1 Einleitung

Die Architektur eines Software-Systems beschreibt dieses zunächst als Komponenten zusammen mit den Verbindungen, die zwischen den Komponenten bestehen. Dazu gehören zwei weitere Sichten: eine Beschreibung der Dynamik, d. h. die Art, wie der Kontrollfluss durch diese Komponenten fließt, sowie die Abbildung der Komponenten und Verbindungen auf Ressourcen (Prozessoren, virtuelle Maschinen) bzw. auf logische oder physikalische Verbindungen. Eine Software-Architektur beschreibt demnach nicht den detaillierten Entwurf, vielmehr geht es darum, die Zusammenhänge zwischen den Anforderungen und dem konstruierten System zu beschreiben, möglichst mit einer Begründung für die Entwurfsentscheidungen. Die Wahl einer bestimmten Architektur hat dann einen starken Einfluss auf die nicht funktionalen (qualitativen), technischen Eigenschaften der resultierenden Systeme. Dieses Handbuch liefert einen fundierten Einstieg und umfassenden Überblick in dieses vielschichtige Thema.

Entsprechend dem IEEE-Standard 1471-2000 wird Software-Architektur als »The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution« definiert [ISO06]. Statt eine grundsätzlich neue Definition dieses für unser Handbuch zentralen Begriffs einzuführen, orientieren wir uns an diesem Standard:

Definition (Software-Architektur) *Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen.*

Eine Software-Architektur hat großen Einfluss auf die erfolgreiche Durchführung eines Software-Entwicklungsvorhabens. Der Beitrag einer Software-Architektur zum Erfolg ist umso größer, je umfangreicher, langwieriger oder komplexer ein Software-Entwicklungsprojekt ist. Diese Beiträge liegen vor allem im Bereich des Projektmanagements.

Eine gute Software-Architektur trägt zur *Effizienz des Entwicklungsprozesses* bei, indem sie Teilaufgaben voneinander entkoppelt und so Arbeitsteilung und eine flexible Projektorganisation ermöglicht. Sie gibt einen Rahmen vor, in dem iterativ-inkrementell entwickelt werden kann.

Die Software-Architektur hilft dabei, *Risiken zu minimieren*, indem diese frühzeitig berücksichtigt, beschrieben und gelöst werden. Darüber hinaus stellt die Software-Architektur ein wichtiges *Kommunikationsmedium* dar, das zwischen den verschiedenen Beteiligten – Kunden, Projektleiter, Entwickler, Lieferanten, Tester, Benutzer, Wartungsmitarbeiter usw. – Anforderungen und Erwartungen übermittelt und Diskussionen ermöglicht. Weiterhin sind frühzeitige Prüfungen möglich, sodass Missverständnisse verringert werden können. Doch eine Software-Architektur ist nicht nur in der Lage, Wissen zu vermitteln und zu transportieren, sondern durch die explizite Darstellung wird sie auch zur *Speicherung des Wissens* in Form einer Dokumentation verwendet. Damit dient sie der Weiterentwicklung und der Planung von Wiederverwendung genauso wie der späteren Überarbeitung und der Einarbeitung neuer Entwickler.

Die *Beschreibung* der Architektur dient dann verschiedenen Zielen, beispielsweise:

- zur Kommunikation mit allen »Stakeholdern« [McB07, Tab05],
- als Ausgangspunkt für Analyse und Simulation zur Bewertung und Vorhersage,
- als Zwischenprodukt im Entwicklungsprozess, das manuell zum Code, zum Deployment bzw. der Task-Aufteilung etc. verfeinert wird,
- als Eingabesprache für einen Generator in der modellgetriebenen Software-Entwicklung.

Je nach Einsatzzweck unterscheidet sich der benötigte Präzisionsgrad der Beschreibung.

Während der Entwicklung einer Architektur ist eine Vielzahl von Entscheidungen zu treffen. Ein methodisches Vorgehen der Entscheidungsfindung geht dabei von den Zielen aus, die sich aus der Aufgabenstellung und weiteren Bedingungen ergeben. In der Praxis treten zahlreiche Ziele und Einflussgrößen auf, die teilweise unscharf sind oder sich gar widersprechen. Außerdem sind die Konsequenzen von Entwurfsentscheidungen häufig nicht bekannt oder nur mit beträchtlichem Aufwand zu ermitteln, wie etwa anhand eines Prototyps. Für das Treffen von rationalen Entscheidungen unter solchen Bedingungen kann die Entscheidungstheorie gute Unterstützung leisten [WE02]. Sie stellt Mittel bereit, bei konkurrierenden Zielen Präferenzen festzulegen und Wahrscheinlichkeiten unsicherer Bedingungen auszuwerten.

Die Entwicklung einer Architektur orientiert sich häufig an Abstraktionen und an der Funktionalität, weil diese für das Verständnis und den Überblick wichtig sind. Für die erfolgreiche Umsetzung ist darüber hinaus eine weitere Aufteilung zwecks Arbeitsteilung notwendig, z. B. um Spezialwissen effektiv zu nutzen und um parallel zu bearbeitende Aufgaben verteilen zu können. Diese Aufteilung kann jedoch nicht an der Funktionalität ausgerichtet werden. Außerdem zeigt die

Erfahrung, dass häufig gerade die Qualitätsanforderungen zu Überarbeitungen eines Systems führen.

Es lassen sich drei Kategorien von Zielen unterscheiden:

- die vom Kunden gestellten funktionalen und qualitativen Ziele für das Produkt,
- die organisatorischen und technologischen Rahmenbedingungen der Entwicklung,
- die zu erwartende Variabilität der Ziele dieser beiden Kategorien.

Die Ziele sind entsprechend ihrem Einfluss auf die Architektur bzw. den durch sie bestimmten Risiken für den Erfolg der Entwicklung zu werten.

Damit ergeben sich verschiedene Faktoren, die eine Software-Architektur beeinflussen oder, andersherum gesehen, die beim Entwurf unbedingt bedacht werden sollten.

Die *funktionalen und qualitativen Ziele*, die üblicherweise zwischen Auftraggeber und Auftragnehmer vereinbart und in einem Pflichtenheft beschrieben werden, haben naturgemäß den größten Einfluss auf die Entwicklung einer Architektur. Funktionale Anforderungen stellen den Hauptteil der Anforderungen dar. Dennoch spielen gerade die Qualitätsanforderungen eine wichtige Rolle bei der Wahl der Architektur. Qualitative Anforderungen an Software-Systeme wurden standardisiert als Qualitätsmerkmale von Software-Systemen (ISO 9126 [ISO01]). Die systematische Behandlung von Qualitätsanforderungen beim Architekturentwurf ist noch Gegenstand der Forschung; ein erster Schritt in diese Richtung stellen Architekturbewertungsverfahren dar. Diese Verfahren schätzen basierend auf der Software-Architektur die nicht funktionalen Eigenschaften eines Systems ab. Auch wenn zu einem so frühen Zeitpunkt im Software-Entwicklungsprozess wegen fehlender Detailinformation, die erst in der Implementierungsphase zur Verfügung steht, noch keine absoluten Aussagen über die Systemeigenschaften (z. B. Echtzeitaussagen) gemacht werden können, so sind diese Verfahren doch hilfreich bei der Bewertung von Architekturalternativen.

Organisatorische Bedingungen schränken die Menge möglicher Alternativen bei Entscheidungen der Architekturentwicklung ein, sie beeinflussen somit die Architektur. Dieser Einfluss zeigt sich z. B. in der lang bekannten Korrespondenz zwischen der Anzahl von Modulen und der Anzahl von Entwicklerteams oder allgemeiner der Korrespondenz zwischen Software-Architektur und Organisationsstruktur (auch bekannt als »Conway's Law«, siehe z. B. [ER03]). Weiterhin sind Unternehmensziele zu beachten, insbesondere bezüglich der Langfristigkeit von Entscheidungen, der Dauer der Pflege eines Produkts, der Wiederverwendung oder Integration bestehender Komponenten, der Etablierung einer Produktlinie oder des Hinzukaufs externer Leistungen.

Technische Rahmenbedingungen stellen eine weitere Gruppe von Einflussfaktoren dar. Beispielsweise kann die Wahl einer 4GL-Sprache mit dazugehören-

den Werkzeugen eine klare Aufteilung in eine Drei-Schichten-Architektur nahezu unmöglich machen. Im Allgemeinen wird die Architektur beeinflusst von technischen Rahmenbedingungen wie Standards für Entwicklungsvorgehen (Prüfungen, Richtlinien, Art der Dokumentation), Vorgaben für die Entwicklungsumgebung (Programmiersprache, Werkzeuge etc.), Vorgaben für die Zielplattform (Prozessor, Speicher, Netzanbindung, Betriebssystem), Vorgaben für die zu verwendende Komponenten (Datenbank, Webserver, Klassenbibliothek, Benutzungsschnittstelle) oder technische Standards (wie Referenzarchitekturen, Datenformate, Kommunikationsprotokolle, Kompatibilität).

Man erkennt, dass der Software-Architekt[1] eine Reihe sehr verschiedener Einflussfaktoren auszubalancieren hat. Analog zu den Aufgaben des »klassischen« Architekten im Bauwesen würden die Aufgaben des Software-Architekten von der Anforderungserhebung, dem Entwurf bis zur Überwachung der Ausführung reichen. Zunächst ist festzustellen, dass heute die Rolle des Architekten als überwiegend technisch wahrgenommen und definiert wird. Die Internetseite »The Duties of an Architect« des Software Engineering Instituts (SEI) bietet die Möglichkeit, eigene Definitionen der Pflichten eines Software-Architekten dort eintragen zu lassen. Dort sind die meisten Aussagen überwiegend bis ausschließlich durch technische Aufgabendefinitionen geprägt und entsprechen damit eben gerade nicht der Analogie zum Gebäudearchitekten [SEI05]. Nun wäre es sicher zu einfach zu sagen, dass die häufigsten Probleme des Software Engineering – nämlich die fehlerhafte Erfassung von Anforderungen und die schlechte Zeitplanung – dadurch gelöst würden, wenn der Software-Architekt genau diese seit Langem als kritisch bekannte Überbrückung zwischen der Anwendungsdomäne und der technischen Domäne übernähme. Es ist vielmehr zu befürchten, dass bei der Komplexität beider Domänen kein einzelner Experte zu finden wäre, der beide Bereiche kompetent abdecken würde. Die Rolle des Software-Architekten wird erfahrungsreich von Wolfgang Weck im Kapitel 2 diskutiert. Dieses Kapitel stellt den Einstieg in das Handbuch dar, in dem das oben angerissene vielfältige Gebiet der Software-Architektur wie folgt gegliedert ist:

Konstruktion von Architekturen: In diesem Teil wird die Erstellung von Architekturen diskutiert. Dabei wird auf die Architekturmodellierung (Kapitel 3) ebenso eingegangen wie auf den Prozess der Architekturbeschreibung (Kapitel 4). Die modellgetriebene Entwicklung stellt eigentlich eine Nutzung von Architekturmodellen dar (der engl. Name »Model-Driven Architecture (MDA)« ist daher irreführend). Dennoch beeinflusst dieser Ansatz mit seiner Trennung der Domänen-, Anwendungs- und Plattformmodellierung

[1] Wir verwenden der besseren Lesbarkeit und der Einfachheit des Satzbaus wegen die gebräuchliche männliche Berufsbezeichnung. Dies soll allerdings nicht über den sehr bedauerlichen Mangel von Software-Architektinnen hinwegtäuschen, den abzustellen es wohl weiter gehender Maßnahmen bedarf als der weiblichen Berufsbezeichnung in einem Handbuch der Software-Architektur.

auch die Art der Architekturbeschreibung, weshalb die modellgetriebene Entwicklung bereits in diesem Teil behandelt wird (Kapitel 5). Der Entwurf serviceorientierter Architekturen ist geprägt von der Nutzung von Software-Services. Die Grundlagen des Entwurfs serviceorientierter Architekturen werden in Kapitel 6 vorgestellt. Zur Gestaltung einer Anwendungslandschaft greifen Software-Architekten auf eine Menge von Prinzipien und Methoden zurück, um die Architektur eines Systems zu erstellen. Die serviceorientierte Gestaltung von Anwendungslandschaften wird in Kapitel 7 erläutert.

Evolution von Architekturen: Da Architekturmodellierung gerade bei »langlebigen« Software-Produkten sinnvoll ist, spielt naturgemäß die Behandlung der verschiedenen Facetten der Evolution von Software-Architekturen eine wichtige Rolle. Zunächst werden die Grundlagen der Evolution von Software-Architekturen (Kapitel 8) gelegt. In dem darauffolgenden Kapitel wird auf das für die Praxis sehr wichtige Architektur-Reverse-Engineering eingegangen (Kapitel 9), bei dem es darum geht, so gut wie möglich Architektursichten aus bestehendem Code zu erhalten. Ebenso wichtig in der Praxis ist die Migration bestehender Software-Systeme zu neuen Architekturen; dazu wird in Kapitel 10 ein bewährtes Muster zur Migration einer bestehenden Architektur hin zu einer mehrschichtigen Architektur dargestellt.

Management von Architekturen: Dieser Teil beschäftigt sich mit dem Management von Architekturen im Gegensatz zu den eher technisch-orientierten ersten beiden Teilen. Zunächst wird allgemein auf Architekturmanagement eingegangen (Kapitel 11), dann etwas spezieller das Problem der Organisation des Architekturmanagements erörtert (Kapitel 12).

Bewertung von Architekturen: Wie oben erläutert, spielt die Bewertung von Software-Architekturen hinsichtlich ihrer Qualitätseigenschaften eine besondere Rolle für die ingenieurmäßige Software-Entwicklung. Zunächst werden allgemeine Grundlagen der Architekturbewertung sowie etablierte Vorgehensweisen dazu vorgestellt (Kapitel 13). Danach wird vertieft auf die Bewertung von Sicherheit (Kapitel 14) – im Sinne des englischen »safety« – und Performance (Kapitel 15) eingegangen. Beide Eigenschaften sind in vielen Anwendungsdomänen von hoher Bedeutung, und für diese Eigenschaften haben sich schon spezifische Verfahren herausgebildet.

Wiederverwendung von Architekturen: Dieser Teil beschäftigt sich mit der Wiederverwendung von Architekturwissen und mit dem Einsatz von Architekturen, um wiederverwendbare Software zu schaffen. Ersteres wird in dem Kapitel über Software-Muster (Kapitel 16), also wiederverwendbare Lösungen für wiederkehrende Probleme, und in dem Kapitel über Referenzarchitekturen (Kapitel 17), also »Standardarchitekturen« für bestimmte Anwendungsdomänen, in die Architekturen dieser Domäne hin abbildbar sind, behandelt. Die zweite Blickrichtung – also Wiederverwendung durch Archi-

tekturen – wird in dem Kapitel über Produktlinienentwicklung (Kapitel 18) und Framework-Entwurf (Kapitel 19) vertieft.

Beispiele von Architekturen: Eine der vielleicht wichtigsten Arten, Wissen über Architekturen zu vermitteln, ist die Beschreibung von Architekturen selbst. In Kapitel 20 wird die Multimedia-Framework-Architektur MM4U vorgestellt. Das Kapitel 21 beschreibt allgemein die Familie der Peer-to-Peer-Architekturen und Kapitel 22 die Familie der Grid-Architekturen. In Kapitel 23 wird anhand zweier Anwendungsszenarien der konkrete Einsatz serviceorientierter Architekturen und die Umsetzung in der Praxis vorgestellt. Die Migration eines Altsystems zu einer Java-Enterprise-Architektur wird dann abschließend in Kapitel 24 präsentiert.

Ein Glossar auf der Webseite des Handbuches fasst alle expliziten Begriffsdefinitionen zusammen. Ein umfangreiches Literaturverzeichnis und ein Index vervollständigen das Handbuch.

Die Fülle des im Handbuch enthaltenen Materials könnte darüber hinwegtäuschen, dass wir schon alleine aus Gründen des beschränkten Umfangs des Handbuches bei jedem Thema entscheiden mussten, mit welcher Ausführlichkeit es behandelt werden soll. So ist beispielsweise entschieden worden, nicht einen weiteren Architekturmusterkatalog ins Handbuch aufzunehmen, auch wenn man dies möglicherweise von einem Handbuch erwarten könnte. Zum einen hätte die breitere Behandlung von Architekturmustern in diesem Buch viele andere Themen verdrängt, zum anderen gibt es bereits die wohletablierte Serie der »POSA-Bücher« (»Pattern-oriented Software Architecture«) [BMR+96, SSRB00, KJ04, BHS07a, BHS07b], und es erschien uns wenig sinnvoll, diese Muster oder die anderer Kataloge hier zu reproduzieren, zumal wir schon alleine wegen der besseren Aktualisierbarkeit und der Nutzung von Querverweisen webbasierte Musterkataloge als geeigneter ansehen. Insbesondere möchten wir hier auf die Webseiten unseres Arbeitskreises Architektur- und Entwurfs-Muster (AK AEM) verweisen, auf denen eine umfangreiche Sammlung von Architekturmustern zu finden ist.[2]

Das Thema der serviceorientierten Architekturen (SOA) [RHS05b] wird in Kapitel 6 eingeführt und in Kapitel 7 vertieft. Eine der grundlegenden Innovationen des SOA-Konzeptes ist die explizite Verknüpfung von fachlichen und technischen Konzepten. Dies geschieht zum einen durch Verwendung des Service-Begriffes auf der Geschäfts- und Anwendungsebene, aber auch insbesondere durch das Konzept der Orchestrierung, welches fachliche Geschäftsprozesse und technische Workflows zusammenbringt. Im Beispielkapitel 23 werden SOA in unterschiedlichen Anwendungskontexten behandelt. Im Kapitel 12 zum Architekturmanagement und in den Kapiteln 10 und 24 zur Migration spielen SOA ebenfalls eine wichtige Rolle.

Unabhängig vom Einstieg in das Handbuch durch Kapitel 2 gibt es keine vorgesehene Kapitelreihenfolge. Naturgemäß werden im Kapitel über die Archi-

[2] http://www.architekturmuster.de/

tekturmodellierung (Abschnitt 3.2) einige Grundlagen behandelt, die zum Verständnis der folgenden Kapitel hilfreich sind. Dennoch kann ein Experte gleich mit dem für ihn wichtigsten Kapitel einsteigen. Eine grafische Übersicht über die Teile des Handbuches ist in Abbildung 1.1 gegeben.

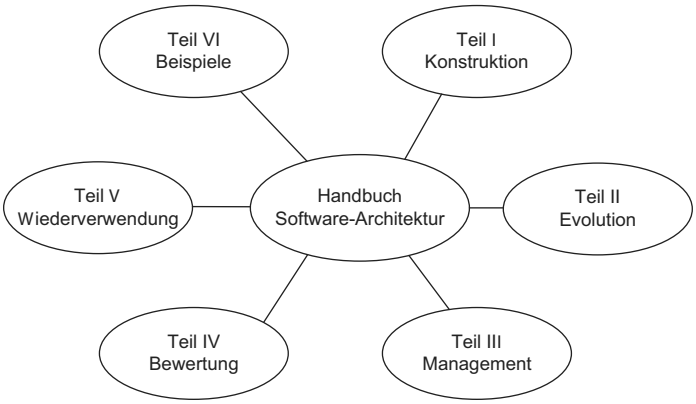

Abbildung 1.1: Übersicht über die Gliederung des Handbuches

Die Internetseite zum Handbuch enthält zusätzliche Materialien, die einer höheren Änderungsrate unterliegen.[3] Dies ist zum einen eine Übersicht bestehender Architekturmodellierungs- und Managementwerkzeuge, zum anderen ein Wiki, das das Glossar enthält. In einer so jungen Disziplin wie der Software-Architektur, die zugleich bereits heute schon sehr verzweigt ist und Einflüsse verschiedener Gebiete der Software-Technik aufnimmt, wird ein solches Glossar der Evolution unterliegen. Daher soll das Wiki auf der Handbuch-Internetseite die Basis bilden für eine breite Diskussion bestehender, aber auch neuer Begriffe eines Glossars zur Software-Architektur, das über dieses Handbuch hinausreicht.

[3] http://www.handbuch-softwarearchitektur.de