste+02] von Johann H. von Stein, Holger G. Köckritz, Friedrich Trautwein: *E-Banking im Privatkundengeschäft* , Wissenschaft & Praxis, Sternenfels, 2002.

[Sun+01] Gerson Sunye, Damien Pollet, Yves Le Traon, Jean-Marc Jezequel: *Refactoring UML Models*, In: 'UML 2001 - The Unified Modeling Language.Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings', Springer, Berlin - Heidelberg, 2001, Seiten: 134 - 138.

[TReq06] o.A.: *System Requirements*, URL: 'http://www.typo3.com/System_Requirements.1241.0.html', Abruf 2006-10-10.

[Turc06] o.A.: *Turck MMCache for PHP*, URL: 'http://turck-mmcache.sourceforge.net/', Abruf 2006-07-04.

[Viss07] Nico de Visser: *CoCoNets MULTIVERSA Produkte als Basis für die integrierte, internationale E-Banking-Architektur*, URL: 'www.coconet.de/stepone/data/downloads/0a/00/00/praxisbericht_artesia_de.pdf', Abruf 2007-11-07.

[Wha+94] Cathleen Wharton, John Rieman, Clayton Lewis, Peter Polson: *The cognitive walkthrough method: a practitioner's guide*. In: Jakob Nielsen, Robert L. Mack: ' Usability inspection methods', John Wiley& Sons, New York (NY, USA), 1994, 105 - 140.

[Yan+99] George Yanbing Guo, Joanne M. Atlee, Rick Kazman: *A Software Architecture Reconstruction Method* , In: 'WICSA1: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)', Kluwer - B.V., Deventer (The Netherlands), 1999, Seiten: 15 - 34.

[Zele96] Gregory Zelesnik: *The UniCon Language Reference Manual*, URL: 'http://www-cgi.cs.cmu.edu/afs/cs/project/vit/www/unicon/reference-manual/Reference_Manual_1.html', Abruf 2007-10-01.

[Zim+08] Olaf Zimmermann, Thomas Gschwind, Jochen Kuester, Nelly Schuster: *Reusable Architectural Decision Models for Enterprise Application Development*, Lecture Notes in Computer Science - Volume 4880 - 2008, Seiten 15-32.

[Zimm05] Hans-Jürgen Zimmermann: *Operations Research : Methoden und Modelle*, Vieweg, Wiesbaden, 2005.

[Zimm97] Walter Zimmer: *Frameworks und Entwurfsmuster*, Shaker Verlag, Aachen, 1997.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalte der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zu Folge hat.

(Ort, Datum) (Unterschrift)

Thesen

(1) Eine systematische Entscheidungsfindung ist die Grundlage, um komplexe Architekturentscheidungen auch bei unvollständigen Informationen treffen zu können.

(2) Im Gegensatz zu Entwurfsentscheidungen sind Architekturentscheidungen abstrakter, weitreichender und dadurch komplexer sowie riskanter. Existierende Methoden und Konzepte zur Entscheidungsunterstützung fokussieren zu wenig auf die Spezifika von Softwarearchitekturen.

(3) Der Einsatz der Entscheidungstheorie ermöglicht eine nachhaltige Reduzierung des Risikos, der Unsicherheiten und der Komplexität von Architekturentscheidungen. Das generische Vorgehen aus der Entscheidungstheorie wurde um Methoden und Konzepte ergänzt, mit denen eine detaillierte Analyse und Bewertung der softwaretechnischen Eigenschaften der alternativen Lösungsansätze durchgeführt werden kann.

(4) Durch eine Dekomposition des Entscheidungsproblems und die isolierte Betrachtung der Bestandteile Ziele, Rahmenbedingungen, Architektureigenschaften und alternative Lösungsansätze können Widersprüche, Konflikte und Abhängigkeiten aufgelöst werden. Dadurch können die wesentlichen Ursachen von Architektur-Fehlentscheidungen behoben werden.

(5) Verbleibende Unsicherheiten über Zustände und Ereignisse werden im Entscheidungsprozess systematisch berücksichtigt, indem Varianten der alternativen Lösungsansätze gebildet werden, die nur unter bestimmten Wahrscheinlichkeiten eintreten. Die Ermittlung der Wahrscheinlichkeiten und die Bewertung der Varianten erfolgt durch spezielle Konzepte aus der Entscheidungstheorie.

(6) Um den Prozess der Entscheidungsfindung an die unterschiedlichen Risiko- und Komplexitätsstufen einer Architekturentscheidung anpassen zu können, stehen dem Entscheidungsträger obligatorische und optionale Design-, Analyse- und Bewertungstätigkeiten zur Verfügung.

(7) Durch den Einsatz des Entscheidungsprozesses werden alle wesentlichen Entscheidungsfaktoren, Annahmen und Einschätzungen dokumentiert. Der Entscheidungsträger kann damit zu einem späteren Zeitpunkt begründen und nachvollziehen, warum und wie eine Architekturentscheidung getroffen wurde.

Ilmenau, 25. März 2008