Der kleine Software-Architekt

Teil 1

Zusammenfassung:
Wer als Entwickler in Software-Projekten arbeitet und sich fachlich weiterentwickeln will, strebt nicht selten die „hohen Weihen“ der Software-Architektur an. Dieser Artikel ist der Beginn einer kleinen Serie, in der dem Leser Orientierung zur Weiterentwicklung der eigenen Fähigkeiten an die Hand gegeben wird. Die Fragen, die hier besprochen werden, sind: Was ist Software-Architektur? Wozu dient sie? Welche Aufgaben hat der Software-Architekt? Und wie können sich Entwickler in diese Richtung bewegen?
Fortsetzungen werden Technologien, häufig anzutreffende Architekturthemen und methodisches Vorgehen zum Gegenstand haben.

Einführung

In Software-Projekten, deren Ziel es ist, Geschäftsanwendungen zu bauen, zu ändern oder zu integrieren, sind viele verschiedene Rollen zu besetzen und Funktionen mit Leben zu füllen. Da gibt es z.B. den *Projektleiter*, den *Tester*, die *System-Analytiker* oder *Fachkonzipierer*, den *Build-Manager*, den *Qualitätssicherungsverantwortlichen* usw. Die Funktionen und Rollen sowie ihre Bezeichnungen sind in der Praxis von Projekt zu Projekt zumindest leicht unterschiedlich. Konsistent begegnet ist mir aber die Trennung von *Architekten* und *Entwicklern*.
Kurz – und etwas überspitzt – formuliert ist es die Einteilung der Entwicklungskräfte in Leute, die etwas zu sagen haben und Leute, die danach zu handeln haben. Ein gutes Projektteam zeichnet sich natürlich dadurch aus, dass so eine scharfe Trennung nicht gelebt wird, aber ein kleines bißchen dieser Kultur der Zweiklassengesellschaft findet sich in jedem Projekt wieder.

Nun macht es natürlich Sinn, dass erfahrene Leute mit Weitblick die fachliche Führung besitzen und andere, mit weniger augeprägter Erfahrung, folgen. Denn Software-Entwicklung für Geschäftsanwendungen ist keine Sandkiste. Hier werden Millionen von Euro ausgegeben, um das Projektziel zu erreichen, und nochmal Millionen, um die Wartung des Systems über viele Jahre zu finanzieren. Falsche technische Entscheidungen können über so lange Zeiträume Schäden in großer Höhe verursachen, selbst wenn niemand das am Ende so genau nachrechnen kann. Die Entscheider (gleich, ob sie sich in einem Projekt Architekten oder Entwickler nennen) tragen also eine große Verantwortung.

Wer als Informatiker von der Uni oder FH kommt oder sich die nötigen Programmierkenntnisse als Quereinsteiger erarbeitet hat, der wird eigentlich immer als Entwickler in Software-Projekten starten und dort einige Jahre Erfahrungen sammeln. Von dieser Position aus wünschen sich manche Menschen eine fachliche Karriere, und dabei ist eine wichtige Etappe, als „Software-Architekt“ zu arbeiten.

Aber wie wird man Software-Architekt? Dazu gehören in meinen Augen drei Dinge:
Erstens: das theoretische Wissen und die praktische Erfahrung zu Entwurfsprinzipien, typischen technischen und fachlichen Problemen sowie Methoden, Konzepten, Technologien und Produkten, um diesen zu begegnen.
Zweitens: die kommunikationsstarke Persönlichkeit, so dass man durch Papiere, Vorträge und insbesondere direkte Diskussion die eigenen Lösungsansätze mit anderen austauschen und notfalls auch durchsetzen kann.
Drittens: das Glück, in der Projektpraxis die Gelegenheit zu kriegen, die Eigenschaften unter erstens und zweitens auszubilden.

Man kann seinem Glück natürlich nachhelfen, denn durch eigene Initiative kann man sich viel Wissen aneignen: vieles ist schon in Büchern niedergelegt und man erhält eigentlich zu jeder Technologie eine freie Implementierung, mit der man auf professionellem Niveau experimentieren kann.
Auch an der eigenen Persönlichkeit kann man arbeiten, indem man im Projekt offen um Verantwortung bittet, Feedback einfordert und sich außerhalb des Projekts seiner Stärken und Schwächen durch Kommunikations- oder Präsentationstrainings bewusst wird. Das Potential mal vorausgesetzt, ergibt sich dann früher oder später die Gelegenheit, als Architekt den eigenen Einfluß im Projekt zu erweitern.

Dieser Artikel ist der Beginn einer Serie, die einem Entwickler oder einer Entwicklerin mit drei oder mehr Jahren Berufserfahrung in größeren Java-Projekten inhaltliche Orientierung bieten soll, damit eine gesunde, fundierte Wissensbasis entsteht, die ihn/sie in die Lage versetzt, architekturrelevante Lösungsansätze zu finden, auszuarbeiten und in die Breite zu tragen.
Ein sehr großer Teil des Wissens ist längst dokumentiert. Ich werde nicht den müßigen Versuch unternehmen, all dies zu wiederholen. Vielmehr versuche ich, die Wissensquellen durch einen roten Faden zu verknüpfen und Anregungen zu geben, was zu lesen oder auszuprobieren ist.
In diesem Beitrag werde ich zum einen die Aufgabe der Software-Architektur und des Architekten beschreiben, denn daraus ergibt sich meine Herangehensweise an Architektur-Probleme. Zum anderen versuche ich, die erste Etappe zu zeichnen, die ein Entwickler auf dem Weg zum Architekten erreichen muss. Schließlich gebe ich noch einen Ausblick auf den nächsten Teil der Serie.

Wozu Software-Architektur?

An dieser Stelle müsste ich definieren, was Software-Architektur eigentlich ist, um dann festlegen zu können, wozu man sie braucht. Das haben schon viele vor mir getan: http://www.sei.cmu.edu/architecture/definitions.html.

Nehmen wir z.B. diese Definition: *"Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled."* von Eoin Woods, Autor des Buches "*Software Systems Architecture : Working With Stakeholders Using Viewpoints and Perspectives*".

Software-Projekte sollten kein Selbstzweck sein, sondern auf Basis eines Business Case gestartet werden, der eine Betrachtung zwischen zu erwartenden Kosten und dem erhofften Nutzen ist. Insofern geht es bei Software-Architektur darum, Lösungen zu finden und zu implementieren, die wirtschaftlich effizient sind. Dabei gibt es Tausende technische Einzelentscheidungen, die während der Entstehung einer Software zu treffen sind, doch nur eine kleine Menge davon kann wirklich maßgeblich Kosten beeinflussen.

Eine Software-Architektur legt explizit diese wichtigen Entscheidungen fest. Aber sie leistet noch mehr, wie die nächsten Absätze ausführen werden.

Eine Eigenschaft nicht-trivialer Software-Projekte ist, dass niemand zu ihrem Beginn genau sagen kann, was die Anforderungen sind. Das gilt für funktionale ebenso wie nicht-funktionale Anforderungen. Unglücklicherweise braucht es aber gerade am Anfang stabile technische Festlegungen, und dazu benötigt man eigentlich stabile Anforderungen. Diesem Mangel begegnet man auf zweierlei Weise: durch Weitsicht, die aus branchenspezifischer Erfahrung resultiert, und durch das bewusste „Offenhalten von Türen", was man durch die Anwendung von Entwurfsprinzipien erreicht.

Software-Architektur schafft darüberhinaus durch Begriffsbildung, anschauliche Darstellungen und Dokumentation ein gemeinsames Verständnis vom technischen Aufbau des Systems und der Abbildung der Fachlichkeit auf technische Elemente. Denn wenn zehn Entwickler und Architekten zueinanderkommen, dann treffen auch immer zehn unterschiedliche Sichtweisen und Erfahrungshintergründe aufeinander. Damit im Software-System in sich schlüssige Konzepte auch tatsächlich realisiert werden, ist es unerlässlich, eine gemeinsame Sicht zu schaffen, wie das System aussehen soll.

Um ein größeres Entwicklungsvorhaben steuern zu können, benötigt man im Projektmanagement Arbeitspakete, durch deren Abarbeitung definierte, nachprüfbare Ergebnisse entstehen. Software-Architektur hilft ganz entscheidend dabei, die Arbeitspakete der Realisierung zu schneiden. Sie liefert zum einen bei größeren Subsystemen durch das Paketdesign und Feature-Roadmaps klar beschreibare Teilziele. Und zum anderen gibt sie uns ein Schema, nach dem wir bereits anhand der Beschreibung der wahrnehmbaren fachlichen Funktionalität eine Zerlegung in technische Elemente bekommen. Aus diesen Elementen werden dann leicht schätzbare Arbeitspakete.

Software-Architektur wird durch die voranstehend skizzierten Leistungen eine tragende Säule in einem Projekt.
Soweit zu dem, was sie leisten soll. Aber welche Aufgaben auf dem Wege dahin sind von einem Software-Architekten auszufüllen?

Welche Aufgaben hat ein Software-Architekt?
Die Fülle an Dingen, die ein Architekt zu tun hat, ist groß. In einem Projekt von mehr als zehn Mitarbeitern Personalstärke sind alle anstehenden Aufgaben gewissenhaft von einer Person kaum leistbar. Daher verteilt sich die Arbeit in der Praxis häufig auf mehrere Personen, die als Architekturteam zusammenwirken. Ob dieses Team als eigene Organisationseinheit etabliert wird oder vielmehr virtuell aus Mitarbeitern der Entwicklungsteams zusammengesetzt ist, ist eine eigene Diskussion wert. Doch ganz gleich, wie die Arbeit organisiert wird, wichtig ist, dass alle Architekten den unmittelbaren Kontakt zu den realen Problemen der Entwicklung behalten, sonst werden die Ergebnisse der Software-Architektur schnell als Werk aus dem Elfenbeinturm wahr- und letztlich nicht ernstgenommen.

Die Rolle des Software-Architekten im Unternehmen und dem Projekt wird in [PBG 04], Kapitel 2, ausführlich erläutert, so dass ich mir hier viele Worte spare. Ich fasse die Aufgaben knapp wie folgt zusammen:

- Architekturrelevante Anforderungen sammeln und Einflussfaktoren bestimmen.
- Architekturthemen identifizieren.
- Lösungsentwürfe zu den Themen finden und ausarbeiten.
- Prototypen implementieren.
- Dokumentation und Kommunikation der Software-Architektur.
- Mitarbeiter anleiten und führen.
- Abstimmung mit Projektmitarbeitern, Stakeholdern und Vertretern anderer Projekte und Umsysteme.
- Realisierungsergebnisse auf Architektureinhaltung prüfen.

- Beratung und Unterstützung der Projektleitung.
- Untersuchungen zu Problemen der technischen Architektur (Performanz, Parallelität, Ausfallverhalten usw.)

Die Liste ist sicherlich nicht erschöpfend, aber wenn in der Praxis schon dies alles geleistet wird, ist viel gewonnen.

Vom Entwickler zum Architekten

Aber jetzt nochmal zurück zur Ausgangsfrage: Wie wird man nun ein Software-Architekt, wenn man bisher als Entwickler arbeitet?
Es gibt natürlich keinen Königsweg oder eine Musterlaufbahn, dafür sind die Wege in der Branche viel zu sehr von individuellen Fähigkeiten und zufälligen Umständen abhängig. Aber um Orientierung zu geben, werde ich mehrere „Etappen“ benennen. So kann jeder versuchen, sich selbst einzuschätzen, um dann die nächste Etappe ins Auge zu fassen.

Das Ziel der ersten Etappe ist, aus der Entwicklerposition heraus als Kandidat für die Übernahme von Architekturaufgaben wahrgenommen zu werden. Spätere Etappen sind dann: die überwiegende und dauerhafte Bearbeitung von Architekturaufgaben, die Übernahme der Verantwortung für ganze Systembereiche oder Themenkomplexe, und letztlich die Führung von Entwickler- oder Architekurteams.

Etappe 1
Ich werde im folgenden versuchen, die Schritte der ersten Etappe zu beschreiben, die ich in der Praxis beobachtet habe.

Als Entwickler beschäftigt man sich mit jeder Menge technischer und fachlicher Details. Das ist gleichzeitig ein Vorteil und ein Nachteil. Der Vorteil ist, dass man die „Wahrheit“ kennt, denn sie liegt bekanntlich im Code. Der Nachteil ist aber, dass größere Zusammenhänge davor in den Hintergrund treten. Wenn das Feature xyz zum Laufen zu bringen ist, dann zählt fast nur dieses Feature, denn das ist meistens schon schwer genug.

Der erste Schritt ist, festzustellen, dass das, was man da gerade programmiert, Beispiel- oder sogar Vorbildcharakter hat. Denn was wird ein anderer tun, der vielleicht Wochen später diese Codefragmente wegen einer Änderung oder Fehlerbehebung anfassen muss? Was wird jemand als Lehre mitnehmen, der sich gerade neu in die Codebasis des Projekts einarbeitet?

In beiden Fällen wird sich der oder die andere an dem orientieren, was da ist. Meistens gibt es ja auch keine andere Wahl, denn „mal eben neuschreiben“ ist nicht drin.

Architektur ist Führungsaufgabe, entweder durch direkte fachliche Mitarbeiterführung oder schlicht durch gute Argumente und Kompetenz. Was immer Du tust, Du hast häufig die Wahl zwischen einer Verbesserung, einer Verschlechterung oder dem Beibehalten des Status Quo. Du kannst diesen Führungsanspruch leben, indem Du Dich – wann immer möglich – für die Verbesserung entscheidest.

Das ist eine Veränderung der inneren Einstellung: es geht eben nicht nur um das Feature, es geht um ein insgesamt besseres System. Klar, das Feature muss laufen, aber ein Architekt versucht, auch eine Verbesserung zu schaffen. In [Fow 00] werden die Methoden des Refactoring beschrieben, die hiermit wärmstens zur Lektüre empfohlen seien.

Der zweite Schritt ist die moderate Abstraktion von den Details. Architektur hat sehr viel mit Kommunikation zu tun, und Kommunikation ist um so wirksamer, je einfacher die Botschaften formuliert sind. Eine große Detailfülle steht einer klaren Nachricht im Wege.
Die Kunst der Abstraktion ist es, die entscheidenden Fakten beizubehalten und Irrelevantes auszublenden.
Entwickler und Architekten praktizieren diese Kunst, indem sie Modellsprachen wie UML verwenden [Lar 02].

Sprachen muss man lesen und schreiben lernen. Du kannst das üben, indem Du Codestrukturen und -verhalten in geeigneten Diagrammen visualisierst. Du kannst wiederum Verantwortung demonstrieren, indem Du ungeschriebene Konzepte, die im System implementiert sind, dokumentierst, so dass neue Mitarbeiter sich schneller einarbeiten können

Durch diese Übungen bilden sich die Fähigkeiten, eigene Ideen wirksam zu verbreiten und später zu dokumentieren. Beim nächsten Schritt kann der Entwickler nun endlich voll von seiner technischen Detailerfahrung profitieren, denn ohne Technologiekenntnisse geht’s natürlich auch in der ersten Etappe nicht.

Fast jeder Entwickler hat schon mal mit der einen oder anderen technischen Infrastruktur zu tun gehabt und ist mir ihren Tücken vertraut. Das kann z.B. ein Web-MVC-Framework, ein Persistenzmechanismus, ein Swing-Client, eine XML-basierte Schnittstelle zu einem Umsystem o.ä. sein.
Der Einsatz der meisten Technologien bedarf eines Konzepts, wofür und in welcher Weise sie herangezogen werden. Ich kann z.B. problemlos eine Swing-Fenster-Klasse erstellen, die sich selbständig per JDBC mit Daten aus einer relationalen Datenbank versorgt. Ich vermenge auf diese Weise ein wenig Fachlichkeit mit viel Technologie (Swing und JDBC). Wer schon mal größere Mengen solchen Codes gesehen hat, wird mir zustimmen, dass das in der Wartung ein Alptraum ist. Also, wie strukturiere ich den Code, so dass ich möglichst die Fachlichkeit vom DB-Zugriff und von Swing trenne?

Technologien können selten „out-of-the-box“ eingesetzt werden. Erarbeite zu einer Dir vertrauten API oder einem Framework in Form eines Prototypen ein sauberes Design, das demonstriert, wie der Einsatz erfolgt. Dokumentiere dieses Design und diskutiere es mit Kollegen.

Bei dieser Gelegenheit: Was ist eigentlich ein „sauberes Design“?
Dazu scheint es einen wahren Wust von Meinungen, Regeln, Gesetzen und Glaubensrichtungen zu geben. Die beste praktisch orientierte Quelle ist meiner Meinung nach [Mar 03], wo mit unzähligen Codebeispielen gezeigt wird, was ein gutes Design ausmacht und wie man dorthin kommt. Der zentrale Inhalt ist auch im Web unter http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf zusammengefasst.

Wer sich diese drei Schritte zu erarbeiten versucht, wird mit großer Wahrscheinlichkeit bald positiv auffallen. Dann fehlt nur noch die günstige Gelegenheit, um diese Fähigkeiten im Ernstfall zur Anwendung zu bringen.

Ausblick
Im kommenden Teil werde ich die nächste Etappe der „Architekten-Karriere“ beschreiben, bei der es darum geht, dauerhaft technische Themen zu bearbeiten, weil man über das entsprechende Wissen verfügt.
Zusätzlich wird es eine kommentierte Liste von technischen Architekturthemen geben, die häufig für betriebliche Informationssysteme relevant sind. Anhand einer solchen Liste, kann sich der Leser gezielt nach Webressourcen und Literatur umschauen.

Literatur

[Fow 00] M.Fowler, „*Refactoring*“, Addison-Wesley, 2000.

[Lar 02] Craig Larmann, “*Applying UML and Patterns”*, Prentice-Hall, 2002.

[Mar 03] R.C.Martin, “*Agile Software Development*”, Prentice-Hall, 2003.

[PBG 04] Posch, Birken, Gerdom, „*Basiswissen Softwarearchitektur*“, dpunkt Verlag, 2004.

[Sie 04] J.Siedersleben, „*Moderne Softwarearchitektur*“, dpunkt Verlag, 2004.