mehr zum thema:
blog.accso.de

SOFTWAREARCHITEKTUR IN AGILEN PROJEKTEN: VON KATHEDRALEN UND IT-SYSTEMEN

der autor

Dr. Markus Voß
(voss@accso.de)
ist Geschäftsführer des Darmstädter Software- und Beratungshauses Accso – Accelerated Solutions GmbH. Er ist Autor zahlreicher Veröffentlichungen und hält einen Lehrauftrag für Softwarearchitektur und SOA an der Hochschule Darmstadt.

Immer wieder fällt mir auf, dass Informatiker den Begriff „Architektur" in ihrer Domäne der Softwaresysteme etwas unreflektiert verwenden. Die am weitesten verbreitete Definition ist (in Anlehnung an IEEE 1471) wohl in etwa die einer „grundlegenden Organisation eines Systems, beschrieben durch dessen Komponenten und die Beziehungen, in denen diese zueinander stehen". Meiner Meinung nach ist diese Definition aber nicht ganz ideal. Was ich für eine bessere Definition halte und warum diese Diskussion gerade im Bereich Effizienz und bei der Betrachtung agiler Softwareentwicklung von Bedeutung ist, möchte ich in diesem Beitrag darstellen.

Abb. 1: Die Kathedrale von León.

Als ich im Jahr 2008 das letzte Drittel des klassischen Jakobswegs nach Santiago de Compostela gewandert bin, war mein Startpunkt die Stadt León in der autonomen Region Kastilien und León im Norden von Spanien. Die dortige Kathedrale (siehe Abbildung 1) hat mich sehr fasziniert. In ihrem Inneren hatte ich – mehr als in allen anderen Kathedralen, die ich zuvor gesehen hatte – tatsächlich den Eindruck, dass sie aus mehr Glas als Stein erbaut wurde. Diese besondere Wirkung gotischer Architektur ist in der Kathedrale von León in besonderer Weise auch heute, über 700 Jahre nach ihrer Erbauung, noch erlebbar und ich bilde mir ein, genau das wollten die Architekten damals erreichen.

Auch als IT-Professionals verwenden wir häufig das Wort „Architektur". Im Rahmen der Entwicklung von Softwarelösungen malen wir für alle möglichen Ebenen beziehungsweise Sichten Graphiken mit Kästchen und Pfeilen und nennen das Ergebnis jeweils Architektur. Das ist in der täglichen Praxis ja meistens auch kein Problem, denn üblicherweise ist intuitiv klar genug, was das Bild aussagen soll und welchen Informationsmehrwert diese Beschreibung einer Architektur damit für den Prozess der Entwicklung bringt. Qualitativ ist man im Projekt damit zumeist auf der sicheren Seite.

Wenn es nun aber quantitativ um die konkrete Effizienz der Vorgehensweise in der Entwicklung geht, stellt sich schnell die Frage, wie viel Architektur denn sinnvollerweise definiert werden soll. Wie viel Architektur muss ich dem Entwicklungsprojekt vorgeben, damit nicht irgendetwas gebaut wird, was am Ende mit hohen Aufwänden wieder grundlegend umgebaut werden muss? Welcher Teil der Architekturdefinition kann getrost entfallen, da er entweder so trivial ist, dass es auf jeden Fall so gebaut wird, oder weil er die Kreativität und letztlich auch die Effizienz der Bauenden einschränkt? Welches Gewicht hat die Definition einer Architektur im agilen Entwicklungsprozess und wie viel davon kann man dem sich selbst organisierenden Team überlassen?

Eine schärfere Definition von Architektur

In diesem Zusammenhang wird es plötzlich wichtig, einen möglichst scharf definierten Begriff von Architektur zu haben. Mangels klar vereinbarter Begrifflichkeit tauchen genau in diesen Diskussionen dann aber immer wieder Begriffe wie „Architekturstil" (den zumindest sollte man vorgeben, was auch immer das nun wieder genau ist) oder „Referenzarchitektur" (zumindest sollte man das Rad nicht neu erfinden) auf.

Ich plädiere nun im Gegensatz dazu dafür, den Begriff der Architektur selbst erst noch ein wenig schärfer zu fassen. Dazu möchte ich allerdings keine neue Definition erfinden, sondern den Begriff schlicht so verwenden, wie er im Bauwesen schon immer verwendet wurde. Zum Beispiel spricht man dort eben von der „gotischen Architektur". Diese zeichnet sich sowohl durch die Verwendung bestimmter Arten von Bauelementen aus (z. B. Spitzbogen, Kreuzrippengewölbe,

Strebewerk und Strebepfeiler), als auch durch eine bestimmte Art und Weise, diese zu verwenden (z. B. erlauben Kreuzrippen und Strebewerk als tragende Elemente eine Minimierung der Wandflächen und eine Maximierung der Fensterflächen). Hier noch ein anderes Beispiel: Die „Architektur einer Basilika" zeichnet sich ebenfalls durch Arten von Elementen aus (Langhaus, Querhaus, Vierung, Chorraum, Apsis) und auch hier ist die Art der Anordnung charakteristisch für die Architektur. Bemerkenswert an dieser Stelle ist, dass sich solchermaßen definierte Architekturen auch miteinander kombinieren lassen, d. h. die „Architektur einer gotischen Basilika" ist klar beschrieben, indem man die Festlegungen der beiden zuvor genannten Architekturen addiert.

Überträgt man diese Überlegungen auf die Domäne der Softwaresysteme komme ich zur folgenden *Definition* von Architektur:

Architektur ist die grundlegende Organisation eines Systems, beschrieben durch dessen Arten von Komponenten und die Arten von Beziehungen, in denen diese zueinander stehen, beziehungsweise der Art und Weise, wie diese zusammenwirken. Architektur beinhaltet damit die Prinzipien, an denen sich das Design des Systems orientiert.

Dazu zwei Anmerkungen:

- *Komponente* bedeutet hier schlicht Teil eines System im Sinne einer allgemeinen Systemtheorie. Ein System besteht aus Teilen – und diese nennen wir Komponenten. Das System als Ganzes ist beschrieben, wenn zum einen seine Komponenten mit ihren Eigenschaften beschrieben sind und zum anderen die Beziehungen, in denen diese zueinander stehen.
- *Beziehungen* zwischen Komponenten fasse ich dabei sehr allgemein auf. Es geht natürlich nicht nur um statische Anordnungsbeziehungen im Raum, wie sie die Beispiele aus dem Bauwesen nahelegen, sondern auch um dynamisch veränderliche (Evolution der Architektur) oder auch die Dynamik des Gesamtsystems bestimmende Beziehungen, d. h. auch um das Zusammenwirken beziehungsweise die Interaktion.

Die schärfere Definition im Test

Anders als in der Definition zu Beginn dieses Artikels, die auch eine andere Verwendung zulässt, beschreibt eine Architektur, wie ich sie definiere, *immer* eine ganze Klasse von Systemen und nie nur ein einzelnes System. Im Bauwesen ist das genauso: Die Systeme „Notre-Dame de Reims", „Notre-Dame de Paris" oder eben „Santa María de Regla de León" haben alle die „Architektur einer gotischen Basilika". Durch diese Definition ist die Architektur eines Systems klar abgegrenzt von dessen Design. In der Architektur interessieren nur die Arten von Komponenten. Im Design interessiert jede einzelne Komponente und gegebenenfalls auch ihr innerer Aufbau beziehungsweise ihr spezifisches Verhalten: In der IT-Domäne wird das oft auch detailliertes Design oder Komponentendesign genannt.

Um diese Sicht auf Architektur zu testen, werde ich im Folgenden verschiedene der in der IT gängigen Architekturen einordnen und schauen, ob sie sich auf diese Art scharf definieren lassen:

	Gotische Architektur	Quasar Architektur
Arten von Komponenten	Spitzbogen Kreuzrippengewölbe Strebewerk Strebepfeiler	A-Komponenten A-Fälle A-Verwalter A-Entitäten
Arten von Beziehungen und Interaktionen	Kreuzrippen und Strebewerke mit Strebepfeilern als tragende Elemente erlauben eine Minimierung der Wandflächen und eine Maximierung der Fensterflächen.	A-Fälle stellen die Schnittstellen der A-Komponenten dar und bedienen sich dazu der A-Verwalter, die ihrerseits alle A-Entitäten eines Typs kapseln .

Tabelle 1: Analogie von Architektur im Bauwesen und Softwarearchitektur.

- *Quasar:* Der Name steht für *Quality Software Architecture* (vgl. [Sie04]) und soll Eigenschaften einer bestimmten Softwarearchitektur festlegen. Die Systeme, um die es geht, sind Softwaresysteme und die Teile sind – je nach Sichtweise – entweder allgemein Codeelemente oder Komponenten im softwaretechnischen Sinne (mit Schnittstellen). Tatsächlich erfüllt Quasar hier den Anspruch an eine Architektur: Bei den allgemeinen Codeelementen wird zwischen den Arten *O*-, *A*-, *T*- und *R*-Software unterschieden (dort Blutgruppen genannt) und es gibt Regeln für deren Anordnung, insbesondere die Vermeidung einer Kombination von *A*- und *T*-Software. Bei den Komponenten im softwaretechnischen Sinne gibt es zusätzlich einzelne Architekturen für unterschiedliche Teile einer Softwarelösung, beispielsweise für den Anwendungskern. Die Arten von Komponenten heißen dort *A*-Fall, *A*-Verwalter und *A*-Entität. Die Art und Weise der Interaktion läuft von den A-Fällen über die A-Verwalter zu den A-Entitäten (Schichtung). Am Client gibt es Dialograhmen mit mehreren zugeordneten Dialogen. Quasar liefert Architektur genau in der von mir definierten Lesart (siehe Tabelle 1).
- *SOA:* Das Akronym steht für *Serviceorientierte Architektur*. Die gängigen Definitionen argumentieren nicht über Arten von Komponenten und Beziehungen – entsprechend unklar sind diese Definitionen häufig. In „Quasar Enterprise" (vgl. [Eng08]) haben wir einen SOA-Begriff formuliert, der sich aber sehr wohl so beschreiben lässt: Die Arten von Komponenten sind solche, die sich ausgehend von den Services des Geschäfts und der Geschäftsstrategie bestimmen lassen, und die Art und Weise der Interaktion ist die der „angemessen losen Kopplung". Eine kurze und knackige Definition.
- *Web-Anwendung:* Arten von Komponenten sind Client-Komponenten auf

der Basis von Web-Browsern und Server-Komponenten auf der Basis von Web- oder App-Servern. Die Interaktion erfolgt unmittelbar oder mittelbar über HTTP.

- *Model-View-Controller:* Die Komponentenarten sind *Model*, *View* und *Controller*. Die Art und Weise der Interaktion ist: *Controller* an *Model* und *Model* an *View*, wobei Letzteres üblicherweise dem *Observer-Pattern* folgt.
- *Web-Services-Architektur:* Hier sind die Arten von Komponenten *Service Requestor, Service Provider* und *Service Registry*. Die Art und Weise der Interaktion erfolgt gemäß *publish, find and bind*.
- …

Es sieht also so aus, als trüge diese Definition von Architektur sehr gut in der Domäne der Softwaresysteme. Aber wo ist jetzt der konkrete Nutzen jenseits der akademischen Diskussion?

Architektur und Design: zwei unterschiedliche Dinge

Ein wichtiger Punkt liegt in der damit möglichen klaren Unterscheidung von Architektur und Design, wie oben eingeführt. Ein gut gemachtes Design eines Softwaresystems folgt insbesondere der Fachlichkeit – also den funktionalen Anforderungen. Eine adäquate Gliederung der Fachlichkeit findet man im Idealfall *1:1* in der Gliederung des Systems in seine individuellen Komponenten wieder, und auch der innere Aufbau beziehungsweise das spezifische Verhalten jeder einzelnen Komponente ist insbesondere durch die funktionalen Anforderungen festgelegt.

Die Architektur eines Softwaresystems leitet sich hingegen vor allem aus den nicht-funktionalen Anforderungen ab. Bestes Beispiel dafür ist wieder Quasar: Die spezifische Strukturierung von Softwaresystemen in die dort definierten Arten von Komponenten resultiert vor allem aus der Anforderung nach langfristiger Wartbarkeit, Erweiterbarkeit oder Änderbarkeit der Software. Andere nicht-funktionale Anforderungen, wie z. B. die initiale Entwicklungsgeschwindigkeit, spielen hingegen weniger eine Rolle. Abbildung 2 illustriert Architektur und Design als unterschiedliche Ergebnistypen im Softwareentwicklungsprozess.

Diesen Zusammenhang mit nicht-funktionalen Anforderungen kann man auch für die anderen oben kurz angerissenen

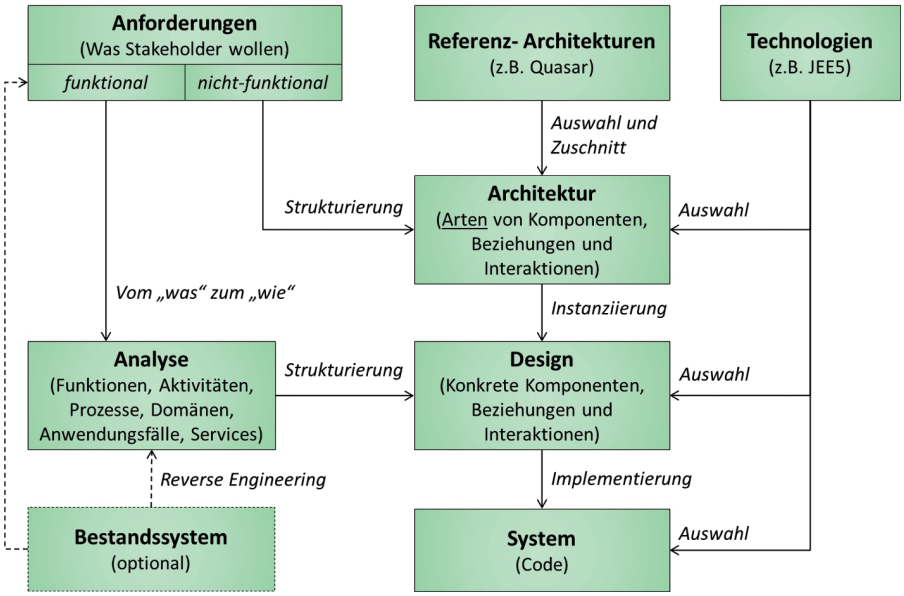

Abb. 2: Architektur und Design als Ergebnistypen der Softwareentwicklung.

Architekturen zeigen: In einer SOA liegt beispielsweise die nicht-funktionale Anforderung in der Ausrichtung der IT auf das Geschäft und in der Möglichkeit zur agilen Adaption an sich verändernde Geschäftsprozesse. Oder eine Web-Anwendung folgt der nicht-funktionalen Anforderung der Vermeidung von Softwareverteilung. Sogar für die Analogie zur gotischen Kathedralen lässt sich dieser Zusammenhang bis zu einem gewissen Punkt herstellen: Funktional muss die Kirche den Gottesdienst ermöglichen, in der gotischen Kathedrale passiert das zudem (= nicht-funktional) bei bestem Licht. Überstrapazieren darf man die Analogie allerdings nicht, denn gerade im Bauwesen resultiert vieles an Architektur auch aus funktionalen Anforderungen.

Architektur als angemessener Trade-Off

Eine Softwarearchitektur ist dann ideal, wenn sie die gewichtete beziehungsweise priorisierte Menge aller nicht-funktionalen Anforderungen optimal erfüllt – das ist letztlich immer ein vorhabenspezifischer und angemessener Trade-Off. Ein ganz ähnlicher Denkansatz liegt auch Methoden wie *ATAM (Architecture Tradeoff Analysis Method)* des SEI (vgl. [SEI12]) zu Grunde. Architekturarbeit betrachtet explizit alle sich potenziell widersprechenden, nicht-funktionalen Anforderungen und folgt dem Optimum im Sinne eines Trade-Offs. Dieser Trade-Off ist in jedem konkreten Entwicklungsprojekt etwas anders – um ihn durchzuführen, benötigt ein Architekt viel Erfahrung.

Aus all diesen Überlegungen leiten sich nun mit Blick auf Entwicklungseffizienz und *beschleunigte Softwaretechnik* aus meiner Sicht drei wesentliche Aussagen beziehungsweise *Best Practices* ab:

- Architektur vorab definieren.
- Architekt im Team sicherstellen.
- Referenzarchitekturen zuschneiden.

Best Practice 1: Architektur vorab definieren

Bei der genauen Erfassung der funktionalen Anforderungen bewährt es sich oft, iterativ vorzugehen. Dabei versucht man gar nicht im Vorfeld, alle Anforderungen zu verstehen und in eine vollständige Spezifikation beziehungsweise ein detailliertes Design des Systems umzusetzen. Vielmehr werden nur Teile genauer erfasst und dann direkt in Software umgesetzt. Die existierende Software hilft den Stakeholdern, die Anforderungen an das System in der nächsten Iteration besser zu beschreiben. In Summe ist dieses Vorgehen dadurch häufig effizient.

Bei den nicht-funktionalen Anforderungen verhält es sich meistens genau anders. Erfahrungsgemäß ist es hier zumeist am effizientesten, diese vorab möglichst gut verstanden zu haben. In Iterationen über die nicht-funktionalen Anforderungen lernen zu wollen, ist nur in komplexen Ausnahmefällen ein guter Ansatz und dann haben die Iterationen zumeist den Charak-

ter von Prototypen zur Architektur-Evaluation. Zumeist ist es deutlich effizienter, sich bereits im Vorfeld Gedanken zum Trade-Off der unterschiedlichen nicht-funktionalen Anforderungen zu machen und daraus die Architektur abzuleiten. Das gilt insbesondere, weil Architektur ja die Arten der Komponenten festlegt und sich jede Veränderung beziehungsweise Erweiterung der Architektur im aufwändigen Umbau aller bislang entwickelten konkreten Komponenten niederschlagen kann. Erfahrungsgemäß ist das nur ganz selten effizient. In diesem Sinne deckt sich meine Definition von Architektur auch mit der von Grady Booch: „All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change" (vgl. [Boo06]).

Auch für agile – und gerade für effiziente – Projekte lautet die logische Forderung daher: *Entwickle die Architektur des Softwaresystems soweit möglich vorab und nicht im Rahmen der Iterationen.*

Das klingt erst einmal nach einem Widerspruch zum agilen Ansatz. Schließlich betont das agile Manifest (vgl. [Bec01]), dass „die besten Architekturen, Anforderungen und Designs von sich selbst organisierenden Teams stammen". In etablierten agilen Vorgehensmodellen wie Scrum (vgl. [Sch01]) bedeutet das eine Erarbeitung im Rahmen der Iterationen. Mit der schärferen Architekturdefinition von oben stimmt das so pauschal aber eben nicht mehr. Zwar dürfen sich iterativ noch gewisse Verbesserungen an der Architektur ergeben, das Wesentliche sollte im Sinne der obigen Effizienzüberlegungen erfahrungsgemäß aber vorgedacht werden, um vermeidbare *Refactorings* (vgl. [Fow99]) im agilen Prozess auch wirklich zu vermeiden. Das Design hingegen darf und soll sich iterativ entwickeln – emergentes Design (vgl. [Cav00]) als Bestandteil agiler Softwareentwicklung ist kein Widerspruch zum Vorziehen der Architekturdefinition.

Dieses Vorziehen der Architekturdefinition lässt sich natürlich mit den gängigen iterativen Ansätzen verbinden – sei es durch ein den Iterationen vorgelagertes Basiskonzept für das System in einem gesteuert-inkrementellen Vorgehen oder durch eine hinreichend umfangreiche *Envisioning*-Phase im Scrum-Projekt. In keinem Fall sehe ich darin ein Anti-Pattern. Abbildung 3 illustriert den Sachverhalt.

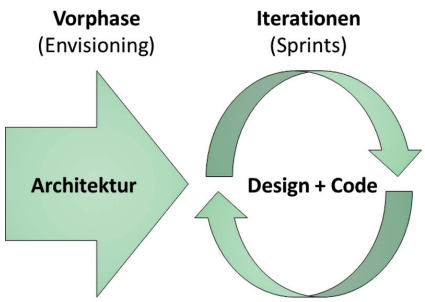

Abb. 3: Architektur und Design im iterativen/agilen Projekt.

Best Practice 2: Architekt im Team sicherstellen

Es ist wichtig, das grundsätzliche Vorgehensmodell im Bereich der Entwicklung von Architektur, Design und Implementierung (in Scrum beispielsweise „Entwicklung iterativ in Sprints") gedanklich von Aufgabenverteilung und Rollenspiel im Projektteam zu trennen (in Scrum beispielsweise „Entwicklung durch das sich selbst organisierende Team").

Zu ersterem habe ich bereits festgestellt, dass Architekturdefinition vor die Iterationen gehört. Zum zweiten stellt sich die Frage, inwieweit es zur effizienten Umsetzung der Architekturdefinition eines ausgewiesenen Architekten beziehungsweise dieser Rolle im Projekt bedarf. In Scrum beispielsweise gibt es diese Rolle nicht. Hier baut man auf das sich selbst organisierende Team für alle inhaltlichen Aufgaben, getreu dem zuvor bereits zitierten Motto aus dem agilen Manifest.

Die obigen Ausführungen zeigen jedoch zumindest ganz deutlich, dass die Architekturdefinition eine wesentliche, den Erfolg des Projekts bestimmende Phase ist und dass es aufgrund der inhärenten Komplexität des Trade-Offs auch umfangreicher Erfahrung bedarf. Ganz nach dem Motto, „Architekt darf sich nur nennen, wer nachgewiesenermaßen mehrfach schon erfolgreich als Architekt gearbeitet hat", ist es nur folgerichtig, in allen nicht völlig unkritischen Entwicklungsvorhaben die Beteiligung eines Architekten zu fordern.

Daher lautet auch für agile und gerade für effiziente Projekte die logische Forderung: *Stelle sicher, dass du einen ausgewiesenen Architekten im Team hast.* Ob dieser Architekt nun die explizite Rolle „Chefarchitekt" und damit explizit auch die mit der Rolle verbundene Verantwortung bekommt oder ob er als „primus inter pares" mit fachlicher Führungskompetenz im sich selbst organisierenden Team qua Erfahrung den Job übernimmt, ist dann im zweiten Schritt abhängig von weiteren Projektcharakteristika zu klären.

Best Practice 3: Referenzarchitekturen zuschneiden

Referenzarchitekturen sind gegebene Architekturen, an denen man sich bei der Architekturdefinition im konkreten Fall orientieren kann. Ein bereits genanntes Beispiel für eine solche Referenzarchitektur ist Quasar als Referenzarchitektur für betriebliche Informationssysteme.

Ideal im Hinblick auf Effizienz ist es natürlich, wenn ich meine Zielarchitektur weitgehend aus Referenzarchitekturen komponieren kann (wie im Sinne der „gotischen (+) Basilika") und dabei jede einzelne Referenzarchitektur einfach ohne weitere Veränderungen wiederverwendet werden kann. Erfahrungsgemäß ist das aber nur selten möglich. Grund hierfür ist auch hier wieder die Forderung nach der Architektur im Sinne eines angemessenen Trade-Offs. Ein hohes Maß an Entkopplung und Abstraktion, wie es beispielsweise Quasar vorsieht, brauche ich gerade dann, wenn langfristige Wartbarkeit, Erweiterbarkeit und Änderbarkeit ganz klar die höchste Priorität haben. Häufig wird diese Priorisierung aber gar nicht explizit hinterfragt, sondern die gegebene Referenzarchitektur wird unreflektiert verwendet. Am Ende sind die Entwicklungsaufwände hoch und die Entwickler sprechen von Over-Engineering, weil es offensichtlich doch noch konkurrierende Anforderungen gab. Ohne Referenzarchitekturen ist man nicht effizient, aber ohne den adäquaten Zuschnitt derselben eben auch nicht.

Daher lautet auch für agile und gerade für effiziente Projekte die logische Forderung: *Arbeite mit Referenzarchitekturen, aber stelle sicher, dass diese entsprechend den Trade-Off-Überlegungen angemessen zugeschnitten werden.*

Dabei gilt: Keine Referenzarchitektur zu verwenden, ist immer noch besser, als eine zu den nicht-funktionalen Anforderungen nicht passende Referenzarchitektur zu Grunde zu legen. Das gilt grundsätzlich auch für inzwischen weit verbreitete Referenzarchitekturen wie Quasar, deren Bedeutung auch im agilen Umfeld gesehen wird (vgl. [Lin09]). Tatsächlich sind Anforderungskonstellationen denkbar, in denen ein System explizit nicht gemäß der

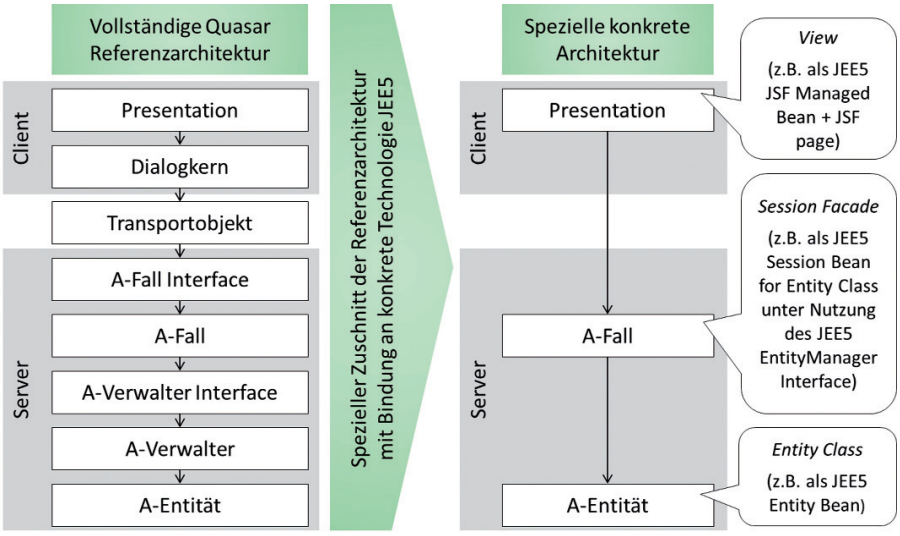

Abb. 4: Spezieller Zuschnitt der Referenzarchitektur Quasar.

Quasar-Referenzarchitektur gebaut werden sollte. Vor allem aber sind viele Konstellationen denkbar, in denen die Referenzarchitektur leichtgewichtig zugeschnitten werden sollte. Das kann beispielsweise durch einfachere Benutzungsschnittstellen ohne aufwändige Dialog/Dialograhmen-Strukturierung oder durch Verzicht auf die Entkopplung mittels Interfaces oder Transportobjekten erfolgen, wenn die Implementierungen nicht ausgetauscht werden müssen.

Beim Zuschnitt der Referenzarchitektur auf die konkrete Architektur spielt häufig auch die ausgewählte Technologie eine wesentliche Rolle. Viele Abstraktionsmechanismen in Quasar beispielsweise werden nicht benötigt, wenn eine homogene und moderne Technologie, wir z. B. „JEE5", eingesetzt werden kann. Ein schönes Beispiel liefert die sehr einfache Softwarearchitektur aus [Bie09]. Bei Einsatz von JEE5 und unter der vereinfachenden Annahmen, dass die Systemlösung ruhig mit der Technologie verheiratet werden darf, kann eine konkrete Architektur gewählt werden, die sich in der Komplexität von der Referenzarchitektur um Faktoren unterscheidet (siehe Abbildung 4). Es ist offensichtlich, dass wenig komplexe Architekturen für das Team, das auf dieser Basis entwickelt, besonders effizient sein können. Wird beim Zuschnitt also das richtige Maß gefunden, so können Referenzarchitekturen maßgeblich zur Steigerung der Effizienz in der Softwareentwicklung beitragen. ■

Literatur & Links

[Bec01] K. Beck et. al., Manifesto for Agile Software Development, 2001, siehe: www.agilemanifesto.org

[Bie09] A. Bien, Real World Java EE Patterns – Rethinking Best Practices, Press.adam-bien.com 2009

[Boo06] G. Booch, Blog, 2006, siehe: www.ibm.com/developerworks/mydeveloperworks/blogs/gradybooch

[Cav00] D. Cavallo, Emergent Design and learning environments: Building on indigenous knowledge, in: IBM Systems Journal, Vol. 39, Nr. 3/4, 2000

[Eng08] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, J. Willkomm, Quasar Enterprise – Anwendungslandschaften serviceorientiert gestalten, dpunkt.verlag 2008

[Fow99] M. Fowler, Refactoring – Improving the design of existing code, Addison-Wesley 1999

[Lin09] T. Lindner, M. Lippert, S. Roock, Kontroverse – Agilität und Architektur, it-agile 2009

[Sch01] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall 2001

[SEI12] SEI / CarnegieMellon, Architecture Tradeoff Analysis Method, 2012, siehe: www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

[Sie04] J. Siedersleben, Moderne Softwarearchitektur: umsichtig planen und robust bauen mit Quasar, dpunkt.verlag 2004