Architekturstile in der Praxis

Carola Lilienthal

Arbeitsbereich Softwaretechnik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg
Vogt-Kölln-Straße 30
D-22527 Hamburg
lilienthal@informatik.uni-hamburg.de

Abstract: Unternehmen wollen ihre Softwaresysteme heute in allen Branchen und insbesondere auch in der Logistik zu serviceorientierten Anwendungslandschaften mit sauber geschnittenen fachlichen Komponenten weiter entwickeln. Dieses Ziel kann nur erreicht werden, wenn sich die vorhandenen Softwaresysteme zu Serviceorientierten Architekturen (SOA) umbauen lassen und dabei auf Dauer wartbar und erweiterbar bleiben. Auf der Basis von achtundzwanzig Fallstudien wird in diesem Beitrag exemplarisch die aktuelle Situation in Unternehmen beschrieben, in denen einerseits komplexe und trotzdem wartbare und erweiterbar Softwaresysteme von mehr als 14 Mio. Lines of Code (LOC) eingesetzt werden. Andererseits findet man kleinere Softwaresysteme (100T LOC), die nur mit großer Mühe oder gar nicht an die modernen Anforderungen angepasst werden können. Welche Wege die Unternehmen einschlagen, um die Komplexität ihrer Softwaresysteme beherrschbar zu halten und sie damit für eine Erweiterung zu Serviceorientierten Architekturen fit zu machen, wird anhand der Fallstudien untersucht. Dabei wird deutlich, dass der jeweils eingesetzte Architekturstil einen entscheidenden Einfluss darauf hat, ob die Architektur über längere Zeit erhalten, der Architekturverfall aufgehalten und so der Grundstein für einen Umbau hin zu Serviceorientierten Architekturen gelegt werden kann.

1 Einleitung

Software, die langfristig eingesetzt werden soll, ist vielen Veränderungen unterworfen. Damit ein Entwicklungsteam diese Veränderungen in akzeptabler Zeit und Qualität durchführen kann, muss die Komplexität des Softwaresystems für das Entwicklungsteam beherrschbar bleiben. Komplexität kann eine Reihe von Ursachen haben, zu denen neben Multifunktionalität, der Mensch-Maschine-Schnittstelle und speziell die Softwarearchitektur eines Systems zählt [BR07]. Soll ein Softwaresystem auf eine serviceorientierte Architektur umgebaut werden, muss insbesondere die Komplexität der Softwarearchitektur so gering wie möglich sein.

Softwarearchitektur wird in der üblichen Architekturdiskussion mit Hilfe von Sichten und zugehörigen Modellen beschrieben [BCK03, HNS00, Kru95, RH06]. Die für diesen Beitrag entscheidende Sicht ist die statische Sicht. Mit Hilfe der statischen Sicht wird auf verschiedenen Granularitätsstufen die Aufteilung des Programmtextes in Elemente, die erlaubten Beziehungen und die Schnittstellen der Elemente festgelegt.

Um die Komplexität der statischen Sicht zu reduzieren, wurden seit den 1960er Jahren eine Reihe von Entwurfsprinzipien entwickelt, die noch heute ihre Gültigkeit haben [BR07]: Modularisierung [My78, Pa72], Abstraktion durch Schnittstellenbildung [Pa72] und Schichtung [Di68, Pa74]. Die konstruktive Umsetzung dieser Entwurfsprinzipien wird durch Architekturstile [SG96] - wie Schichtenarchitekturen [BCK03, Fo03, RH06], die Quasar-Architektur [Si04] sowie die WAM[1]-Architektur [Zü05] - geleistet, die diese Entwurfsprinzipien auf verschiedenen Ebenen und in unterschiedlichem Maße berücksichtigen. In diesem Beitrag wird anhand der Ergebnisse aus 28 Fallstudien deutlich, welche Architekturstile in der Praxis eingesetzt werden und wie ihr Potential zur Reduktion der strukturellen Komplexität genutzt wird.

2 Fallstudien

Die diesem Beitrag zugrunde liegenden Fallstudien wurden von März 2005 bis Februar 2007 durchgeführt und erstreckten sich über Systemgrößen zwischen 25T und 14 Mio LOC. Es wurden ausschließlich Java-Systeme analysiert, die zwischen zwölf und zwei Jahren alt waren. 26 der 28 Systeme werden in der Wirtschaft entwickelt und eingesetzt. Zwei Systeme werden an der Universität in der Lehre mit Studierenden als Kommunikations- und Lehrplattform verwendet. Die Einsatzkontexte der kommerziellen Systeme sind Bank, Versicherung, Abrechnungs- und Buchungssystem für Ticket-Verkauf und Vermietung, Koordination und Planung von logistischen Problemstellungen, CMS, Behördenverwaltung, Arztpraxis, CallCenter, Kommunikationsplattform, Simulation und Software-Entwicklungsumgebung. Für etwa die Hälfte der Unternehmen war klar, dass ihr Softwaresystem in den nächsten drei Jahren mit Hilfe von Services unternehmensintern oder in drei Fällen auch über Unternehmensgrenzen hinaus mit anderen Softwaresystemen vernetzt werden soll.

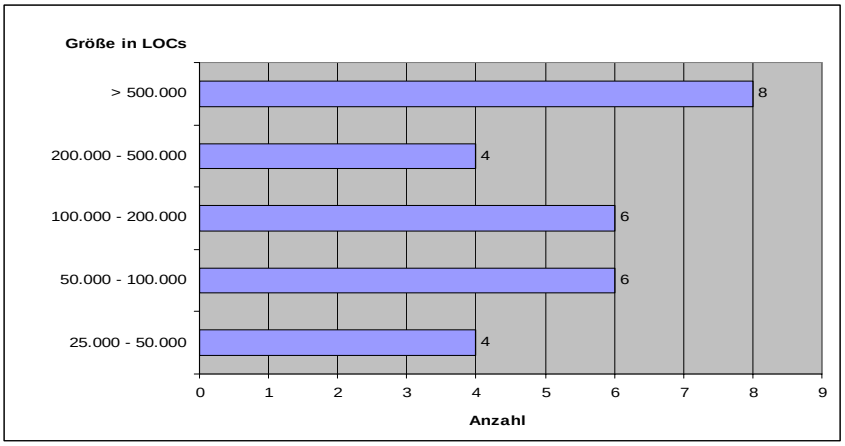

Abbildung 1: LOC der Fallstudien

[1] Quasar steht für Qualitätssoftware-Architektur und WAM für Werkzeug-Automat-Material

Im Rahmen der Fallstudien wurde der Source- und Byte-Code der Systeme mit Hilfe des Analysewerkzeugs Sotograph®[2] untersucht sowie Architekturdiskussionen und Interviews mit Architekten und Entwicklern geführt und ausgewertet.

3 Architekturstile

Einer der am häufigsten in der Literatur beschriebenen und in vielen Unternehmen eingesetzten Architekturstile ist die *Schichtenarchitektur* [BCK03, BD04, Fo03, HNS00, ZR06]. Bei einer Schichtenarchitektur werden Subsysteme[3] mit Schnittstellen definiert, die in Schichten eingeteilt werden. Für die Schichten wird gefordert, dass höhere Schichten nur Beziehungen zu den unter ihnen liegenden Schichten haben dürfen. Die einzelnen Schichten haben genau wie Subsysteme Schnittstellen, über die sie von anderen Schichten benutzt werden. Die von einer Schichtenarchitektur festgelegten architekturellen Einschränkungen beziehen sich nur auf die Ebene der Subsysteme und Schichten und machen keine Vorgaben für die Package- und Klassen-Ebene.

Für flexiblere Systeme wurden in den letzten Jahren vermehrt komplexere Architekturstile diskutiert und eingesetzt [Ev04, Fo03, Si04, Zü05]. Diese Architekturstile, so genannte *Referenzarchitekturen*[4], geben im Gegensatz zu Schichtenarchitekturen detailliertere Einschränkungen für die Strukturierung eines Softwaresystems vor und legen die Architektur insbesondere auch auf der Klassen-Ebene stärker fest. Bei einer Referenzarchitektur werden die Klassen in Kategorien, wie Entitäten, BusinessObjects, ValueObjects, Service, Werkzeuge etc. eingeteilt. Es werden Regeln angegeben, welche Zuständigkeiten die einzelnen Klassen haben und wie die Klassen miteinander interagieren dürfen [BKL06]. Beispielsweise dürfen BusinessObject-Klassen keine Service-Klassen benutzen. Gängige Referenzarchitekturen sind Quasar [Si04], Domain-Driven Design [Ev04] und die WAM-Architektur [Zü05].

In den Fallstudien wurden drei Systeme ohne erkennbaren Architekturstil, elf Systeme mit einer Schichtenarchitektur und sieben Systeme mit einer Referenzarchitektur identifiziert. Die restlichen sieben Fallstudien teilen sich in zwei Gruppen auf: Vier Systeme waren mit einer *projekteigenen Referenzarchitektur* entwickelt worden. Solche projekteigenen Referenzarchitekturen folgen zwar keiner in der Literatur veröffentlichten Vorgabe, enthalten aber ebenfalls Kategorien und Regeln für die Klassen [KG06]. Die restlichen drei Systeme besitzen einen in der Literatur nicht beschriebenen Architekturstil, der im Folgenden als *Subsystemarchitektur* bezeichnet wird. Bei diesen Systemen stehen die Dienste, die von den einzelnen Subsystemen an ihrer Schnittstelle angeboten werden, im Mittelpunkt, d.h. die Beschreibung der vom Subsystem angebotenen Klassen und ihr Verhalten auf einer höheren Ebene als Application Programming Interface (API).

---

[2] www.software-tomography.com

[3] Unter einem Subsystem wird hier eine Organisationseinheit in der statischen Struktur eines Softwaresystems verstanden, die ein oder mehrere Package-Bäume zusammenfasst. Alternativ sind auch die Begriffe Komponente oder Modul gebräuchlich [BCK03, RH06].

[4] In der Literatur werden synonym zu Referenzarchitektur die Begriffe Modellarchitektur und Musterarchitektur verwendet [RH06, Si04, ZR06, Zü05].

Die Beziehungen zwischen den Subsystemen werden bei diesen Architekturen über Erlaubt- und Verboten-Regeln beschränkt. In den drei Fallstudien führten diese Regeln weder dazu, dass eine hierarchische Schichtung der Subsysteme entstand, noch waren die Architekten bei der Analyse ihrer Systeme an einer Schichtung interessiert, sondern ihr Interesse galt der Frage, ob die Schnittstellen ihrer Subsysteme verletzt werden.

4 Architekturstile und Evolution

Aus der Auswertung der Fallstudien lassen sich Erkenntnisse über den Einsatz von Architekturstilen und ihren Einfluss auf die Evolution der Systeme gewinnen.

4.1 Zustand der Architekturen

Bei den Systemen ohne Architekturstil waren die Entwickler nicht in der Lage, die statische Architektur zu beschreiben. Lediglich die Deployment-Sicht und damit die Aufteilung des Systems in Client- und Server war den Entwicklern ein Begriff. Eins dieser Systeme mit 60T LOC hatte nur einen Entwickler, der auch für die Wartung zuständig ist. Bei den anderen beiden Systemen mit fast 1 Mio. LOC gab es ähnliche Organisationsformen, aber es existieren auch Teilbereiche im Sourcecode, die niemand mehr versteht und ändern will. Diese Systeme sind für die sie einsetzende Organisation ein kaum kalkulierbares Risiko, so dass eins der in den Fallstudien untersuchten Systeme inzwischen abgelöst wurde und sich die zwei anderen in einem massiven Refactoring-Prozess befinden, bei dem Teilbereiche neu implementiert werden müssen.

Alle analysierten Systeme, bei deren Konstruktion ein Architekturstil zugrunde gelegt worden war, wiesen im Code Verletzungen der jeweiligen Architekturregeln auf: Aufwärts-Beziehungen zwischen Schichten, Missachtung der Schnittstellen von Subsystemen sowie nicht erlaubte Beziehungen zwischen Klassen. War die Architektur in einem guten Zustand, so ließ sich ein Großteil dieser Verletzungen durch einfache Refactoring-Maßnahmen auflösen. War die Anzahl und Komplexität der notwendigen Refactoring-Maßnahmen hoch, so hatten auch die Architekten im Interview von Problemen bei Wartung und Erweiterung dieser Systeme gesprochen. Von den insgesamt elf Referenzarchitekturen war ein System in einem schlechten Zustand, bei den Schichten- und Subsystemarchitekturen jeweils knapp über die Hälfte.

4.2 Besonderheiten der Architekturstile

Obwohl Schichtenarchitekturen in der Literatur immer mit Schnittstellen einhergehen, lässt sich in den Fallstudien beobachten, dass Systeme unter 500T LOC nur wenige explizite Schnittstellen für Subsysteme und Schichten haben. I. d. R. bildet sich als erstes die Schnittstelle zwischen Client und Server heraus. Später, wenn das System größer wird, kommen weitere Schnittstellen für Subsysteme und Schichten hinzu. Anders verhält es sich bei den drei Fallstudien mit Subsystemarchitekturen: Hier sind bereits bei kleinen Systemen Schnittstellen zu finden und die Entwicklungsteams berichten, dass die Schnittstellen von Beginn an das zentrale Thema der Architekturdiskussionen waren.

Außerdem ließ sich in den Fallstudien beobachten, dass Schichtenarchitekturen zwar auf der Ebene der Schichten zu zyklenfreien Strukturen führen, auf Klassen-Ebene aber kein vergleichbarer Effekt zu beobachten ist. In allen Fallstudien mit Schichtenarchitektur waren relativ viele Klassen in große, schwer aufzulösende Zyklen (Ø 25%) verflochten. Referenzarchitekturen hingegen erzeugen auf der Klassen-Ebene eine Struktur mit wenig Zyklen (Ø 3%). Andererseits wiesen die reinen Referenzarchitekturen auf der Ebene der Schichten und Subsysteme mehr Verletzungen auf. Vier der Systeme mit Referenzarchitektur hatten zusätzlich eine Schichtenarchitektur, so dass diese Systeme sowohl auf Klassen- als auch auf Schichten-Ebene eine hierarchische Struktur haben.

4.3 Entwicklung von Architekturen

Systeme ohne Architekturstil lassen sich mit großer Mühe nachträglich an eine Schichtenarchitektur anpassen. Bei zwei Fallstudien war dieser Umbau vorgenommen worden und hatte zu einer besseren Wartung geführt. Ein System im Nachhinein auf eine Referenzarchitektur umbauen zu wollen, kommt einer vollständigen Reimplementierung gleich und konnte daher bei keiner Fallstudie beobachtet werden. Referenzarchitekturen lassen sich über einen längeren Zeitraum nur erhalten, wenn die Wartung von Entwicklern vorgenommen wird, die sich der Regeln der Referenzarchitektur bewusst sind.

4.4 Umbau zu Serviceorientierten Architekturen

Eine SOA kann nur dann gelingen, wenn es möglich ist, einzelne Dienste aus einem vorhandenen Softwaresystem herauszuschneiden. Zwischen dem Dienst und der restlichen Software dürfen dabei keine Zyklen bestehen. Hier weisen Referenzarchitekturen eindeutig die besten Ergebnisse auf, so dass sie einen Umbau begünstigen. WAM und Quasar verfügen darüber hinaus über spezielle Komponenten, die in der Sprache der Referenzarchitektur als Services bezeichnet werden [HHV06, Zü05]. In zwei der Fallstudien hat es sich gezeigt, dass diese Services sich bei einer SOA direkt einsetzen lassen.

Dienste benötigen außerdem eine klar und gut definierte Schnittstelle. Dieses Ziel wird am stärksten von Subsystemarchitekturen unterstützt, die besonderen Wert auf die Schnittstellen von Subsystemen legen. Wird dieser Stil mit einer Referenzarchitektur kombiniert, so ist ein Ausbau zu einer SOA gut vorbereitet.

5 Fazit

Für die 28 Fallstudien lässt sich festhalten, dass der Einsatz eines Architekturstils von Beginn an zu wartbareren und erweiterbareren Systemen geführt hat und somit der Grundstein für einen Ausbau zu einer SOA gelegt wurde. Schichten- und Subsystemarchitekturen bewirken eine bessere Struktur auf Ebene der Schichten und Subsysteme; auf der Klassen-Ebene geben sie aber keine Anleitung und führen so zu komplexen Strukturen innerhalb der Subsysteme. Die Systeme mit der geringsten strukturellen Komplexität waren mit einer Kombination aus Referenzarchitektur und Schichten- oder Subsystemarchitektur entwickelt worden.

Literaturverzeichnis

[BCK03] Bass, L.; Clements, P.; Kazman, R.: Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley, Reading, Mass., 2003.

[BKL06] Becker-Pechau, P.; Karstens, B.; Lilienthal, C.: Automatisierte Softwareüberprüfung auf der Basis von Architekturregeln. In (B. Biel; M. Book; V. Gruhn, Hrsg.): Software Engineering 2006. Gesellschaft für Informatik, Leipzig, 2006; S. 27-38

[BR07] Broy, M.; Rumpe, B.: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik-Spektrum, Vol. 30, 2007; S. 3-18

[BD04] Brügge, B.; Detroit, A. H.: Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java. Pearson Studium, München, 2004.

[Di68] Dijkstra, E. W.: The Structure of the T.H.E. Multiprogramming System. Commun. ACM, Vol. 11, 1968; S. 341-346

[Ev04] Evans, E.: Domain Driven Design: Tackling Complexity in the heart of Software. Addison-Wesley, 2004.

[Fo03] Fowler, M.: Patterns of enterprise application architecture. Addison-Wesley, 2003.

[HHV06] Hess, A.; Humm, B.; Voß, M.: Regeln für serviceorientierte Architekturen hoher Qualität. Informatik-Spektrum, Vol. 29, 2006; S. 395-411

[HNS00] Hofmeister, C.; Nord, R.; Soni, D.: Applied Software Architecture. Object technology series, Addison-Wesley, 2000.

[KG06] Kim, J. S.; Garlan, D.: Analyzing architectural styles with alloy. Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis. ACM Press, Portland, Maine, 2006; S. 70-80

[Kru95] Kruchten, P.: The 4+1 View Model of Architecture. IEEE Software 12, Vol. 6, 1995; S. 42-50

[My78] Myers, G. J.: Composite/Structured Design. Van Nostrand Reinhold, New York, 1978.

[Pa74] Parnas, D. L.: On a 'Buzzword': Hierarchical Structure. IFIP Congress 74. North-Holland Publishing Company, 1974; S. 336-339

[Pa72] Parnas, D. L.: On the Criteria to be Used in Decomposing Systems into Modules. Commun. ACM, Vol. 15, 1972; S. 1053-1058

[RH06] Reussner, R.; Hasselbring, W.: Handbuch der Software-Architektur. Vol. 1. Aufl. dpunkt.verlag, Heidelberg, 2006; S.

[SG96] Shaw, M.; Garlan, D.: Software architecture: perspectives on an emerging discipline. Prentice-Hall, Inc., 1996.

[Si04] Siedersleben, J.: Moderne Softwarearchitektur: Umsichtig planen, robust bauen mit Quasar. dpunkt.verlag, Heidelberg, 2004.

[Zü05] Züllighoven, H.: Object-Oriented Construction Handbook. Morgan Kaufmann Publishers, San Francisco, 2005.

[ZR06] Züllighoven, H.; Raasch, J.: Softwaretechnik. In (Rechenberg/Pomberger, Hrsg.): Informatik-Handbuch, Vol. 4. Hanser Verlag, München, Wien, 2006; S. 837-879