→ 3., aktualisierte und erweiterte Auflage

Torsten Posch · Klaus Birken · Michael Gerdom

Basiswissen Softwarearchitektur

Verstehen, entwerfen, wiederverwenden

dpunkt.verlag

Basiswissen Softwarearchitektur

Torsten Posch studierte technische Informatik an der Berufsakademie Ravensburg als Student der DaimlerChrysler Aerospace AG. Bevor er im Management und der Geschäftsleitung mittelständischer Softwareunternehmen tätig war, arbeitete er als Entwickler, Berater und Projektleiter mit Fokus auf UML und Objektorientierung. Lange Zeit war er Vorsitzender des iSQI German Software Architecture Board. Seit 2004 ist Torsten Posch bei Continental Automotive als Programm-Manager schwerpunktmäßig für die Definition und Einführung neuer Elektronik- und Softwareplattformen tätig.
Seine E-Mail-Adresse lautet *swa@torsten-posch.de*.

Dr. Klaus Birken studierte Informatik in Erlangen und promovierte über die Themen High Performance Computing und Numerische Simulation an der Universität Stuttgart. In zahlreichen Embedded-Projekten brachte er sein Wissen über Architekturen und Entwicklungsmethodik ein. Dr. Birken ist Autor zahlreicher Veröffentlichungen und Vortragender auf internationalen Konferenzen, auch zum Thema Softwarearchitektur. Seit mehreren Jahren ist er Senior Software Architect bei Harman/Bekker Automotive Systems im Bereich Research & Development für Infotainmentsysteme.
Seine E-Mail-Adresse lautet *swa@birken.info*.

Michael Gerdom studierte Informatik an der Universität Erlangen-Nürnberg. Er besitzt langjährige, praktische Erfahrung im Einsatz von objektorientierten Technologien und UML in Engineering-Projekten sowie in der Schulung und Beratung zu den Themen Requirements Engineering, Softwarearchitektur und Prozessverbesserung. Bei Method Park leitete er zuletzt mehrere Jahre den Bereich Training & Consulting. Aktuell ist er als Managing Consultant bei Capgemini im Bereich Business Technology mit Schwerpunkt Projektmanagement und Prozessberatung tätig.
Seine E-Mail-Adresse lautet *swa@michael-gerdom.de*.

Torsten Posch · Klaus Birken · Michael Gerdom

Basiswissen Softwarearchitektur

Verstehen, entwerfen, wiederverwenden

3., aktualisierte und erweiterte Auflage

dpunkt.verlag

Torsten Posch
swa@torsten-posch.de

Klaus Birken
swa@birken.info

Michael Gerdom
swa@michael-gerdom.de

Lektorat: René Schönfeldt
Copy-Editing: Annette Schwarz, Ditzingen
Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: Media-Print Informationstechnologie, Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Buch 978-3-89864-736-6
PDF 978-3-86491-042-5
ePub 978-3-86491-043-2

3., aktualisierte und erweiterte Auflage 2011

Ringstraße 19 B
69115 Heidelberg

5 4 3 2 1 0

Geleitworte

Das Buch »Basiswissen Softwarearchitektur« vermittelt dem Leser einen sehr guten Überblick über das komplexe Thema. Es beschreibt umfassend und praxisnah und ist somit nützliche Literatur für alle, die in der Softwareentwicklung als Projektleiter, Architekt oder Entwickler tätig sind. Darüber hinaus dient es als sehr gute Begleitliteratur für Studierende der Informatik zum Thema Softwareentwurf und Software-Engineering.

Prof. Dr. Bernd Hindel,
CEO Method Park Software AG
(Method Park)

Preparing for a role as a software architect is difficult; this book overcomes that difficulty, teaching the required knowledge in a complete yet concise form. With a strong eye to practical use, the authors of this book show how to rapidly and successfully use UML 2.0 for describing application architectures.

Richard Mark Soley,
CEO Object Management Group
(OMG)

Vorwort

Klassische Softwareentwicklungsmethoden versagen mit zunehmend komplexer werdender Software, da sie zu einer Kostenexplosion am Ende des Projektes führen und aufgrund unflexibler, schwer zu ersetzender Altsysteme Innovation hemmen. Daher wird mittlerweile mehr und mehr Aufwand in die frühen Phasen der Softwareentwicklung investiert: in das Anforderungsmanagement und vor allem in die Softwarearchitektur, die Thema dieses Buches ist.

Im Rahmen der Ausarbeitung des Lehrplanes für den *iSQI Certified Professional for Software Architecture*, einem Weiterbildungsprogramm zum Softwarearchitekten, an dem sich Experten aus ganz Deutschland beteiligen, ist die Idee für dieses Buch entstanden. Als Autoren können wir unsere langjährige Erfahrung aus der Softwareentwicklung einbringen: Angefangen als Entwickler, später Designer oder Projektleiter, sind wir heute Architekten, Berater oder Manager für Softwarearchitekturen.

Das Buch ist technologieneutral geschrieben und richtet sich somit an alle Softwareentwickler, Designer, Architekten und in wichtigen Teilen auch an Projektleiter. Verschiedene, konkrete Technologien werden zur Veranschaulichung anhand von Beispielen dargestellt. Der Schwerpunkt liegt jedoch auf den Inhalten, die allen Softwarearchitekturen gemein sind: der Definition, dem Vorgehen, dem Entwurf, der Dokumentation, der Bewertung, bis hin zu den Werkzeugen des Architekten. Abschließend beleuchten wir die Technologien zur Industrialisierung der Softwareentwicklung wie Produktlinien, MDA und domänenspezifische Sprachen. Das Buch gibt somit einen umfassenden Überblick aller Themen, die für den Einsatz von Softwarearchitektur in der Praxis notwendig sind. Zur Vertiefung spezieller Themen dient die Angabe weiterführender Literatur.

Neben vielen kleinen Beispielen in den einzelnen Kapiteln enthält das Buch ein eigenes Kapitel mit einer kompletten Fallstudie »Fahrzeugnavigation«. Diese dient dazu, die wesentlichen Themen des

Buches nochmals anhand eines praktischen Beispiels zusammenzufassen. Wir haben uns bewusst dagegen entschieden, die Fallstudie auf die einzelnen Kapitel zu verteilen, um den Zusammenhang zu gewährleisten.

Von der Bedeutung und dem Nutzen von Softwarearchitekturen sind wir aus eigener, praktischer Erfahrung überzeugt. Mit dem Buch hoffen wir, auch Ihnen die vielseitigen Aspekte um Softwarearchitektur näherzubringen und Ihnen damit den Einsatz von Softwarearchitekturen in Ihren Projekten zu erleichtern. Über Rückmeldungen, Anregungen, Kritik oder Fragen freuen wir uns. Erreichen können Sie uns über die Webseite zum Buch unter **www.sw-architecture.com**.

Torsten Posch, Aschaffenburg
Klaus Birken, Stuttgart
Michael Gerdom, München
Dezember 2010

Danksagung

Der Ausgangspunkt für das vorliegende Buch war unsere Zusammenarbeit mit Frank Hoffmann im Rahmen der Erstellung des Kurses iSQI Certified Professional for Software Architecture. In vielen Arbeitssitzungen sind dort die Ideen entstanden, die später die Initialzündung für dieses Buch darstellten. Frank Hoffmann danken wir gemeinsam herzlich und freuen uns auf weitere Zusammenarbeit in spannenden Projekten.

René Schönfeldt, unser Lektor beim dpunkt.verlag, hat uns stets sehr unterstützt und uns trotz einiger Verwerfungen im Zeitablauf mutig begleitet. Ihm und der anonymen Gutachterriege des dpunkt.verlags danken wir sehr für wertvolle Kommentare und deren Interpretation.

Der solide wirtschaftliche Hintergrund ist nicht unwesentlich bei der Freiheit, sich Hunderte von Arbeitsstunden Zeit zum Schreiben eines Buches zu nehmen. Wir danken unseren Unternehmen, die uns diese Möglichkeiten und Freiheiten gegeben haben durch teilweise massiven Support und Bereitstellung von Infrastruktur.

Auch in das vorliegende Buch sind unzählige Gespräche, Diskussionen und Anregungen eingeflossen. Insbesondere danken wir in diesem Zusammenhang Paul-Roux Wentzel, Thomas Schütz und Torsten Pretzsch. Bei unseren bisherigen Lesern bedanken wir uns für das große Interesse an dieser Lektüre, das eine 2. und 3. Auflage ermöglicht hat. Danke auch an dieser Stelle für die vielen positiven Rückmeldungen.

Der größte Dank gilt unseren Lebenspartnerinnen und Familien, die besonders an vielen Abenden und Wochenenden mit großer Geduld unser Tun beobachteten und unterstützten: Charlotte, Federika, Julia, Mathis, Carlotta und Smilla.

Torsten Posch, Aschaffenburg
Klaus Birken, Stuttgart
Michael Gerdom, München
Dezember 2010

Hinweise für den Leser

Das Buch ist in drei Teile untergliedert. Teil I enthält ein Kapitel zu den Grundlagen der Softwarearchitektur. Das Kapitel vermittelt dem Leser ein Bild, was zu einer Architektur gehört und welche Bedeutung sie hat. Kapitel 2 beschreibt die Wechselwirkungen zwischen der Organisation des Unternehmens und der Architektur. Im Fokus stehen vor allem die Rolle des Architekten sowie die Interaktion zwischen Projektleiter und Architekt.

Teil II beschreibt in Kapitel 3 das Vorgehen bei der Architekturerstellung und in den weiteren Kapiteln einzelne vertiefende Themen. Kapitel 4 befasst sich mit Einflussfaktoren für die Architektur, den resultierenden Risiken und Lösungsstrategien. Kapitel 5 geht detailliert auf das zentrale Thema Entwurf ein und greift die Erkenntnisse aus dem vorherigen Kapitel auf. Kapitel 6 gibt einen Überblick über wichtige Prinzipien bei der Dokumentation und bezieht sich auf die UML 2 als konkrete Notationsform. In Kapitel 7 werden Strategien zur Bewertung einer Architektur vorgestellt. Dabei wird die Architecture Tradeoff Analysis Method (ATAM) als szenariobasierte Bewertungsmethode vorgestellt. Kapitel 8 gibt einen Überblick über das Handwerkszeug eines Architekten. Es werden Methoden und Werkzeuge für den Architekten besprochen. Den Abschluss von Teil II bildet eine Fallstudie, die anhand des Beispiels Navigationssoftware die vorher beschriebenen Themen nochmals aufgreift.

Im letzten Teil des Buches wird auf neue Technologien und Methoden der Softwarearchitektur hin zu einer industriellen Softwareentwicklung eingegangen. Kapitel 10 gibt hierzu eine Einführung und einen Überblick. In Kapitel 11 wird das Thema Produktlinien behandelt. Hier soll gezeigt werden, welche besondere Rolle die Architektur für eine erfolgreiche Umsetzung von Produktfamilien spielt. Kapitel 12 geht abschließend auf modellbasierte Entwicklung ein, mit dem Schwerpunkt auf dem MDA-Standard der OMG. Ergänzend werden domänenspezifische Sprachen behandelt. Beide Technologien werden

die Art und Weise, wie wir Software entwickeln, grundlegend verändern.

Im Weiteren wollen wir einige Empfehlungen zum Lesen der einzelnen Kapitel geben.

Die Kapitel 1–3 vermitteln einen Überblick zum Thema und sollten von jedem gelesen werden. Kapitel 4–8 stellen vertiefende Informationen zu einzelnen Themengebieten bereit und können unabhängig voneinander gelesen werden. Auch die Fallstudie in Kapitel 9 kann sowohl parallel zu den anderen Kapiteln als auch im Ganzen, als abschließendes Beispiel, gelesen werden. Die Fallstudie kann ebenso als Einstieg in das Buch dienen. Der Leser bekommt dann anhand eines praktischen Beispiels einen Überblick zu allen wichtigen Themen. Er kann eventuell nicht alle vollständig einordnen, bekommt aber über entsprechende Querverweise in die Kapitel Hilfestellungen. Der dritte Teil des Buches stellt ein eigenständiges Thema dar und kann auch als solches gelesen werden. Interessant ist er insbesondere für Architekten von Plattformsoftware und Produktfamilien.

Abschließend möchten wir noch direkt einige Zielgruppen ansprechen:

- Management, Projektleiter und Qualitätsmanager sollten die Kapitel 1–3 für einen Überblick über das Thema lesen. Die Fallstudie in Kapitel 9 kann dann das Verständnis weiter vertiefen. Außerdem beschreibt Kapitel 10 wichtige Aspekte für übergreifende Architekturen und zukünftige Entwicklungen.
- Architekten sollten das ganze Buch lesen. Auch Entwicklern empfehlen wir, das ganze Buch zu lesen, um ein umfassendes Verständnis zu bekommen.
- Leser, die leichter anhand konkreter, praktischer Beispiele lernen, sollten mit Kapitel 1 beginnen und sich dann der Fallstudie im Kapitel 9 zuwenden. Mit Hilfe der Querverweise können dann einzelne Kapitel vertieft werden.

Inhaltsverzeichnis

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 hat 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 beschreibt typische Symptome, die bei fehlender Softwarearchitektur auftreten, und unterstreicht damit die Bedeutung einer guten Softwarearchitektur.

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, die 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, in dem Software der Schlüssel für die Funktionsfähigkeit von Produkten wird, wird sie auch das erfolgsentscheidende Element. Die Wettbewerbsfähigkeit eines Unternehmens ergibt 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. der Navigation oder der 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 Abbildung 1–1 dargestellt. Neuere Methoden wie die Objektorientierung oder die agile Softwareentwicklung versuchen dies vor allem zu erreichen, indem sie den Grad der Wiederverwendung erhöhen und die Entwicklungsprozesse verbessern. Die Objektorientierung hat durch das Konzept der Klassen und Vererbung dazu beigetragen, den Grad der Wiederverwendung zu erhöhen, und damit das Spannungsdreieck entschärft.

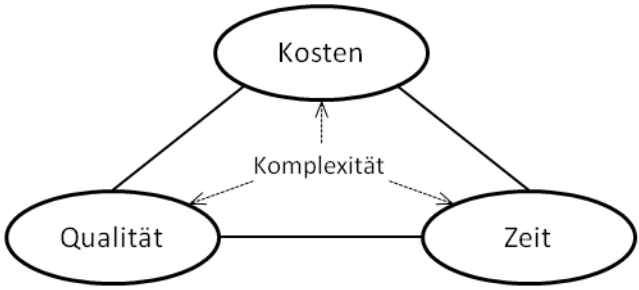

Abb. 1–1 Das Spannungsdreieck der Softwareentwicklung unter dem Druck der steigenden Komplexität ist die treibende Kraft der Methodenentwicklung.

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 ihren 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 Experten in einem Projekt, die die Softwarearchitektur erstellen. Sie sind keine Entwickler – sollten aber, bevor sie zum Architekten werden, Software entwickelt haben. Den Architekten obliegt die technische Verantwortung in der Softwareentwicklung. Dabei stehen sie vor der Herausforderung, zunehmend komplexere Software mit neuesten Technologien bauen zu müssen. 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 Komponententechnologien 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 einfa-

chen statischen Webseiten einzelner Einrichtungen oder Privatpersonen bis hin zu komplexen dynamischen Online-Portalen von Unternehmen. Neben der Beherrschung dieses Technologiewandels muss der Softwarearchitekt ein System so entwerfen, 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 ist jedoch erkennbar, dass sich zunehmend ein Konsens herausbildet. Wegweisend sind dabei die Arbeiten des Software Engineering Institute der Carnegie Mellon Universität (SEI), Siemens Corporate Research sowie einzelner Autoren aus dem universitären und Beratungsbereich. 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, indem sie Architekturbausteine und deren Schnittstellen in verschiedenen Sichten dokumentiert. Softwarearchitektur bildet damit 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. Diese Strukturen bestehen aus *Architekturbausteinen*. Hinzu kommen die von außen sichtbaren Eigenschaften und das von außen beobachtbare Verhalten der Architekturbausteine sowie die *Beziehungen* und *Interaktionen* zwischen diesen Bausteinen. Ein wesentlicher Bestandteil der Softwarearchitektur ist somit die Definition der *Schnittstellen* von Architekturbausteinen. 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 sollen 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 blendet Details aus. 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 Lauf-

zeiten 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.

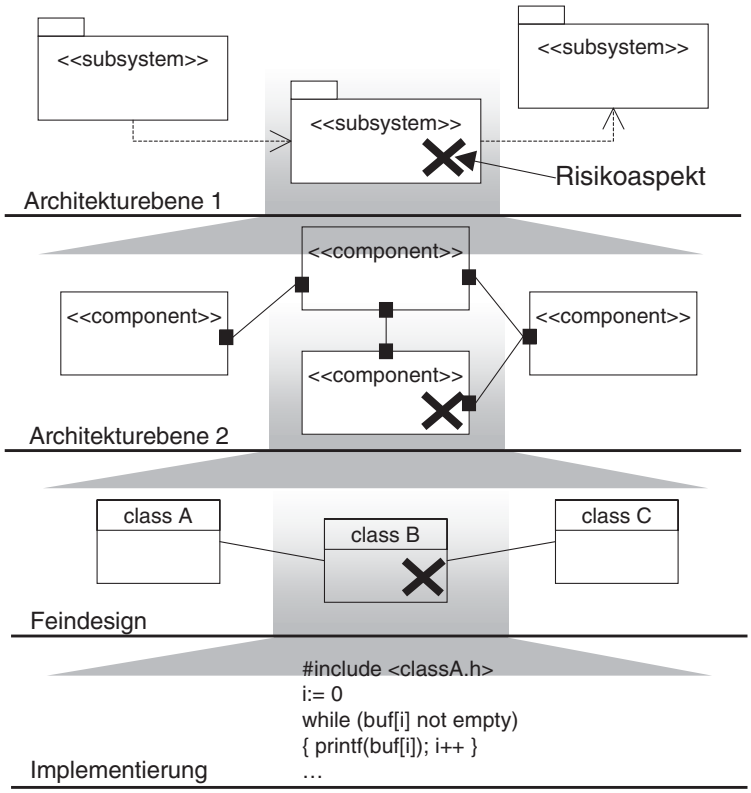

Abb. 1–3 Architektur bewegt sich auf den oberen Ebenen. Einzelne Risikoaspekte müssen in tiefere Ebenen hinein beleuchtet werden.

Kompakt: Was bestimmt den Detaillierungsgrad?

Softwarearchitektur bewegt sich letztlich 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 Fein-