Xpert.press

Software-Architekturen für Verteilte Systeme

Prinzipien, Bausteine und Standardarchitekturen für moderne Software

Bearbeitet von
Schahram Dustdar, Harald Gall, Manfred Hauswirth

1. Auflage 2003. Buch. x, 264 S. Hardcover
ISBN 978 3 540 43088 9
Format (B x L): 15,5 x 23,5 cm
Gewicht: 578 g

1 Was ist Software-Architektur?

Software-Architektur hat sich in den letzten Jahren als eine wesentliche Teildisziplin in der Software-Entwicklung herausgebildet. Es hat zwar seit jeher gute Designer und Analytiker von Software-Systemen gegeben, die hervorragende Software-Architekturen entworfen haben, die Richtlinien und Werkzeuge für gutes architekturelles Design wurden aber erst in jüngster Zeit konkretisiert. So hat sich – beeinflusst von den zahlreichen Methoden, Techniken und Werkzeugen für den architekturellen Entwurf – im Abgleich mit den Vorgehensweisen zur Erstellung von Software eine eigene Teildisziplin entwickelt, die sich explizit mit der Architektur von Software-Systemen befasst.

Die zunehmende Komplexität in der Erstellung von Software-Systemen erfordert heute einen geordneten und expliziten Zugang zur Modellierung von Software, um die neuen technologischen Möglichkeiten und die damit einhergehenden wachsenden Anforderungen an ein Software-System zu bewältigen und Lösungen zu schaffen, die flexibel und erweiterbar für künftige Anforderungen sind.

Es genügt nicht mehr, eine Lösung für bestimmte Anforderungen zu schaffen, sondern die Bausteine einer guten Lösung müssen erweiterbar, wartbar und auch im Kontext von System-Familien einsetzbar sein. Um solche Anforderungen umzusetzen, müssen Technologien für den architekturellen Entwurf von Software-Systemen entwickelt, in den Entwicklungsprozess integriert und stets aktualisiert werden.

Faktoren wie Kosten und Zeit sind für Software-Entwicklungen kritischer und enger denn je, und dies ist darüber hinaus in Verbindung mit einer neuen Dimension von Komplexität im Zusammenwirken von vielen Software-Teilen (z.B. Subsystemen) umzusetzen. Es ist daher erforderlich, sich mehr denn je über den Entwurf und die Weiterentwicklung (d.h. Evolution) eines Software-Systems von Anbeginn Gedanken zu machen.

Die Architektur eines Software-Systems dient als wesentliche Beschreibung, um ein System zu verstehen. Dabei geht es nicht nur um das Verständnis der Einzelteile, sondern um deren Zusammenwirken hinsichtlich Kontroll- und Datenfluss sowie die damit verbundenen Einschränkungen oder geltenden Bedingungen (d.h. *constraints*). Das Verständnis vom System erstreckt sich von den beteiligten Personen (vom Kunden bis zum Entwickler) hin zum Entwicklungsteam selbst sowie den Personen, die das System nach Inbetriebnahme warten, pflegen und weiterentwickeln sollen.

Ähnlich zu anderen Ingenieurdisziplinen wie Bauingenieurwesen oder Architektur sollte es auch für Software eigene „Baupläne“ geben, welche die wesentlichen statischen und dynamischen Eigenschaften eines Systems wohldefiniert darlegen. Werden Komponenten geändert oder gar ersetzt, soll dieser Bauplan für die

Abschätzung der erforderlichen Änderungen und der einzusetzenden Ressourcen dienen.

Trotzdem bedingt eine gute Architektur noch lange nicht automatisch ein gutes Software-System. Ein Abgleich der technologischen Möglichkeiten mit den Benutzeranforderungen stellt nach wie vor einen kritischen Faktor in der Software-Entwicklung dar und wird durch das Erstellen einer Software-Architektur nicht ersetzt. Unterschiedliche Arten von Lösungen und Technologien bieten eine Vielfalt an Auswahlmöglichkeiten für den architekturellen Entwurf und in zahlreichen Fällen werden dabei die inhärenten Rahmenbedingungen, die mit der Entscheidung für eine bestimmte Ausprägung einer Architektur einhergehen, nicht oder nicht ausreichend berücksichtigt. Dies bedeutet, dass nur dann eine gute Architektur erreicht werden kann, wenn die Lösung und deren Charakteristik hinsichtlich Qualitätsattribute des Systems ausreichend analysiert wird.

Zu den architekturellen Überlegungen sind noch weitere Aspekte hinzuzunehmen: Ist das Ergebnis der Entwicklung ein einzelnes Software-System oder ist darüber hinaus geplant (oder absehbar), dass eine Menge von ähnlichen Systemen – eine so genannte System-Familie – entstehen soll? Dies bringt eine zusätzliche Dimension an Komplexität für die Modellierungsphase, da plötzlich für eine Klasse von Systemen eine Architektur entworfen werden soll.

Die Teildisziplin der Software-Architektur als solche ist noch eine sehr junge Disziplin, die sich erst Mitte der 90er-Jahre herausgebildet hat und die dann erst Ende der 90er-Jahre zu anerkannten Techniken, Methoden und Werkzeugen geführt hat. Das Gebiet ist weiterhin in steter Fortentwicklung, sodass Software-Architekten und -Designer stets neue Erkenntnisse und Methoden in ihre Vorgangsweise zu integrieren haben. Fragen der Repräsentation einer Software-Architektur, Bewertungs- und Validierungsmethoden für einen möglichen Entwurf sowie Fragen der software-technischen Umsetzung unterliegen noch immer großen Schwankungen und kurzen Erneuerungszyklen.

Ziel dieses Buches ist es, dem Leser die aktuellen Methoden, Techniken und Vorgangsweisen für den architekturellen Entwurf von verteilten Software-Systemen zu vermitteln und konkrete Architekturen und Komponenten samt deren Charakteristiken und inhärenten Annahmen anschaulich darzustellen und zu diskutieren. Dazu werden die aktuellen Software-Technologien wie z.B. architekturelle Stile und Muster herangezogen und im Kontext konkreter Software-Architekturen wie z.B. Web-Services oder Peer-to-Peer-Systeme erläutert.

1.1 Software-Architektur als Abstraktion

Ein wichtiger Aspekt von Software-Architektur ist, dass sie eine Abstraktion darstellt, die zur Reduktion der Komplexität in der Veranschaulichung eines Software-Systems wesentlich beiträgt. Dabei werden Mechanismen der Dekomposition und Transparenz verwendet, wie sie beispielsweise aus dem OSI-Modell der Rechnernetze oder dem ODP-Referenzmodell (Bair u. Stefani 1998; Putmann 2001) bekannt sind.

Details werden gruppiert, abstrahiert und in abstrakten Teilen gekapselt. Eine Software-Architektur soll die Teile eines Systems möglichst abstrakt festlegen und beschreiben und dabei die Details verbergen. Die Architektur soll beschreiben, wie diese Software-Teile die Anforderungen des Kunden erfüllen, welche Verantwortlichkeiten die einzelnen Teile haben und wie diese Teile für die Umsetzung der Anforderungen zusammenwirken.

Dabei gilt es neben den funktionalen und nichtfunktionalen Anforderungen auch weitere Einflussgrößen, die nicht notwendigerweise von Kundenseite eingebracht oder festgelegt werden – wie z.B. Portabilität, Testbarkeit oder Änderbarkeit – zu berücksichtigen. Die Software-Architektur ist letztendlich Ausdruck all jener Anforderungen und Einflussgrößen, die im Zuge des architekturellen Entwurfs beachtet und umgesetzt worden sind. Absehbare technologische Entwicklungen und Innovationen sind wesentliche Aspekte im Entwurf und der Beschreibung einer Software-Architektur und lassen somit auch eine Unterscheidung in bessere oder schlechtere architekturelle Entwürfe zu.

Gibt es wesentliche Algorithmen oder Klassenstrukturen zu beachten (z.B. an den Schnittstellen zu externen Systemen oder anderen Subsystemen), so sind diese zwar in die entsprechenden Teile der Architektur aufzunehmen, aber von der detaillierten Betrachtung vorerst auszublenden. Ihnen wird man sich erst im Detailentwurf zuwenden und sie realisierungstechnisch konkretisieren. Somit kann man sich im architekturellen Entwurf auf die Charakteristiken des Systems und seine Ausprägungen konzentrieren, ohne bereits von den programmtechnischen Details und deren Komplexität vereinnahmt zu werden.

Klassische Prinzipien der Abstraktion, Kapselung sowie *information hiding* (Parnas 1972) finden sich somit auch auf der Ebene der Software-Architektur wieder und tragen auch hier stark zu einem guten architekturellen Entwurf bei.

Die Herausforderung im architekturellen Entwurf ist es, sowohl für andere Beteiligte (vom Kunden bis zum Entwickler) eine geeignete und verständliche Definition und Beschreibung der Software-Architektur zu finden als auch für den Architekten und dessen Team selbst eine derartige Form der Architektur zur Verfügung zu haben, die es erlaubt, den Entwurf zu evaluieren und zu bewerten, sodass eine möglichst optimale Lösung für die Anforderungen erarbeitet werden kann.

1.2 Software-Architektur als Bauplan

Software-Architektur wird oft als Bauplan für ein Software-System gesehen. Diese Betrachtung wurde aus anderen Ingenieurdisziplinen herangezogen und bietet einige interessante Aspekte für die Software-Entwicklung.

Die Software-Architektur als Bauplan beschreibt nicht die Aktivitäten und Ressourcen für die System-Entwicklung, sondern die Elemente und Teile der Software sowie deren wesentliche Interaktionen. In diesem Sinn ist ein solcher Bauplan wohl mehr im bautechnischen oder architekturellen Blickwinkel zu sehen, der vorgibt, welche (Bau-)Elemente mit welchen anderen (Bau-)Elementen direkt verbunden werden, welche Verbindungselemente zu verwenden sind, welche

(bau)technischen Charakteristiken die einzelnen Elemente haben, sodass letztendlich durch das systematische Zusammenfügen der Einzelteile eine (statisch) tragfähige Lösung entsteht. Ähnlich einem Gebäude oder einem anderen Bauwerk werden durch den Bauplan die technischen Vorgaben und damit die entstehende Charakteristik des Bauwerks definiert.

Ferner gibt es auch bei Bauwerken nicht genau einen Bauplan, sondern unterschiedliche (An-)Sichten oder perspektivische Simulationen für die Einbettung in die Landschaft bis hin zu detaillierten technischen Plänen zur Ausführung konkreter Aspekte wie z.B. Verbindungen, Verkabelungen, Raumgestaltung etc.

Ähnlich den Geschossplänen können auch im software-technischen Sinn Pläne für Schichten oder Komponenten erstellt werden, die unterschiedliche Detailgrade aufweisen können. D.h., um ein Software-System erfolgreich zu erstellen, ist es wichtig, stets das gesamte System im Auge zu behalten, um nicht durch die Menge und Komplexität der Details verwirrt oder gar negativ beeinflusst zu werden.

Aus dem Gesagten folgt, dass es nicht eine einzige Darstellung oder Dokumentation einer Software-Architektur geben kann, sondern dass bestimmte Aspekte in eigenen Plänen beschrieben werden, um nicht das Wesen des zu erstellenden Software-Systems aus den Augen zu verlieren. Solche Aspekte können beispielsweise die Art der Kommunikation, Synchronisation, Kontrolle oder des Datenaustauschs zwischen Komponenten, Schichten, Subsystemen oder Modulen sein. Je nach Aspekt wird man also einen eigenen Bauplan entwerfen, der näher auf einen software-technischen Aspekt eingeht.

Weiters ist es essenziell, ob ein Bauplan für ein einzelnes Software-System erstellt wird, oder ob es sich um mehrere ähnliche Systeme im Rahmen einer so genannten System-Familie handelt. Baupläne für System-Familien weisen andere Charakteristiken als solche für Einzelsysteme auf, da sie sowohl die Gemeinsamkeiten der System-Familie als auch die Variabilitäten der einzelnen Familien-Mitglieder beschreiben müssen. Solche Baupläne stellen hohe Herausforderungen an die Software-Architekten dar. Ein Beispiel für eine solche System-Familie ist die Software für Mobiltelefone, die für eine Reihe von Produkten sehr ähnlich ist (z.B. Anruferlisten, Kurzwahl, *short message service*), sich aber in der Integration oder Aktivierung von bestimmten Features unterscheidet. Wesentliche Teile dieser Software wären allerdings gleich und daher in einem Bauplan als solche Gemeinsamkeiten explizit darzustellen und zu beschreiben.

Noch einen Schritt weiter geht der Ansatz, eine so genannte Referenzarchitektur für eine Generation von ähnlichen Software-Produkten innerhalb einer System-Familie zu definieren. Eine solche Referenzarchitektur fasst die Gemeinsamkeiten mehrerer ähnlicher System-Familien zusammen und beschreibt diese in einer allgemeinen Software-Architektur. Diese Referenzarchitekturen bilden beispielsweise bei Software in der Mobiltelefonie Baupläne für eine Generation von Software-Produkten; diese Software-Produkte leiten sich von der Referenzarchitektur ab, bilden selbst aber noch keine produktspezifische Einzelarchitektur, sondern beschreiben die Architektur für eine Menge sehr ähnlicher Produkte. Im Beispiel der Mobiltelefonie gibt es somit eine Referenzarchitektur für eine Generation von Mobiltelefonie-Software mit einem Zyklus von ca. 6 Monaten und eine System-Familien-Architektur für eine Produktreihe (wie z.B. Mobiltelefon X, Produktreihe 300, mit einzelnen Modellen 310, 320, 330 etc.). D.h., alle 300er Modelle bil-

den eine System-Familie und werden nach deren System-Familien-Architektur gebaut; die Architektur der 300er sowie weiterer anderer System-Familien folgen der Referenzarchitektur einer konkreten Generation (und deren Features).

Geht man nun noch einen Abstraktionsschritt weiter, so könnte man noch beschreiben, wie Software für Mobiltelefone herstellerunabhängig funktioniert. In diesem Fall beschreibt man den Anwendungsbereich, die so genannte Domäne. Jede spezifische Mobiltelefonie-Software hätte dieselben Elemente und Verbindungen zwischen den Elementen. Ein solches Domänenmodell beschreibt somit sehr allgemein eine Klasse von Software-Systemen.

Software-Architektur als Bauplan ist daher stets mit dem Ziel, für das eine solche erstellt wird (vom Einzelsystem über eine Referenzarchitektur bis hin zu einem Domänenmodell), eng gekoppelt und hat unterschiedliche Ausprägungen. Die Prinzipien der Definition und Beschreibung einer Software-Architektur bleiben aber dieselben.

In den nachfolgenden Kapiteln werden die hier nur kurz eingeführten Begriffe genauer definiert.

1.3 Software-Architektur-Terminologie

Wann immer wir über Software-Architektur sprechen, ist damit das zuvor diskutierte Bauplan-Konzept für ein Software-System gemeint. Eine wesentliche Unterscheidung ist darüber hinaus vorab zu treffen: *System-Architektur* beschreibt einen komplett anderen Bereich von Architekturen, der sich mit dem Arrangement und der Integration von diversen Systemen beschäftigt. Dabei ist die Betrachtungsweise auf Systeme als Bauteile gerichtet und nicht auf die Software-Bauteile für ein oder mehrere Systeme. Komponenten einer System-Architektur sind daher anders zu diskutieren als Komponenten und Beziehungen einer Software-Architektur.

Architekturelle Entwürfe lassen sich aufgrund ihrer Charakteristik in der Anordnung von Software-Komponenten und deren Kommunikationsverbindungen gruppieren. Daraus resultieren bestimmte *architekturelle Stile*, die konkrete Eigenschaften hinsichtlich Kommunikation, Verhalten, Erweiterbarkeit etc. haben. In späteren Kapiteln werden einige dieser Stile näher vorgestellt.

Arrangements von Komponenten und deren Interaktionen, die häufig für dieselben Anforderungen verwendet werden, kann man – ähnlich wie im Design oder der Codierung – in so genannten *architekturellen Mustern* zusammenfassen. Ähnlich wie architekturelle Stile definieren diese architekturellen Muster konkrete Teillösungen samt vorgeschlagenen Komponenten und Interaktionsmöglichkeiten. Während architekturelle Stile wesentlich allgemeiner die Charakteristik einer Software-Architektur prägen, sind architekturelle Muster mehr auf der Teile-Integrations- und Teile-Interaktionsebene angesiedelt.

Die Vorgehensweise, um eine Software-Architektur zu erstellen, auf die Anforderungen abzustimmen und zu dokumentieren, wird als *architektureller Entwurf* angesehen. Diese Aktivität ist nunmehr ein eigenständiger Teilprozess der Software-Entwicklung geworden (ähnlich wie Anforderungsanalyse oder Design).

Die Modellierung und Beschreibung einer Architektur wird mit so genannten architekturellen Beschreibungssprachen (ADLs, *architecture description languages*) durchgeführt. Diese ADLs wurden teilweise speziell für den architekturellen Entwurf designed bzw. als bereits existierende Beschreibungssprachen (z.B. Petri-Netze, Blockdiagramme etc.) für den architekturellen Entwurf herangezogen. Den ADLs werden wir uns in einem nachfolgenden Kapitel näher widmen.

Versucht man nicht bloß für ein einzelnes Software-System eine Architektur zu erstellen, sondern für eine Klasse von ähnlichen Systemen, so spricht man von einer Software-Architektur für eine *System-Familie*. Es entsteht in diesem Prozess eine *Referenzarchitektur*, der die einzelnen Mitglieder der System-Familie folgen. Naturgemäß ist eine Referenzarchitektur allgemeiner und abstrakter als eine Software-Architektur für ein einzelnes Software-System. Oft sind die Mitglieder einer System-Familie auf eine Produktlinie eines Unternehmens fokussiert; in diesem Fall spricht man von einer Architektur für eine Produktfamilie (*product line*). Verwenden beispielsweise alle Produkte einer Produktfamilie dieselbe Fehlerroutine, so wird diese Komponente einmal entworfen und entwickelt und dann in allen Produkten verwendet.

Erstellt man über eine System- oder Produktfamilie hinaus eine Architektur, die für einen ganzen Anwendungsbereich (i.e. eine so genannte Domäne) gelten soll, so ist die Architektur für diese Kategorie von Systemen noch wesentlich allgemeiner festgelegt. Software-Komponenten und Interaktionen werden für eine Domäne definiert und beschrieben; diese können dann auf konkrete Software-Systeme in dieser Domäne umgelegt werden, sodass für jedes dieser Software-Systeme die wesentlichen Elemente und Interaktionen festgelegt sind. Man spricht in diesem Zusammenhang von einer domänenspezifischen Software-Architektur (*domain-specific software architecture*).

Nach dieser kompakten Kurzvorstellung der Begriffe im Bereich Software-Architektur ist es wichtig, diese auch im Vergleich zueinander zu sehen. Dazu haben Hofmeister et al. eine Übersicht erstellt, die nachfolgend in Tabelle 1.1 dargestellt ist. Einer der wesentlichen Unterschiede besteht demnach darin, ob jeweils Instanzen von architekturellen Elementen definiert werden: Stile und Muster sind genauso wie Referenzarchitekturen und domänenspezifische Architekturen derart allgemein und definieren keinerlei Instanzen von Komponenten. Produktlinien sowie Software-Architekturen für Systeme oder Produkte sind da schon wesentlich spezieller und definieren auch Instanzen von den architekturellen Bauplänen. Die restlichen Einträge in der Tabelle folgen direkt den obigen Ausführungen zur inhaltlichen Abgrenzung der einzelnen Begriffe.

Eine Software-Architektur fokussiert demzufolge auf ein einzelnes System und beschreibt dessen Komponenten (Elemente bzw. Elementtypen), die Interaktionen zwischen den Komponenten, die Abbildung von Funktionalität auf Komponenten und Interaktionen sowie die Instanzen des architekturellen Bauplans, die im Software-System dann existieren.

Tabelle 1.1. Ein Vergleich von Software-Architektur-Begriffen (Hofmeister et al. 2000)

Begriff	Bereich	definiert Element-typen und deren Inter-aktion	definiert Abbildung von Funktionalität auf Architek-tur-Elemente	definiert Instanzen von Archi-tektur-Elementen
Architektureller Stil oder architek-turelles Muster	domänen-unabhängig	Ja	manchmal	Nein
Referenzarchitek-tur oder domänen-spezifische Soft-ware-Architektur	domänenspezifisch	Ja	ja	Nein
Produkt-Linien-Architektur	Produktmenge einer Organisation	Ja	ja	Manchmal
Software-Architektur	System oder Pro-dukt	Ja	ja	Ja

1.4 Was ist Software-Architektur?

Die softwaretechnischen Grundlagen von Software-Architekturen wurden schon früh im Software-Engineering selbst gelegt: David Parnas, Frederick Brooks, Edsger Dijkstra und andere haben beginnend in den 60er-Jahren über Prinzipien der Entwicklung von sehr großen Systemen wesentliche Erkenntnisse gewonnen: modulare Struktur, Dekompositionsprinzipien, Datenkapselung, Dokumentation oder auch Änderbarkeitsaspekte großer Software-Systeme wurden erforscht und in Industrieprojekten ermittelt.

Ein bedeutender Faktor in der heutigen Software-Entwicklung ist die Größe und Komplexität der zu entwickelnden Software-Systeme, sodass es nicht nur guter Programmierkenntnisse, sondern vielmehr guter Dekompositions-, Strukturierungs- und Kapselungskenntnissen sowie viel Erfahrung bedarf, um qualitativ gute Software-Systeme zu bauen.

Clements et al. 2002 führen drei Gründe an, warum die Software-Architektur für große, komplexe, softwareintensive Systeme so wichtig ist:

1. *Software-Architektur ist ein Kommunikationsvehikel für alle am System Beteiligten*: Als gemeinsame Abstraktion eines Software-Systems dient es nahezu allen System-Beteiligten als Basis für ein wechselseitiges Verstehen, zur Festlegung von Entscheidungen sowie zur Kommunikation über das System selbst.
2. *Software-Architektur ist die Explizitmachung der frühen Design-Entscheidungen*: Als früheste Dokumentation des zu erstellenden Software-Systems dient sie zur Reihung der einzelnen technischen Interessen und Wichtigkeiten und sie hat somit maßgeblichen Einfluss auf die späteren Qualitätsattribute des Systems. Die Kompromisse zwischen Performance und Sicherheit, Wartbarkeit

und Zuverlässigkeit oder den Kosten der Entwicklung und den Kosten der Erweiterung und Evolution sind darin begründet.

3. *Software-Architektur ist eine wieder verwendbare, übertragbare Abstraktion eines Software-Systems*: Die Software-Architektur stellt ein einfaches Modell der Komponenten und ihrer Interaktionen dar, das leicht zu verstehen sein soll. Ein solches Modell kann unter ähnlichen Anforderungen wiederum teilweise oder vollständig verwendet werden und somit Wiederverwendung im größeren Umfang sowie Produktlinien-Entwicklung ermöglichen.

All diese Aspekte sind wesentlich in der Betrachtung einer Software-Architektur und für die Herangehensweise in ihrer Erstellung. Zu beachten ist dabei jedoch, dass mit dem Begriff „Architektur" zahlreiche abweichende Bedeutungen in Verbindung gebracht werden: Informations-Architektur, Prozess-Architektur, System-Architektur, Geschäftsfeld-Architektur, Informationssystem-Architektur u.v.a.m. Gemeinsam ist diesen Begriffen die Verwendung von Architektur als Strukturierung und Organisationsprinzip von Information, Prozessen etc. und die Abbildung auf Komponenten und deren Zusammenspiel. Viele Informationssystem-Architekturen bestehen beispielsweise aus Benutzerschnittstelle, Geschäftslogik und Datenbank(en), die auf dreiteilige (*3-tier*) Client-Server abgebildet werden. Eine genauere Definition der Terminologie folgt in einem späteren Kapitel.

1.5 Was ist eine „gute" Software-Architektur?

Betrachtet man die Entwürfe mehrerer Software-Architekten, die unter den gleichen Anforderungen erstellt worden sind, so stellt sich die Frage nach der Bewertung und Qualifizierung einzelner Entwürfe von Software-Architekturen: Gibt es *gute* Software-Architekturen und was zeichnet solche aus? Welcher Entwurf ist der *richtige* unter den gegebenen Anforderungen und Rahmenbedingungen?

Es gibt keine für sich gute oder schlechte Architektur, sondern nur Architekturen, die ihren Zweck mehr oder weniger gut erfüllen. Dies zu bewerten, ist die Kunst. Mit der Festlegung einer Software-Architektur sowie deren Dokumentation ist es nunmehr explizit möglich, eine Evaluierung von Entwürfen durchzuführen und sich danach für den besseren Entwurf zu entscheiden. Zahlreiche Methoden und Techniken wurden für die Evaluation von Software-Architekturen in den letzten Jahren entwickelt, z.B. ATAM – *Architecture Tradeoff Analysis Method* (Kazman et al. 2000; Kazman et al. 1998), SAAM – *Software Architecture Analysis Method* (Kazman et al. 1994) oder ARID – *Active Reviews for Intermediate Designs* (Clements 2000).

Ein Buch, das sich speziell auf die Evaluation von Software-Architekturen konzentriert, ist (Clements et al. 2002). Darin wird dargelegt, dass eine Architektur tatsächlich in ihrem gegebenen Kontext (Anwendungsbereich und Anforderungen) evaluiert werden kann und welche Qualitätsattribute dafür jeweils herangezogen werden können.

Bass, Clements und Kazman haben in (Bass et al. 1998) eine (naturgemäß unvollständige) Menge von Richtlinien zur Erstellung einer Software-Architektur definiert, die wir hier sinngemäß wiedergeben wollen.

Hinsichtlich der *Vorgehensweise* gibt es folgende Richtlinien:

- Eine Architektur soll das Produkt eines einzelnen Architekten oder einer kleinen Gruppe von Architekten sein.
- Das Team der Software-Architekten sollte klar formulierte technische Anforderungen sowie eine Liste von angestrebten qualitativen Eigenschaften (wie z.B. Portabilität) haben.
- Die Software-Architektur sollte in einer für alle Beteiligten bekannten (verständlichen) Notation dokumentiert sein.
- Die Festlegung der Architektur sollte alle Beteiligten in mehreren Durchläufen aktiv einbinden.
- Die Architektur sollte quantitativ (wie z.B. maximaler Durchsatz) analysiert und auf qualitative Eigenschaften (wie z.B. Portabilität, Änderbarkeit) überprüft werden.
- Die Architektur sollte ein Grundgerüst für die spätere Implementierung festlegen, in dem die Kommunikationspfade vorerst mit minimaler Funktionalität erprobt werden. Dieses Grundgerüst kann dann für eine inkrementelle Entwicklung des Systems verwendet werden.
- Die Architektur sollte etwaige Ressourcenbegrenzungen (z.B. Minimum an Netzwerkbelastung) aufzeigen und deren Lösung sollte ausdrücklich festgehalten, allen Beteiligten weitergegeben und stets aktualisiert werden.

Auch im Bereich der *Struktur* lassen sich Richtlinien definieren:

- Die Architektur sollte wohldefinierte Module aufweisen, deren funktionale Verantwortlichkeiten den Prinzipien des *information hiding* und der Datenkapselung folgen. Jedes Modul sollte eine klar definierte Schnittstelle haben, die veränderliche Teile nach außen verbirgt (wie z.B. spezielle Algorithmen oder Datenstrukturen zur Erbringung der Funktionalität).
- Die Trennung von Zuständigkeiten (*separation of concerns*) in Module sollte eine möglichst lose gekoppelte Entwicklung der einzelnen Teile durch verschiedene Teams ermöglichen.
- Eigenarten der Infrastruktur sollten in Modulen gekapselt sein, sodass eine Änderung der Plattform auf einige wenige Teile des Systems konzentriert bleibt.
- Die Architektur sollte nie von einer bestimmten Version eines Produkts oder eines Tools abhängen. Sollte es dennoch eine unvermeidliche Abhängigkeit geben, so sollten bei einem Wechsel des Produkts oder Tools nur geringfügige Modifikationen erforderlich sein.

- Module, die Daten produzieren, sollten von jenen, die Daten konsumieren, getrennt sein. Dies erhöht die Änderbarkeit, da Modifikationen sich meist auf produzierende oder auf konsumierende Teile konzentrieren.
- Die Architektur sollte wenige und einfache Interaktionsmuster aufweisen. Das System sollte die gleichen Dinge auf dieselbe Art und Weise tun. Dies erhöht die Verständlichkeit, reduziert Entwicklungszeit, erhöht Zuverlässigkeit und steigert die Änderbarkeit. Es zeigt ebenfalls die konzeptuelle Integrität einer Software-Architektur.

Bei der Betrachtung einer konkreten Software-Architektur zeigt sich bald, welchen dieser Kriterien der Entwurf jeweils folgt. Diese einfachen Kriterien eignen sich sehr gut dafür, die Qualitätsattribute einer Software-Architektur zu beurteilen und diese Erfahrungen in den Entwurfsprozess dann einzubringen. Auch sind solche Checklisten besonders gut verwendbar, da sie wesentliche Aspekte einer architekturellen Evaluation abdecken und dem Architekten somit ein wertvolles und doch einfaches Rahmenwerk in die Hand geben.

1.6 Software-Architektur versus System-Architektur

Architekturen für IT-Systeme unterscheiden sich wesentlich von Software-Architekturen; sie bestehen aus einer Menge von Hardware- und Software-Komponenten, die gemeinsam einen Netzwerk- bzw. Internet-Dienst zur Verfügung stellen. Diese Perspektive einer System-Architektur schließt viele Computer, Speichersysteme sowie Netzwerk-Komponenten ein, die durch ihr Zusammenwirken eine Reihe von verfügbaren, sicheren und skalierbaren Diensten aus dem Geschäftsbereich abwickeln.

Dabei konzentriert man sich auf die Modellierung dieser speziellen Komponenten, ohne auf deren interne softwaretechnische Realisierung (i.e. Software-Architektur) einzugehen. Die softwaretechnische Beschreibung werden wir im Rahmen von Web-Services noch in einem späteren Kapitel kennen lernen.

Bei System-Architekturen meint man sehr oft Mehrschichten-Architekturen (*n-tier architectures*), wobei der häufigste Vertreter die 3-Schichten Architektur ist, die aus einer Client-seitigen Präsentationsschicht, einer Applikationsschicht und einer Datenbank-Schicht besteht. Damit werden Daten, Anwendungslogik und Darstellung (i.e. Präsentation) voneinander getrennt betrachtet. Anwendungslogik und Datenschicht können somit wieder verwendet und die Client-seitige Programmierung wesentlich erleichtert werden. Nachstehende Abbildung stellt eine 3-Schichten-Architektur dar.

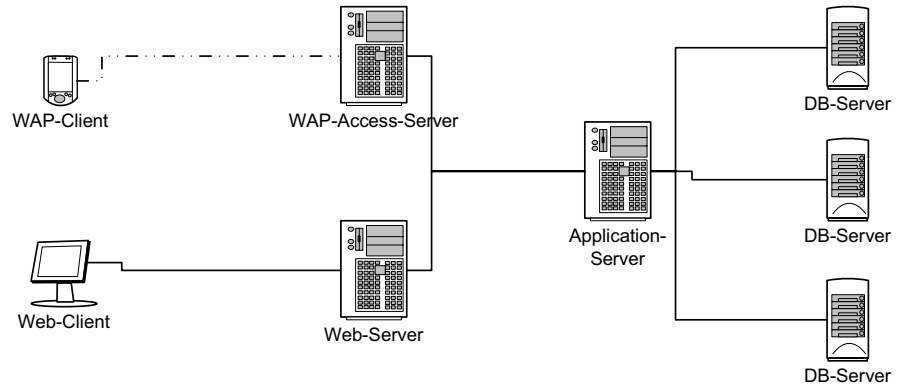

Abb. 1.1. Eine 3-Schichten-System-Architektur

Verissimo u. Rodrigues 2001 beschreiben in ihrem Buch zahlreiche Details von verteilten Systemen für System-Analytiker und beziehen sich dabei auf obige Komponenten von System-Architekturen. In diesem Buch konzentrieren wir uns auf Software-Architekturen und kommen bei den Standardarchitekturen (z.B. Web-Services) nochmals auf die Beschreibung von System-Architekturen zurück.

Literatur

Bass L, Clements P, Kazman R (1998) Software Architecture in Practice. Addison Wesley Longman, Inc.

Blair GS, Stefani JB (1998) Open Distributed Processing and Multimedia. Addison Wesley Longman Ltd.

Clements P (2000) Active Reviews for Intermediate Designs. Technical Note CMU/SEI-2000-TN-009. Software Engineering Institute, Carnegie Mellon University

Clements P, Kazman R, Klein M (2002) Evaluating Software Architectures: Methods and Case Studies. SEI Series in Software Engineering, Addison-Wesley

Kazman R, Klein M, Barbacci M, Lipson H, Longstaff T, Carriere SJ (1998) The Architecture Tradeoff Analysis Method. In Proceedings of the 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 1998), pp 68-78, Monterey, USA, IEEE Computer Society

Kazman R, Klein M, Clements P (2000) ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University

Loftus CW, Sherratte EM, Gautier RJ, Grandi PAM, Price DE, Tedd MD (1996) Distributed Software Engineering. Prentice-Hall

Putman JR (2001) Architecting with RM-ODP. Prentice Hall

Verissimo P, Rodrigues L (2001) Distributed Systems for System Architects. Kluwer Academic Publishers