Teil I
Grundlagen und Organisation

1 Grundlagen

Wir sind alle stolz auf die Werkzeugmaschinen, die in den entlegensten Gebieten der Welt hohes Ansehen genießen. Deutsche Automobile zählen zu den Prestigeobjekten schlechthin. Aber man muss wissen, dass z. B. die schwäbische Werkzeugmaschine heute zu 70 % ihres Werts aus Mikroprozessoren und Software besteht. Die neuesten Autogenerationen, ob aus Wolfsburg, München oder Sindelfingen, sind in Wahrheit kleine Rechenzentren, die auch fahren können. Auch der hartnäckigste Computerhasser wird schon sehr bald fünf davon bedient haben, bevor er aus dem Bad kommt.

Erwin Staudt, Vorsitzender der Geschäftsführung IBM Deutschland.
Stuttgarter Zeitung, 29.09.2002

Warum benötigen wir eine Softwarearchitektur? Was genau ist Softwarearchitektur und welche Ziele verfolgt sie? Welche Auswirkungen gibt es, wenn wir in unseren Projekten keine explizite Softwarearchitektur haben? Woran erkennen wir eigentlich eine gute Softwarearchitektur?

Im ersten Kapitel dieses Buches wollen wir die grundlegenden Fragen zum Thema Softwarearchitektur klären. Der Leser bekommt ein deutliches Bild davon, was Softwarearchitektur ist, was sie leistet und wie sie uns in unseren Projekten helfen kann, bessere Software zu entwickeln. Somit wird die Basis gelegt, um in den anschließenden Kapiteln die verschiedenen Aspekte von Softwarearchitektur vertiefend zu besprechen.

Das Kapitel ist in drei Abschnitte eingeteilt. Im ersten Abschnitt wird dargestellt, warum Softwarearchitektur benötigt wird und wo ihre Wurzeln liegen. Der zweite Abschnitt führt aus, was exakt Softwarearchitektur ist. Dazu gehört eine Definition, die Ziele, die sie verfolgt, sowie die Faktoren, welche Softwarearchitektur beeinflussen. Der letzte Abschnitt geht nochmals ausdrücklich auf ihre Bedeutung ein und welche Symptome bei fehlender Softwarearchitektur auftreten.

1.1 Warum Softwarearchitektur?

Zunehmend komplexere Software

Unsere heutige Gesellschaft ist in einem hohen Maße abhängig von Software. Der Grad der Abhängigkeit nimmt dabei von Jahr zu Jahr zu. Verstärkt übernehmen Softwaresysteme Aufgaben, welche in vorhergehenden Produktgenerationen noch durch Hardware oder Mechanik gelöst wurden. Kommunikationsbusse ersetzen Verkabelungen und erhöhen den Grad der Vernetzung von Systemen. Displays ermöglichen komplexe grafische Benutzeroberflächen. Hinzu kommt, dass Software aufgrund der rasanten Entwicklung der Elektronik eine enorme Bandbreite an neuer, leistungsfähiger Funktionalität liefern kann.

Gesellschaft und Unternehmen sind heute von Software abhängig.

Fehler in Softwaresystemen können weitreichende, teils verheerende Auswirkungen haben. Dies reicht vom Verlust wichtiger persönlicher Daten über für eine Firma lebensbedrohliche Systemausfälle, bis hin zur Gefährdung zahlreicher Menschenleben. Eine Online-Bank, deren Webportal ausfällt, hat schlagartig alle ihre Filialen geschlossen. Ein Fehler in einem Flugsicherungssystem kann den Tod für viele Menschen bedeuten. Diese kritische Abhängigkeit unserer Gesellschaft von Software verlangt zugleich ein hohes Maß an Verantwortung von der Software-Entwicklungsgemeinde. Im gleichen Atemzug, indem Software der Schlüssel für die Funktionsfähigkeit von Produkten wird, wird sie auch das marktunterscheidende Element. Die Wettbewerbsfähigkeit eines Unternehmens entscheidet sich heute dadurch, ob dieses in der Lage ist, Softwaresysteme effizient und effektiv zu entwickeln. In der Automobilindustrie liegt bereits ein Großteil der Wertschöpfung in den elektronischen Systemen, wie z. B. Navigation oder Fahrsicherheit.

Ziele von Methoden der Softwareentwicklung

Um den stets wachsenden Anforderungen gerecht zu werden, verfolgen die *Methoden zur Softwareentwicklung* vor allem drei wesentliche Ziele: die Verkürzung der Entwicklungszeiten (engl. *time to market*), die Reduzierung der Wartungs- und Entwicklungskosten sowie die Verbesserung der Qualität von Software. Dieses *Spannungsdreieck* der Softwareentwicklung ist in Abb. 1–1 dargestellt. Neuere Methoden versuchen dies vor allem zu erreichen, indem sie den Grad der Wiederverwendung erhöhen und die Entwicklungsprozesse verbessern. Bekannte Vertreter hierfür sind strukturierte Analyse und Design sowie Objektorientierung. Letztere hat durch das Konzept der Klassen und Vererbung dazu beigetragen, den Grad der Wiederverwendung zu erhöhen und damit das Spannungsdreieck adressiert.

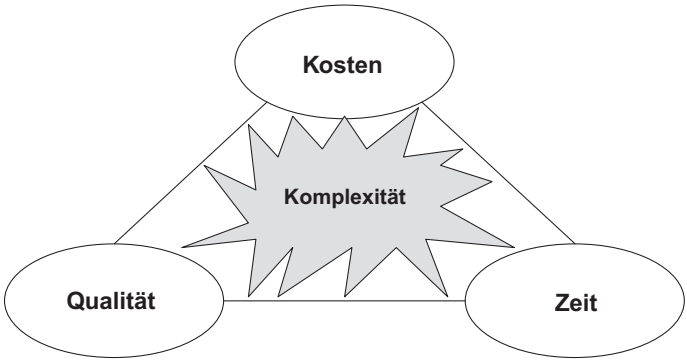

Abb. 1–1 Das Spannungsdreieck der Softwareentwicklung unter dem Druck der steigenden Komplexität ist die treibende Kraft der Methodenentwicklung.

Ursprung von Softwarearchitektur

Softwarearchitektur spielt in beiden Bereichen – der Wiederverwendung und der Prozessverbesserung – eine Schlüsselrolle. Sie beschreibt die Strukturen des Systems auf den obersten Abstraktionsebenen in Form von Bausteinen der Software mit deren Beziehungen. Softwarearchitektur entstand aus der Notwendigkeit heraus, immer größere und komplexer werdende Softwaresysteme zu beherrschen. Die gedanklichen Grundlagen zu diesem Gebiet wurden bereits zwischen 1960 und 1980 durch Parnas, Brooks, Dijkstra und andere gelegt. Deren Veröffentlichungen behandelten die Notwendigkeit zur sauberen Einteilung der verschiedenen Strukturen von Softwaresystemen sowie die Wichtigkeit der Partitionierung und Strukturierung von Softwaresystemen. Jedoch erst um 1990 herum, als große Systeme zunehmend die Regel wurden, fand das Thema weitreichende Anerkennung und Aufmerksamkeit in Forschung und Industrie. Erst in neuerer Zeit wurde für diese Thematik der Begriff der Softwarearchitektur geprägt.

Herausforderungen eines Softwarearchitekten

Softwarearchitekten sind ausgewiesene Personen in einem Projekt, die die Softwarearchitektur erstellen. Sie sind keine Entwickler – sollten aber, bevor sie zum Architekten wurden, Software entwickelt haben. Den Architekten obliegt die technische Verantwortung in der Softwareentwicklung. Softwarearchitekten sehen sich heute vor der Herausforderung, zunehmend komplexere Software bauen zu müssen und dabei die neuesten Technologien einzusetzen. Gleichzeitig müssen die Produkte bei hoher Qualität immer schneller auf den Markt kommen. Hinzu kommt, dass sich Technologien in einem rasanten Tempo ändern. Im Bereich des Internets hat sich zum Beispiel in nur wenigen Jahren eine komplett neue Art von Softwaresystem entwickelt. Die eingesetzten Technologien entwickelten sich anfangs von einfachen Skriptsprachen, wie HTML, bis zu komplexen Komponententechno-

logien wie J2EE. In gleichem Maße veränderten sich Strukturen und Mechanismen, nach denen solche Systeme gebaut wurden. Ständig musste Neuland betreten werden. Zugleich wurden die Systeme um ein Vielfaches komplexer: von einfachen statischen Webseiten einzelner Einrichtungen oder Privatpersonen, bis hin zu komplexen dynamischen Online-Portalen von Unternehmen. Neben der Beherrschung dieses Technologiewandels kommt hinzu, dass der Softwarearchitekt ein System so entwerfen muss, dass die heutigen Technologien durch zukünftige ersetzt werden können. Erfolgreiche Softwareprodukte haben die Fähigkeit, sich an die sich im Laufe der Zeit ändernden Anforderungen anzupassen. Dies gilt für Geschäftsanwendungen ebenso wie für technische Systeme.

Explizite Softwarearchitektur als mächtiges Werkzeug

Softwarearchitektur ist das mächtigste Werkzeug, um diesen gewachsenen Anforderungen standzuhalten. In der heutigen Software-Entwicklungspraxis sind die meisten Softwarearchitekturen jedoch implizit und nur wenige detailliert beschrieben. Softwarearchitektur muss jedoch als treibende Kraft im Mittelpunkt der Entwicklung stehen. Es ist gängige Praxis, erst zu implementieren und dann auszutesten, ob z. B. das System die geforderten Laufzeiten einhält. Softwarearchitektur befähigt Unternehmen bereits vor Implementierungsbeginn, fundierte Aussagen über die *Qualitätsmerkmale* und *Risiken* des Systems wie Laufzeiten, Robustheit oder Änderbarkeit zu treffen.

1.2 Was ist Softwarearchitektur?

Softwarearchitektur ist noch immer eine junge Disziplin. Eine einzelne, allgemein akzeptierte Definition gibt es nicht. In den Veröffentlichungen seit 1998 lässt sich jedoch erkennen, dass zunehmend ein Konsens herausgebildet wird. Hier spielen vor allem die Arbeiten des Software Engineering Institute der Carnegie Mellon Universität (SEI), Siemens Corporate Research sowie einzelner Autoren aus dem universitären und Beratungsbereich eine entscheidende Rolle. Bei unserer Beschreibung, was Softwarearchitektur ist, orientieren wir uns an diesen Arbeiten und bringen sie in einen gemeinsamen Kontext.

1.2.1 Definition von Softwarearchitektur

Aufbau der Definition

Im Folgenden werden wir den Begriff Softwarearchitektur definieren. Die *Definition* setzt sich aus mehreren Bestandteilen zusammen:

- Zeitliche Einordnung in den Entwicklungsprozess
- Festlegung der Aspekte von Software, die durch die Architektur beschrieben werden
- Abstraktionsebenen und Detaillierungsgrad von Softwarearchitektur
- Beantwortung der Frage, was eine gute Softwarearchitektur ist

Zeitliche Einordnung von Softwarearchitektur in die Entwicklungsphasen

Brücke zwischen Anforderungen und Implementierung

Nach Christine Hofmeister [Hofmeister00] bildet Softwarearchitektur die Brücke zwischen Anforderungsanalyse und Implementierung. Als solche kommt sie nach der Definition der Anforderungen und vor dem Feindesign, der Implementierung, Integration und dem Test. Dies ist eine grobe Abfolge der Aktivitäten aufgrund deren anfänglicher Abhängigkeiten zueinander. Reale Softwareentwicklung findet inkrementell in Iterationen statt. In den verschiedenen Iterationen wiederholen sich die Aktivitäten mit unterschiedlichen Schwerpunkten und können sich überlappen.

Abb. 1–2 Die Softwarearchitektur beschreibt die Strukturen des Systems und deren Beziehungen. Damit bildet sie die Brücke zwischen Anforderungen und Implementierung.

Welche Aspekte von Software beschreibt die Architektur?

Definition nach Len Bass: Strukturen, Schnittstellen, Verhalten und Sichten

Len Bass [Bass98] definiert Softwarearchitektur als die Strukturen des Systems, welche die *Architekturbausteine*, deren extern sichtbare Eigenschaften sowie die *Beziehungen* und *Interaktionen* zwischen diesen umfassen. Das *Verhalten* der Architekturbausteine ist Teil der Architektur, insoweit es aus der Sicht eines anderen Architekturbausteins sichtbar ist oder andere beeinflussen kann. Ein wesentlicher Bestandteil von Softwarearchitektur ist somit die Definition der

Schnittstellen der Architekturbausteine. Dabei bestehen Softwaresysteme nicht nur aus einer einzigen Struktur, sondern aus mehreren. Bei der Dokumentation von Softwarearchitektur werden diese Strukturen durch unterschiedliche *Sichten* (engl. *views*) dargestellt. Strukturen eines Softwaresystems können unter anderem die statische Struktur, die Prozessstruktur oder die physikalische Struktur sein. Zieht man als Vergleich den Bauplan eines Gebäudes heran, so hat der Maurer eine andere Sicht auf das Gebäude als der Elektriker. Beide interessieren sich vorrangig für unterschiedliche Strukturen des Gebäudes. Ein Gebäude hat noch zahlreiche weitere solcher Strukturen. Auch wenn diese Strukturen sehr unterschiedlich sind, beschreiben sie doch alle zusammen die Architektur des Gebäudes. Welche Strukturen für die Softwarearchitektur jeweils von Bedeutung sind, hängt zum Teil von dem zu entwickelnden System ab. Auf Strukturen und deren Sichten wird noch konkreter im Kapitel 6, »Dokumentation«, eingegangen.

Arten von Architekturbausteinen

Ein Softwaresystem ist aus unterschiedlichen Bausteinen aufgebaut, die sich je nach Abstraktionsebene in ihrer Komplexität unterscheiden. Auf der untersten Ebene kann bereits eine Funktion als Baustein bezeichnet werden. Abhängig vom Vorgehen schließt sich daran die Klasse oder das Modul an. Auf Architekturebene sind typische Bausteine Klassenstrukturen, Frameworks, Pakete, Komponenten oder Subsysteme. Häufig wird in Definitionen von Softwarearchitektur anstatt Architekturbaustein der Begriff *Komponente* verwendet. In diesem Zusammenhang ist das Gleiche gemeint. Der Begriff Komponente ist im Rahmen konkreter *Komponententechnologien*, wie J2EE oder .Net, inzwischen jedoch sehr eng festgelegt, so dass er für die Definition von Softwarearchitektur nicht mehr geeignet ist. Komponenten im Sinne von Komponententechnologien können eine Art von Architekturbaustein im Sinne der Definition von Softwarearchitektur sein.

Frühe Designentscheidungen und Softwarearchitekturdesign

Zusammenfassen lässt sich die Definition derart, dass Softwarearchitektur die Zerlegung des Systems in seine Hauptbestandteile auf der obersten Ebene ist. Sie definiert die Architekturbausteine, deren Verantwortlichkeiten, Instanzen, Schnittstellen und wie diese miteinander interagieren. *Softwarearchitekturdesign* ist der zugehörige Designprozess. Softwarearchitektur manifestiert somit die frühesten und wichtigsten Designentscheidungen für das Softwaresystem. Diese haben weitreichende Auswirkungen. Sie bestimmen zu einem Großteil, ob es dem Softwaresystem möglich sein wird, die gestellten Anforderungen zu erfüllen.

Ein Beispiel soll diese enorme Bedeutung der frühen Designentscheidungen verdeutlichen. Die Nachfolgegeneration eines Automobils wird mit einem Display im Cockpit ausgestattet. In diesem sol-

len Informationen vom Bordcomputer, Radio und Telefon dargestellt werden. In der bestehenden Softwarearchitektur sind diese drei Geräte bereits Architekturbausteine in Form von Subsystemen, die kaum Beziehungen zueinander haben. Die drei Subsysteme werden zudem von unterschiedlichen Zulieferunternehmen realisiert. Der Softwarearchitekt entscheidet sich nun, die grafische Ausgabe auf dem Display jedem Subsystem selbst zu überlassen, indem es direkt auf die Grafikbibliothek zugreifen kann. Um Konflikte zu vermeiden, erhält jedes Gerät seinen eigenen Bereich auf dem Display. Dieses Vorgehen erleichtert die Abstimmung zwischen den Zulieferfirmen und optimiert die ansonsten kritischen Laufzeiten für die Grafikausgabe. Das System funktioniert mit dieser Variante gut. Was jedoch übersehen wurde, ist, dass zukünftig noch ein Navigationssystem hinzukommen soll, das ebenfalls auf dem Display dargestellt wird. Alternativ muss zudem ein größeres Display in der Mittelkonsole genutzt werden können. Die bestehende Architektur bereitet für diese Änderungsanforderung gravierende Probleme. Das Navigationssystem benötigt das gesamte Display. Die einzelnen Geräte müssen somit ihr Konkurrenzverhalten untereinander abstimmen. Eine solche Koordination ist bisher nicht möglich, da jedes System autark arbeitet. Ebenso problematisch ist die alternative Ausgabe auf dem größeren Display. Zum einen ist auch hier wieder eine Koordination notwendig, und zum anderen wird eine abweichende Grafikausgabe benötigt. Da die einzelnen Geräte jedoch direkt auf die Grafikbibliothek des kleineren Displays zugreifen, ist dies mit größerem Änderungsaufwand verbunden. Hätte der Architekt die Änderungsanforderung von Beginn an berücksichtigt und die Display-Ausgabe in einen eigenen Architekturbaustein verlagert, mit dem alle beteiligten Geräte kommunizieren, wäre die Realisierung der Änderung um ein Vielfaches einfacher, da nur ein einzelner Architekturbaustein davon betroffen gewesen wäre.

Detaillierungsgrad und Abstraktionsebenen

Architektur abstrahiert von Details

Wie bereits erwähnt, beschreibt Architektur die Software auf den obersten Abstraktionsebenen. Softwarearchitektur abstrahiert somit und versteckt Details. Sie betrachtet nicht das Innenleben von Architekturbausteinen. Klassen und Algorithmen sind nur dann Bestandteil einer Architektur, wenn sie sich an den Grenzen von Architekturelementen befinden, wenn sie also beschreiben, wie diese untereinander, mit der Außenwelt oder der Ausführungsplattform interagieren [Hofmeister00]. Existiert z. B. ein Baustein, der Dienstleistungen wie Sortieren oder Suchen anbietet, so sind die Sortier- und Suchalgorithmen nicht Architekturbestandteil. Teil der Architektur sind jedoch die

Schnittstellen, über welche die Dienstleistungen bedient werden. Dennoch muss Softwarearchitektur genug Informationen beinhalten, um als Basis für Analyse, Bewertung, Entscheidungsfindung, Risikominimierung und Projektmanagement dienen zu können.

Maßstab: Systemgröße, Qualitätsmerkmale und Gewissheit

Welcher *Detaillierungsgrad* soll somit erreicht werden? Jan Bosch [Bosch00] definiert den Detaillierungsgrad einer Softwarearchitektur als Funktion der Größe des Systems, der Qualitätsmerkmale und dem Maß an Gewissheit, das erreicht werden soll. Bei großen Systemen wird es für die Softwarearchitektur nicht möglich sein, in die Herausforderungen einzelner Teile einzudringen. So können Sie nicht die Gesamtarchitektur eines Airbus entwerfen und gleichzeitig die Laufzeiten für die Grafikausgabe der LCD-Displays am Sitzplatz des Passagiers berücksichtigen. Dennoch sind die Qualitätsmerkmale, wie Leistung, Laufzeiten, Sicherheit, Robustheit, Änderbarkeit usw., der wichtigste Faktor, um den Detaillierungsgrad zu bestimmen. Das Ziel von Softwarearchitekturdesign ist es, mit einem ausreichenden Maß an Gewissheit eine Softwarearchitektur zu entwickeln, die alle Anforderungen inklusive der Qualitätsanforderungen erfüllen kann. Existieren somit kritische, anspruchsvolle Qualitätsanforderungen muss Softwarearchitekturdesign in eine beachtliche Detailtiefe gehen, um kritische Teile des Systems zu betrachten. Nehmen wir wieder unseren Airbus. Ausfallsicherheit stellt hier für viele Teile der Software ein Qualitätsmerkmal dar, für welches Sie ein hohes Maß an Gewissheit in der Architektur erreichen müssen. In einem gewissen Maße können Sie die Ausfallsicherheit z. B. durch Redundanz bereits auf der obersten Ebene einbauen. Jedoch ist es hier auch wahrscheinlich, dass Sie einzelne Entscheidungen bis auf Codeebene überprüfen müssen. So können *Architekturvorgaben* für Ausnahmebehandlung oder Speichermanagement gemacht werden, die direkt in Form von Kodierungsvorlagen bereitgestellt werden. Eventuell müssen auch Laufzeiten von konkurrierenden Prozessen in Form eines Prototyps überprüft werden, so dass es zu keinem Systemausfall kommen kann.

Detaillierungsgrad variiert je nach Risiko.

Der Detaillierungsgrad muss nicht für die gesamte Architektur gleich sein, sondern kann je nach *Risiko* für einzelne Bereiche variieren. Entscheidend ist es somit sicherzustellen, dass es der Software mit den durch die Softwarearchitektur festgelegten Strukturen und Mechanismen möglich ist, die Anforderungen zu erfüllen. Zu beachten ist, dass dies aber noch keine Garantie dafür ist, dass alle Anforderungen auch wirklich erfüllt werden. Schlechtes Feindesign und schlechte Implementierung können dies ebenso verhindern.

Projektmanagement hat Einfluss auf Detaillierung.

Auch andere *Einflussfaktoren*, wie z. B. solche aus dem Projektmanagement, spielen eine Rolle für den Grad der Detaillierung. Dem

Projektmanager muss es möglich sein, auf Basis der Softwarearchitektur den Projektplan für die Entwicklung und somit die Aufteilung von Arbeitspaketen auf Entwickler und Teams planen zu können. Hierfür benötigt er ein gewisses Maß an Zerlegung.

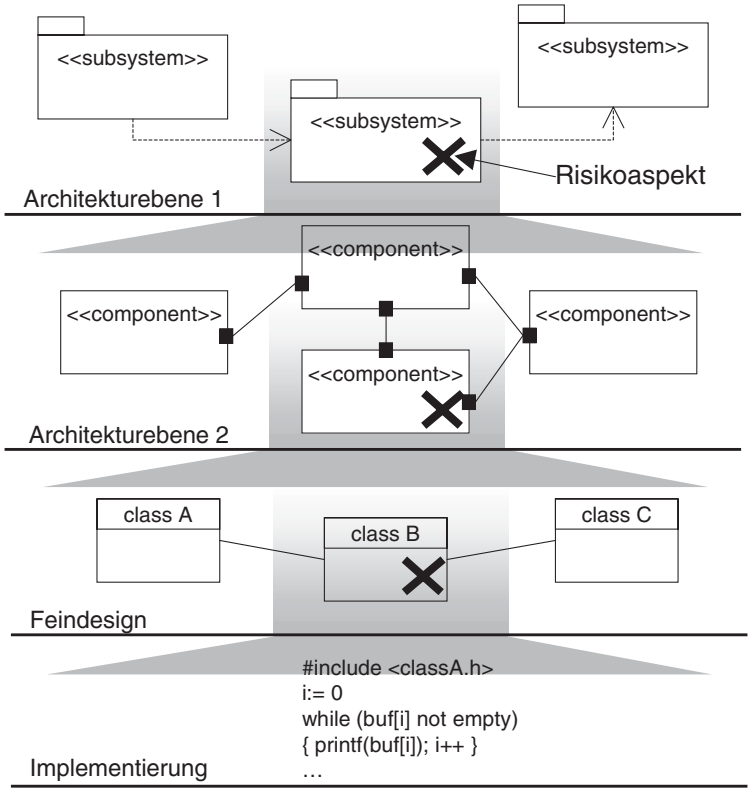

Abb. 1–3 Architektur bewegt sich auf den oberen Ebenen. Einzelne Risikoaspekte müssen in tiefere Ebenen hinein beleuchtet werden.

Kompakt: Was bestimmt den Detaillierungsgrad?

Zusammenfassend lässt sich sagen, Softwarearchitektur bewegt sich auf der Ebene der Hauptbestandteile von Software – den Architekturbausteinen – und beschreibt deren Schnittstellen, Beziehungen, Verhalten und Instanzen. Sie muss aber auch sicherstellen, dass die Architekturbausteine die von ihnen geforderten Aufgaben und Qualitätsmerkmale erfüllen können. Wenn diese Sicherheit für bestimmte Bereiche nicht gegeben ist, muss dementsprechend intensiver in die Details des Bausteins vorgedrungen werden. Dem Projektmanager muss die Softwarearchitektur die Planung der Arbeitspakete ermöglichen. Alle weiteren Details gehören in den Aufgabenbereich des Feindesigns und der Implementierung.

Architekturebenen und gewachsene Unternehmen

Für sehr große Systeme existieren nicht nur eine *Architekturebene* und eine Ebene des Feindesigns. Hier haben wir es mit mehreren Hierarchieebenen zu tun. So werden z. B. auf der obersten Ebene Subsysteme, deren Beziehungen, Verhalten und Instanzen definiert. Es handelt sich also laut unserer Definition um eine Architektur. Alles, was darunter liegt, ist aus Sicht dieser Ebene streng genommen Feindesign. Die einzelnen Subsysteme sind jedoch so groß, dass in diesen nochmals eine Ebene mit Subsystemen definiert wird. Aus Sicht dieser Ebene handelt es sich wieder um Architektur. Schritt für Schritt kommen wir der untersten Ebene aus Klassen und Methoden näher. In der Praxis wird es nun kaum möglich sein, von der obersten Ebene so weit ins Detail zu gehen, um sicherzustellen, dass die Subsysteme dieser Ebene ihre Anforderungen garantiert erfüllen können. Hier spielt die Erfahrung, das Wissen des Unternehmens eine entscheidende Rolle. Kein Unternehmen wird ein solches System bauen, ohne entsprechende Erfahrungen auf diesem Gebiet, in dieser Domäne, zu haben. Firmen, die solch große Systeme bauen, sind über lange Zeit gewachsen. Sie haben das entsprechende Wissen, wie deren Architekturen funktionieren. Ist dies nicht der Fall, müssen sie eine sehr aufwändige und teure Entwicklung durchlaufen. Oder können Sie sich als Spezialist für Datenbanken vorstellen, morgen die gesamte Software eines Airbus zu planen und zu bauen?

Was ist eine gute Softwarearchitektur?

Eine gute Architektur ermöglicht es einem speziellen System, seine Anforderungen zu erfüllen!

Um die Definition von Softwarearchitektur abzurunden, stellt sich noch die Frage, was eine gute Softwarearchitektur ist? Was ist die richtige Softwarearchitektur? [Bass98] definiert eine gute Architektur als solche, die es einem System ermöglicht, seine Verhaltens-, Qualitäts- und Lebenszyklusanforderungen zu erfüllen. Eine schlechte Architektur ist demnach eine, die ein System daran hindert. Es gibt somit keine generell richtige Architektur. Eine Softwarearchitektur ist mehr oder weniger fit für das gegebene Problem. Sie kann im Kontext konkreter Ziele bewertet werden.

Beispiel: Gleiche Funktionalität, zwei verschiedene Architekturen

Als Beispiel soll eine Datenbankanwendung dienen. Über die reine Funktionalität, welche die Datenbank leisten soll, wird schnell Einigkeit zwischen den Beteiligten hergestellt werden können. Der Architekt muss dann auf Basis der geforderten Funktionalität eine Architektur für die Datenbank entwerfen. Wie bei vielen Softwarelösungen kann er aber auch hier in seinen Entwurfsentscheidungen abwägen, ob das System geringere Laufzeiten – also kürzere Zugriffszeiten – oder weniger Speicher beanspruchen soll. Im ersten Fall mit den schnelleren Zugriffszeiten wird er Daten nach Bedarf mehrfach halten sowie

Berechnungen bereits im Vorfeld durchführen und deren Ergebnisse speichern. Dies benötigt Speicherplatz. Im zweiten Fall verzichtet er auf die doppelte Datenhaltung und Vorausberechnung. Dadurch verringert sich der Speicherplatzbedarf. Im gleichen Maße werden längere Zugriffszeiten nötig. Je nach Anwendung der Datenbank kann nun die eine oder andere Architekturlösung die richtige sein. Entscheidend neben den funktionalen Anforderungen sind die zusätzlichen Verhaltens- und Qualitätsanforderungen, die an die Datenbank gestellt werden. Muss sie z. B. nur wenige Daten halten und in einem eingebetteten System, wie einem Mobiltelefon, laufen, so wird man vorsichtiger mit dem Speicherbedarf umgehen. Die Zugriffszeiten sind in diesem Fall weniger kritisch. Eine Architekturlösung, die dies berücksichtigt, wäre somit eine gute Architektur. In einem anderen Fall haben wir eine Datenbank auf einem Server mit verteilten Zugriffen auf sehr viele Daten. Dort werden eher die Zugriffszeiten der kritische Aspekt sein. Speicher spielt auf unserem Server keine Rolle. Eine völlig andere Architekturlösung, wie im ersten Fall, die dies berücksichtigt, ist eine gute Architektur. Umgekehrt, hätten wir für unsere verteilte Datenbank eine wunderschöne, Speicher sparende Architektur entworfen, wäre dies sicherlich mit eine der schlechtesten denkbaren Lösungen geworden. Im Kapitel 4, »Einflussfaktoren«, und Kapitel 7, »Bewertung«, werden wir genauer darauf eingehen, wie Sie eine für Ihr spezielles System gute Softwarearchitektur entwerfen.

Allgemeine Kriterien für eine gute Architektur

Über diese Kriterien hinaus gibt es noch allgemeine Ziele und Aufgaben, die jede Architektur erfüllen muss. Solche allgemeinen Ziele werden im nächsten Abschnitt erläutert. Allgemeine Ziele im Spezielleren sind aber auch Kriterien für einen guten Entwurf oder eine gute Dokumentation. So setzt sich eine gut entworfene Architektur aus wohldefinierten Abstraktionsschichten zusammen; und eine gute Dokumentation ist aus Sicht des potenziellen Lesers geschrieben. Wie ich zu einer guten Softwarearchitektur komme, werden wir Schritt für Schritt in den weiteren Kapiteln des Buches genauer beschreiben.

1.2.2 Ziele und Aufgaben von Softwarearchitektur

Vier Ziele der Softwarearchitektur

Was will Softwarearchitektur erreichen? Was muss sie leisten und wie bringt sie dies zuwege? Um diese Fragen zu beantworten, betrachten wir zunächst vier wesentliche Ziele, die eine Softwarearchitektur verfolgt:

- Das Entwicklungsprojekt effizienter gestalten.
- Risiken minimieren durch frühes Berücksichtigen der Einflussfaktoren.
- Verständnis schaffen bei allen Beteiligten.
- Kernwissen über das System konservieren.

Effizientere Entwicklung

Wie kann Softwarearchitektur die Effizienz des Entwicklungsprojekts steigern? Indem sie an drei Punkten ansetzt. Punkt eins ist, dass Softwarearchitektur das Rückgrat eines jeden *iterativ inkrementellen Entwicklungsprozesses* darstellt. Ohne sie ist es nicht möglich, eine Software wirtschaftlich, inkrementell zu entwickeln. Sie stellt den notwendigen *Integrationsrahmen*, in dem die Software heranwachsen kann [Kruchten00]. Stellen Sie sich vor, ein System inkrementell entwickeln zu wollen, ohne das Ganze schon angedacht zu haben. Dies würde bedeuten, dass mit jeder neuen Anforderung, die inkrementell eingebaut wird, die Architektur zurechtgebogen werden muss. Ab einer bestimmten Systemgröße würden Sie unendlich viel Zeit für die Überarbeitung benötigen. Der zweite Punkt ist, dass Softwarearchitektur die Grundlage der Projektplanung und des Projektmanagements bildet. Insbesondere ist sie die Basis für Projektorganisation. Den Architekturbausteinen können Teams zugeordnet werden und den Teams wiederum Arbeitspakete. Mit Hilfe der Softwarearchitektur ist es dem Projektleiter möglich, sein Projekt aktiv zu führen, da sie ihm die essenziellen Informationen liefert. Da Architektur abstrahiert, gibt sie dem Projektmanager Einblick in die Entwicklung auf dem von ihm benötigten Abstraktionsniveau. Dies erleichtert es festzustellen, in welchen Bereichen technische oder zeitliche Probleme auftreten, so dass entsprechende Gegenmaßnahmen eingeleitet werden können. Zudem ist die Architektur das Artefakt, auf dessen Basis Änderungen verhandelt und gemanagt werden können. Der dritte wesentliche Punkt ist, dass Softwarearchitektur das Fundament bildet, das effektives, unabhängiges, verteiltes Arbeiten der Entwicklungsteams ermöglicht. Softwarearchitektur definiert Struktur, deren Beziehungen, Verhalten und Schnittstellen. Entwicklungsteams werden auf Basis dieser Struktur zusammengestellt. Softwarearchitektur enthält alle Informationen, damit diese Teams unabhängig voneinander in ihren Bereichen arbeiten können. Softwarearchitektur ist der Rahmen für die Implementierung.

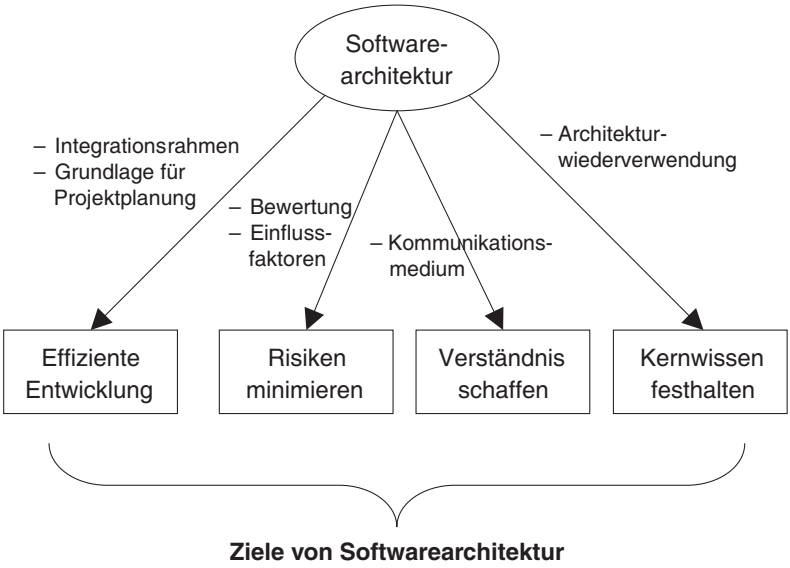

Abb. 1–4 Softwarearchitektur verfolgt vier Ziele. Die wesentlichen Aspekte hinter den Zielen sind an den Pfeilen notiert.

Risiken minimieren

Das zweite Ziel von Softwarearchitektur ist es, *Risiken* der Entwicklung zu minimieren. Dies erreicht sie, indem Einflussfaktoren früh berücksichtigt werden. Einflussfaktoren sind Anforderungen, Umstände oder Tatsachen, die den Entwurf der Architektur maßgeblich beeinflussen. Arten von Einflussfaktoren sind z. B. Qualitätsanforderungen wie Sicherheit oder auch Managementanforderungen wie Zeit- und Kostenvorgaben. Softwarearchitekturdesign richtet sich nach diesen Einflussfaktoren. Das heißt, es werden bewusst Entscheidungen getroffen, welche die Einflussfaktoren berücksichtigen. Denken Sie an unser Datenbankbeispiel aus dem vorherigen Abschnitt und den Abwägungen zwischen den Qualitätsanforderungen für Speicher und Laufzeit. Die Softwarearchitektur ist somit das erste Artefakt, das ausdrückt, wie die Software ihre Anforderungen erreichen will. Dementsprechend kann sie auch mit Blick auf die Einflussfaktoren analysiert und bewertet werden. Es ist also möglich vorherzusagen, ob eine Softwarearchitektur in der Lage ist, den gestellten Anforderungen gerecht zu werden. Wäre dies nicht möglich, könnte man jede beliebige Architektur wählen. Dies werden wir im Kapitel 7, »Bewertung«, noch ausführlicher darstellen.

Verständnis schaffen

Verständnis schaffen bei allen Beteiligten ist ein weiteres wichtiges Ziel. Softwarearchitektur dient als *Kommunikationsmedium* zwischen den *Stakeholdern* und insbesondere zwischen allen Mitarbeitern des Entwicklungsprojektes. Unter den Stakeholdern werden alle Personen

verstanden, die ein Interesse an dem System haben. Dies können sein der Kunde, Manager, Benutzer, Architekten, Entwickler, Tester, Qualitätssicherung, Marketing, Wartungsmitarbeiter usw. Auf Basis der Architektur können unterschiedliche, teils widersprüchliche Forderungen ausgedrückt, verhandelt und gelöst werden. Sie ist die Basis für Schulungen von neuen Projektmitgliedern. Somit dient sie allen Beteiligten als Leitbild und Referenz. Die verschiedenen Sichten auf eine Architektur ermöglichen es, dass viele Beteiligte mitreden können. Ist die einzige Sicht auf eine Software der Quellcode, so wird ein großer Personenkreis bei der Diskussion ausgeschlossen, da der notwendige Einblick für ein Verständnis verwehrt bleibt. Auch ein Feindesign ist meistens schon viel zu technisch und zu detailliert für eine Diskussion unter den Stakeholdern. Finden solche Diskussionen deshalb nicht statt, birgt dies eine große Gefahr, dass wesentliche Aspekte beim Entwurf des Systems übersehen werden. Hier gilt das Prinzip: Vier Augen sehen mehr als zwei!

Kernwissen konservieren

Ein letztes Ziel, welches Softwarearchitektur verfolgt, ist das *Kernwissen* des Systems zu konservieren. Als übertragbares Modell kann sie eine enorme Hebelwirkung für Systeme mit ähnlichen Anforderungen haben. Dies geht im Idealfall bis hin zur Entwicklung einer Produktlinie mit einer gemeinsamen Basisarchitektur. Es findet *Wiederverwendung* auf Ebene der Architektur statt (engl. *architecture reuse*). Architektur wird dadurch zu einem wertvollen geistigen Eigentum des Unternehmens. Entsteht bei der Entwicklung keine Architektur, sondern nur Quellcode, so ist eine Übertragung von Wissen auf zukünftige Systeme sehr schwierig und auf jeden Fall nur dann möglich, wenn die gleichen Personen, die das Wissen in ihren Köpfen gespeichert haben, in dem neuen Projekt wieder zum Einsatz kommen. Dies ist aber oftmals nicht möglich, da dieser Personenkreis noch in einem großen Maße mit der Einführung und Wartung des alten Systems beschäftigt ist, während die Grundlagen für das neue System bereits gelegt werden. Es gibt zahlreiche weitere Gründe, warum andere Personen das neue System entwickeln. Liegt keine Architektur vor, können diese nicht aus dem bereits entwickelten System lernen, da eine Einarbeitung in die Strukturen und Beziehungen des Quellcodes viel zu aufwändig ist. Die Folge ist, dass die Lernkurve, wie ein solches System funktioniert, in vielen Punkten erneut durchlaufen werden muss. Mit einer expliziten Softwarearchitektur lernen Unternehmen schneller!

Wege zum Ziel

Wie erreicht Softwarearchitektur diese Ziele? Auf welche Weise versucht Softwarearchitektur, die oben beschriebenen Ziele zu verwirklichen? Im Wesentlichen wendet sie dazu drei Mittel an:

- Sie ist ein konkretes Artefakt. Entscheidungen werden bewusst getroffen und explizit gemacht.
- Sie definiert ein Gerüst des Systems. Dieses abstrahiert und legt die frühesten Designentscheidungen fest.
- Sie wird zielgerichtet dokumentiert.

Entscheidungen bewusst treffen

Der erste Schritt wird bereits dadurch getan, dass Softwarearchitektur ein definiertes *Artefakt* im Projekt darstellt. Unter einem Artefakt versteht man ein physisches Ergebnis, das im Entwicklungsprozess definiert ist. Artefakte können Dokumente wie Anforderungsspezifikation, Architektur und Benutzerhandbuch oder auch Quellcode, Binärdateien, UML-Modelle und Ähnliches sein. Anders ausgedrückt bedeutet dies, Architekturdesign wird mit Zeit, Ressourcen und Kosten eingeplant. Dadurch werden Entscheidungen bewusst getroffen, kommuniziert und diskutiert.

Ein Gerüst festlegen

Die Entscheidungen, die im Rahmen von Architekturdesign getroffen werden, definieren ein Gerüst des Systems. Dieses Gerüst legt die frühesten Designentscheidungen fest und abstrahiert von Details. Es stellt eine technische Blaupause des Systems dar. Softwarearchitektur ist keine umfassende Zerlegung oder Verfeinerung des Systems. Viele Implementierungsdetails sind abstrahiert und gekapselt in Elemente der Architektur. Die frühen Designentscheidungen sind die, welche am schwierigsten korrekt zu erstellen und später am schwierigsten zu ändern sind. Sie haben die am weitesten reichenden Auswirkungen für das Projekt. Denken Sie an unser Datenbankbeispiel. Will der Architekt nach der Fertigstellung der Implementierung das System von Speicher schonend auf Laufzeit optimiert umstellen, würde dies höchstwahrscheinlich einen kompletten Umbau des Systems bedeuten. Diese Kosten wären nicht tragbar! Des Weiteren bestimmen die frühen Designentscheidungen die organisatorischen Strukturen des Projekts, ermöglichen oder hemmen bestimmte Qualitätsmerkmale des fertigen Systems und definieren Einschränkungen für die Implementierung. Die Langlebigkeit des Systems, wie es unter dem Druck der Evolution besteht, wird vor allem durch diese frühen Entscheidungen beeinflusst. Sie teilen zukünftige Änderungen an der Software in lokale, nicht lokale und die Architektur übergreifende Änderungen ein. Bestimmen also, wie aufwändig zukünftige Anpassungen und Erweiterungen sind. Die meisten dieser Punkte werden durch die Zerlegung des Systems in

seine Architekturbausteine sowie deren Verantwortlichkeiten und Beziehungen festgelegt. Die Organisationsstruktur des Entwicklungsprojektes wird sich z. B. an dieser Struktur der Software orientieren. Sie wird sehr früh im Projekt aufgesetzt, indem Teams gebildet und Mitarbeiter in Verantwortungspositionen gesetzt werden. Aufgrund einer falschen Softwarearchitektur eine solche Organisationsstruktur später wieder zu ändern ist sehr aufwändig und schafft viel Unruhe und Reibungsverluste im Projekt.

Zielgerichtet dokumentieren

Als letztes Mittel sei hier noch die zielgerichtete Dokumentation genannt. Diese wird zur Kommunikation eingesetzt. Mit zielgerichtet ist gemeint, dass sich die Dokumentation an den unterschiedlichen Aspekten von Softwarearchitektur und an den unterschiedlichen Lesern, den Stakeholdern, orientiert. Dies wird dadurch erreicht, indem die Architektur aus verschiedenen Sichten (engl. *views*) dokumentiert wird. Jede Sicht betrachtet dabei eine andere Menge an Informationen, einen anderen Ausschnitt der Gesamtarchitektur. Die statische Sicht zeigt z. B. Subsysteme und deren Beziehungen. In der dynamischen Sicht werden die wichtigsten Mechanismen dargestellt, wie z. B. die Verteilung von Nachrichten im System. Die Verteilungssicht beschreibt, wie die Software auf mehreren Prozessoren verteilt ist und wie diese miteinander vernetzt sind. Im Kapitel 6, »Dokumentation«, wird auf die Sichten der Architektur noch genauer eingegangen.

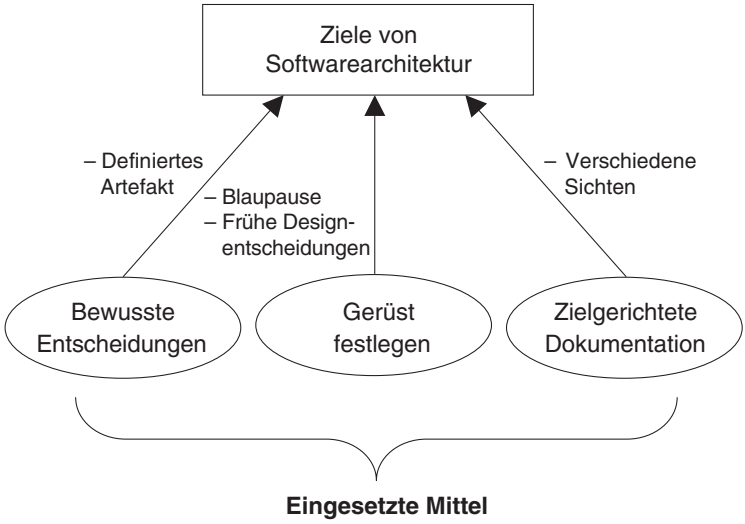

Abb. 1–5 Drei wesentliche Mittel setzt Softwarearchitektur ein, um ihre Ziele zu erreichen. Die wesentlichen Aspekte hinter den Mitteln sind an den Pfeilen notiert.

1.2.3 Wodurch wird Softwarearchitektur beeinflusst?

Funktionale Anforderungen sind nicht die wesentlichen Architekturtreiber!

Es scheint nahe liegend, dass die vom System geforderten funktionalen Anforderungen einen wesentlichen Einfluss darauf haben, wie die Softwarearchitektur zu entwerfen ist. Ein PDA benötigt nun einmal *Fachlichkeiten* wie ein Adressbuch, einen Terminkalender, ein Notizbuch oder einen E-Mail-Client. Demnach klingt es logisch, dass diese Funktionalitäten die Architektur der Software bestimmen werden. Tatsache ist jedoch, dass man für die Realisierung rein funktionaler Anforderungen diese auch in einen einzigen monolithischen Block implementieren könnte. Aus funktionaler Sicht würde dies keinen Unterschied machen. Dennoch beeinflusst die Struktur der Fachlichkeit die Architektur in dem Maße, dass sie dem Architekten als Ausgangsbasis für seinen Entwurf dient. Im Idealfall liegt ihm hierfür bereits aus der Anforderungsanalyse ein fachliches Modell vor. Im Kapitel 3.2.1, »Anforderungsanalyse«, werden wir darauf noch genauer eingehen. Wichtig ist aber, zu verstehen, dass die treibenden Kräfte für den Architekturentwurf andere sind.

Einfluss durch Qualitätsmerkmale

Einen ersten, wirklichen substanziellen Einfluss auf die Softwarearchitektur haben die geforderten *Qualitätsmerkmale* wie Leistung, Änderbarkeit, Sicherheit usw. Aber nicht nur die funktionalen Anforderungen und Qualitätsanforderungen beeinflussen Architektur. Wäre dies der Fall, würden zwei getrennt voneinander arbeitende Architekten auf Basis der Anforderungen die gleiche Softwarearchitektur entwerfen, da beide die gleichen Vorgaben besitzen.

Einfluss durch Managementanforderungen

Managementanforderungen wie Termin, Kostendruck und Ressourcenvorgaben beeinflussen die Architektur ebenso. Die Aufteilung funktionaler Anforderungen auf eine Struktur wird z. B. dadurch beeinflusst, dass die Software durch eine vorgegebene Anzahl von Entwicklerteams, unabhängig voneinander, effizient, in einem bestimmten Zeitrahmen implementiert werden muss. Auch die Ziele, die mit der Architektur verfolgt werden, sind durch das Management festgelegt. So ist eine grundlegende Entscheidung zum Beispiel, ob die Architektur nur für dieses Projekt entworfen oder ein Produktlinienansatz damit verfolgt wird.

Einfluss durch Stakeholder

Neben dem Management beeinflussen aber auch alle weiteren Stakeholder die Architektur. Kunden, Anwender, Marketing, Wartungsmitarbeiter oder die Qualitätsabteilung haben unterschiedliche Interessen, mit denen sie oftmals widersprüchlich auf den Architekten einwirken.

Einfluss durch Bestehendes

Weitere Einflussfaktoren sind der *Entwicklungsprozess*, das *technische Umfeld* sowie aktuelle *Trends und Industriestandards*. Werkzeuge und Technologien, die im Unternehmen etabliert sind, werden

eher zum Einsatz kommen als solche, zu denen keine Erfahrungen existieren bzw. neue, zusätzliche Investitionen getätigt werden müssen. Deshalb werden auch bereits bestehende Alt- oder Vorgängersysteme die Architektur des neuen Systems beeinflussen.

Einfluss durch Wissen

Am meisten beeinflusst wird die Architektur durch das *Wissen des Unternehmens* sowie *das Wissen, die Erfahrungen und die Motivation des Softwarearchitekten* selbst. Hat der Architekt bereits früher bestimmte Ansätze erfolgreich angewendet, wird er diese wieder heranziehen bzw. umgekehrt. Auch spielen die Aus- und Weiterbildung des Architekten sowie dessen Interesse, neu Gelerntes anzuwenden, eine wichtige Rolle.

Einfluss durch organisatorisches Umfeld

Letzten Endes hat auch das *organisatorische Umfeld*, die Natur und Struktur der Organisation eine entscheidende Bedeutung. Bestehen z. B. bestimmte Abteilungen in der Entwicklungsorganisation, die für spezielle Themen verantwortlich sind, werden diese Abteilungen dafür Sorge tragen, dass sie in der Architektur eines neuen Systems entsprechend berücksichtigt werden. Dies kann durchaus zu suboptimalen Architekturentscheidungen führen.

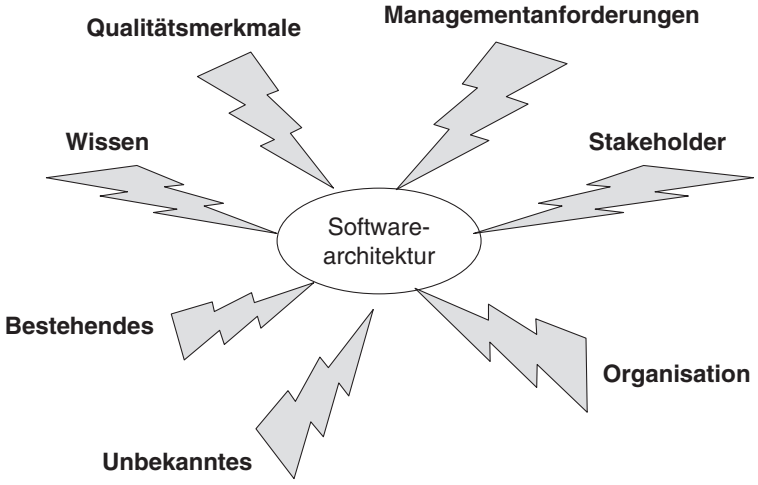

Abb. 1–6 Einflüsse auf Softwarearchitektur

Einfluss durch Unbekanntes

All die bisher aufgeführten Einflüsse lassen sich alleine dadurch in einer Architektur berücksichtigen, indem man sich diese bewusst macht. Bei nahezu jeder Softwareentwicklung wird aber auch *Neuland* betreten. So werden neue Technologien eingeführt, größere Systeme gebaut oder neue Mitarbeiter integriert. Dabei handelt es sich immer um unbekannte Größen, die Einfluss auf die Softwareentwicklung nehmen. Je nach Gewichtung können diese zu echten Risiken heran-

wachsen. Die Kunst liegt nun darin, auch diese unbekannten Einflüsse in der Architektur zu berücksichtigen. Die Kapitel 4, »Einflussfaktoren«, und 7, »Bewertung«, setzen sich mit dieser Problematik intensiver auseinander.

1.3 Bedeutung von Softwarearchitektur

Softwarearchitektur ist das kritische Artefakt.

Ist Softwarearchitektur wichtig? Warum noch einen Baustein in der Softwareentwicklung einfügen? Wieso Geld dafür ausgeben? Softwarearchitektur ist wichtig, da sie das zentrale, kritische Artefakt in der Softwareentwicklung ist. Weil sie abstrahiert und aus unterschiedlichen Perspektiven dokumentiert wird, ist sie der Schlüssel zum Systemverständnis für alle Projektbeteiligten. Sie legt die wichtigsten Designentscheidungen für das Softwaresystem fest und bestimmt somit wesentlich, ob das System in der Lage sein wird, die gestellten Anforderungen zu erfüllen. Sie erlaubt dadurch auch eine frühe Bewertung, ob die Software die geforderte Qualität erreichen kann. So wird eben nicht erst nach Beendigung der Implementierung festgestellt, ob das System die geforderten Laufzeiten einhält.

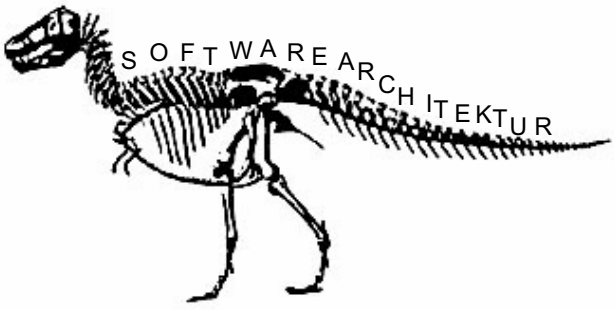

Abb. 1–7 Was die Natur schon seit langem weiß: Ein leistungsfähiges System benötigt ein stabiles Rückgrat. Architektur ist das Rückgrat der Softwareentwicklung.

Hoher Return on Investment

Zugleich wirft Softwarearchitektur einen sehr hohen Gewinn in Bezug auf Qualität, Zeitplan und Kosten für die getätigten *Investitionen* ab (engl. *return on investment (ROI)*). Wiederverwendung auf Basis einer Softwarearchitektur ist effektiv und vielversprechend, da es sich um bewährtes, erfolgreich angewandtes Know-how handelt. Gut dokumentiert konserviert sie vorhandenes, praktisches Wissen im Unternehmen. Können zudem Fehler bereits auf der Ebene der Architektur entdeckt und beseitigt werden, kostet ein solcher Fehler nur ein Hundertstel von dem, was er kosten würde, hätte man ihn in der fertigen Software entdeckt. Softwarearchitektur liefert deshalb sowohl einen

kurzfristigen wie auch langfristigen ROI. Auf kurze Sicht steigert sie die Effizienz des Projektes und reduziert Risiken. Langfristig erhöhen sich das Wissen des Unternehmens und das Wiederverwendungspotenzial.

Rückgrat des Entwicklungsprojekts

Eine gute Softwarearchitektur setzt den Grundstein für alles Weitere im Lebenszyklus eines Softwareprojektes: Entwicklung, Integration, Test, Änderungen, Wartung, Weiterentwicklung. Nur auf Basis einer gelungenen Softwarearchitektur ist es möglich, komplexe Softwareprojekte effektiv zu managen. Projektplanung und Aufwandsschätzungen ohne die Basis einer Softwarearchitektur haben nur minimalen Wert. Das folgende Kapitel wird diese Aspekte noch ausführlicher beleuchten. Die richtige Softwarearchitektur ist der erste Schritt zum Erfolg. Die falsche führt in die Katastrophe. Softwarearchitektur ist somit das *Rückgrat* einer jeden Softwareentwicklung.

Softwarearchitektur hat Einfluss über das Projekt hinaus.

Der Einfluss einer Softwarearchitektur hält länger an als die Lebensdauer eines Projektes. Sie ist ein wichtiges Gut des Unternehmens. Eine erfolgreiche Softwarearchitektur kann Einfluss darauf haben, wie zukünftige Systeme im Unternehmen entwickelt werden, und stellt somit ein hochwertiges geistiges Eigentum dar, das als solches dem Unternehmen einen entscheidenden *Wettbewerbsvorteil* verschaffen kann.

1.3.1 Symptome bei fehlender Softwarearchitektur

Qualitätsmerkmale werden erst nach Implementierung betrachtet.

Projekte ohne explizite Softwarearchitektur weisen oftmals typische Probleme auf, die direkt daraus resultieren, dass die von der Softwarearchitektur adressierten Ziele nicht unterstützt werden. Ein Beispiel hierfür ist, dass Qualitätsmerkmale erst adressiert werden, wenn sie benötigt werden. Das heißt, die Leistung des Systems wird erst ernsthaft betrachtet, wenn das gesamte System lauffähig ist und dann eventuell festgestellt wird, dass es zu langsam ist. Weitere Probleme kann die Forderung nach Änderbarkeit aufwerfen. Solche Probleme werden erst erkannt, wenn die ersten echten Änderungsanträge anfallen. Der Umbau der Software kann dann jedoch sehr teuer werden.

Machtlose Projektleiter, ineffiziente Entwickler

Projektleiter sind bei Problemen und Verzug der Projekte machtlos, da ihnen die Informationen und der nötige Einblick fehlen, um aktiv Maßnahmen zu ergreifen. Die Effektivität der Entwicklungsmannschaft bricht drastisch ein, da aufgrund des fehlenden Rahmens zu viel Kommunikation im Kleinen stattfinden muss. Neue Mitarbeiter können nur schwer eingearbeitet werden, da ihnen abstrahierte Information fehlt; aber nur dieses Wissen ermöglicht einen systematischen Einstieg in das laufende Projekt.

Unternehmen verliert technologischen Anschluss.

Im Unternehmen wird immer nur von Projekt zu Projekt geplant. Eine systematische Wiederverwendung von Architekturwissen findet nicht explizit statt. Dies führt dazu, dass sich das Unternehmen nur sehr langsam entwickelt. Somit besteht die Gefahr, dass der technologische Anschluss verloren geht. Unternehmen, die Softwarearchitektur frühzeitig einsetzen, lernen mehr über ihre Systeme und werden in die Lage versetzt, Softwareproduktlinien aufzubauen. Am Ende des Buches werden wir auf Produktlinien noch speziell eingehen.

Dies sind nur einige der Probleme, die durch eine ausdrückliche Softwarearchitektur adressiert werden.

1.4 Zusammenfassung

Unsere Gesellschaft wird in zunehmendem Maße abhängig von Software. Umso wichtiger ist es, dass wir Softwaresysteme mit hoher Qualität bei angemessenen Kosten- und Zeitvorgaben entwickeln können. Softwarearchitektur hilft uns, dies zu erreichen. Sie beschreibt die wesentlichen Strukturen und Mechanismen der Software auf den oberen Hierarchieebenen. Durch diese Abstraktion von Details dient sie als hervorragendes Kommunikationsmedium und schafft bei allen Projektbeteiligten ein Verständnis für das zu entwickelnde System. Darüber hinaus ist sie das Rückgrat für das Entwicklungsprojekt. Indem sie die wesentlichen Strukturen und deren Beziehungen beschreibt, legt sie zugleich in einem hohen Maße die Organisationsstruktur des Projektes fest und ist damit ein wichtiges Werkzeug des Projektleiters. Aber auch für die Entwickler liefert die Architektur wichtige Informationen. Die Entwickler wissen durch die Architektur, wie ihre Arbeitspakete im Gesamtsystem eingebunden sind. Die Architektur stellt somit eine Art Integrationsrahmen für die Entwicklung dar. Ein weiteres zentrales Problem, das Softwarearchitektur adressiert, ist, dass Qualitätsmerkmale, wie Laufzeit, Sicherheit oder Änderbarkeit, früh im Entwicklungsprozess berücksichtigt werden können. Gängige Praxis ist oftmals, dass diese erst sehr spät, am Ende der Implementierung, betrachtet werden und dann sehr hohe Kosten verursachen. Softwarearchitektur bringt diese Qualitätsanforderungen an den Anfang des Entwicklungsprozesses und trägt dadurch wesentlich zur Risikominimierung in Projekten bei. Besonders interessant macht Softwarearchitektur, dass sie all diese Ziele mit relativ einfachen und günstigen Mitteln erreicht. Dabei erhöhen sich langfristig zudem das Wissen des Unternehmens sowie dessen Potenzial für Wiederverwendung von Software. Wichtig ist, dass explizit Zeit für die Erstellung der Architektur im Projektplan vorgesehen ist.