

Knut Lünse, CTO bei der SAPHIR GmbH

Machen agile Software-Architekturen die Programmierung überflüssig?

INHALTSVERZEICHNIS

Executive Summary	3
Was sind eigentlich Software-Architekturen.....	4
Die Bausteine von Software-Architekturen	4
Die Entwicklung von Software-Architekturen	5
Terminologische Schwierigkeiten	5
Software-Architekturen als Kombination von Diensten	6
Auf dem Weg von der IT in die Fachabteilungen	6
Was macht eine Architektur agil?	7
Starre Software verhindern.....	8
Die Stärken von agilen Software-Architekturen	8
Agilitätsbremsen.....	9
Fazit	10
Autoreninformation	11

Executive Summary

Immer wieder fällt in den Diskussionen um Software-Systeme der Begriff „Architektur“. Dieses Whitepaper geht in kurzen Impulsen der Frage nach, was man unter einer Software-Architektur zu verstehen hat, wie man sie am besten auf- und umsetzt und welche Schwierigkeiten dabei zu überwinden sind. Zudem befasst es sich mit der Frage, was man unter einer **agilen** Software-Architektur zu verstehen hat, was die Voraussetzungen dafür sind und wie man sie erreicht.

Was sind eigentlich Software-Architekturen

Sehr gerne spricht man im Zusammenhang mit Software von Architekturen. Warum eigentlich? Natürlich, weil der Aufbau das entscheidende (Qualitäts-)Kriterium für die Funktionalität ist. Das trifft auf Gebäude genauso wie auf Software zu: Denn auf welchem Fundament ein Bauwerk steht, entscheidet maßgeblich darüber, wie man es verwenden kann. Oder andersherum: Der Zweck ist an bestimmte Voraussetzungen gebunden, die man schon bei der Planung berücksichtigen muss.

Jedes Stück Software muss im geschäftlichen Umfeld vor allem eines: das Geschäftsmodell unterstützen. Eben hier versagt Standardsoftware immer wieder, weil es den vielfach propagierten Standard nicht gibt, und wahrscheinlich auch nicht geben kann. Hier geht es um Regeln und Strukturen, die sich eine Organisation selbst gibt und die sie selbst verändert. Das betrifft sowohl die ablaufenden Geschäftsprozesse als auch die Zuständig- oder Verantwortlichkeiten innerhalb des Unternehmens, aber auch außerhalb. Eine Software-Architektur muss deshalb diese Regeln aufnehmen und abbilden.

Am Ende soll so ein Gebilde entstehen, bei dem die Beteiligten je nach ihrer Rolle in der Organisation immer wieder bestimmte Bausteine herausziehen und an einer anderen Stelle gemäß vordefinierter Regeln wieder neu einsetzen können.

Dazu benötigt man vor allem funktionierende Patterns, die ein Unternehmen angemessen beschreiben können und dann vor allem dafür sorgen, dass die Fachabteilungen die von ihnen benötigten IT-Dienste selbst konfigurieren können. Der Fokus muss also weg von der Implementierung hin zur Modellierung wandern. Eben deshalb rücken auch die Architekturen mehr und mehr ins Blickfeld, denn nur wenn sie entsprechend ausgerichtet sind, kann man einen Paradigmenwechsel auch vollziehen.

Die Bausteine von Software-Architekturen

Wenn man von Software-Architekturen spricht, geht es vorrangig um Funktionalität. Was soll eine Software leisten und wie kann sie das erreichen? Welche IT-Ressourcen sind dafür notwendig und wie müssen diese zusammenwirken? Solche und ähnliche Fragen verbergen sich in erster Linie hinter dem Begriff Architektur.

Agilität heißt dabei auch in diesem Zusammenhang, dass man zusätzliche Spielräume schaffen will. Zum Beispiel, indem sich die Fachabteilungen wie aus einem Shop bedienen und die von ihnen benötigten Applikationen selbst erstellen können. Der Zugriff auf diesen Bausatz ist dabei verhältnismäßig einfach über Benutzerportale zu gewährleisten, an denen sich die Benutzer mit ihrer Rolle anmelden. Diese Portale können und sollen dabei durchaus beides sein: Sowohl eine Umgebung zum produktiven Arbeiten als auch ein Bausatz, mit dem man sich seine Anwendungen „zusammensteckt“. Das scheitert bislang vor allen Dingen daran, dass die Fachabteilungen in der IT-Begriffswelt vollkommen fremd sind. Deshalb muss man die IT- und Service-Begriffswelt an die Fachabteilungen heranrücken.

Bewerkstelligen lässt sich das etwa über sogenannten Ontologien, die quasi an der Oberfläche die Begrifflichkeiten der Fachabteilungen aufweisen und dahinter die Techniken der IT verwenden. Die Architekturen dürfen jedoch nicht starr werden: Sie müssen jederzeit änder- und erweiterbar bleiben.

Die Entwicklung von Software-Architekturen

Die Frage, wie eine Software-Architektur aufgebaut ist, lehnt sich damit natürlich auch eng an die Entwicklung und die dabei herangezogenen Hilfsmittel an. Sollen eine oder mehrere ablauffähige Programme miteinander kommunizieren? Laufen sie vielleicht auf verschiedenen Hardwareplattformen? Und kommunizieren sie über ein Bussystem oder geht man über das Internet? Wenn ja, welche Protokolle kommen dann zum Einsatz? Mit jedem Schritt ergeben sich neue Fragen.

Wer jedoch vor der Aufgabe steht, eine Architektur einführen zu wollen, die mit einem klaren Paradigmenwechsel und mehr Agilität verbunden ist, sollte das nicht im Großen beginnen: Wenn man – wie bereits dargelegt – berücksichtigt, dass man die Architektur jederzeit ändern und erweitern können muss, ist das auch überhaupt nicht notwendig. Denn dann kann die Revolution im Kleinen anfangen und sich dann sukzessive wie konzentrische Kreise ausbreiten.

Eben hier scheitern allerdings viele Metadatenplattformen, weil sie keine Möglichkeit dazu bieten, die Metadaten zu ändern. Genau das ist jedoch eine Grundvoraussetzung, die beispielsweise die Object Process Methodology konsequent umsetzt, indem die Metadaten mit derselben Methodik behandelt werden wie die Master- und Produktionsdaten.

Die Upgrades für die Applikationen werden dabei letztlich nicht notwendig als neues Release eingeführt, sondern ergeben sich auch aus Veränderungen aus den eigenen Repositories heraus.

Terminologische Schwierigkeiten

Vollkommen unproblematisch ist die Vokabel "Architektur" im Zusammenhang mit Software jedoch nicht. Nur allzu leicht entsteht dadurch nämlich der Eindruck, man könne so wie Architekten beim Hausbau vorgehen. Und eben dieser Ansatz führt bei Softwareprojekten meistens zum Scheitern. Denn Architekten planen ihr Projekt in aller Regel von A bis Z und gehen dann an die Umsetzung. So kann man in Softwareprojekten aber nicht verfahren, denn das Umfeld ist dafür schlicht zu schnelllebig geworden – und zwar sowohl das geschäftliche als auch das technologische. Deshalb verlaufen Softwareprojekte heute zumeist iterativ-inkrementell. Denn sonst läuft man Gefahr, zu lange nach einem obsoleten

oder sogar falschen Plan zu verfahren und stellt dann am Ende fest, dass die eigentliche Idee doch nicht funktioniert.

Iterativ-inkrementell heißt dabei:

1. Die Reichweite der Architektur klären: Geht es nur um einen isolierten funktionalen Bereich oder um größere Bereiche.
2. Definieren, ob die Architektur eher daten- oder prozessdominiert sein soll.
3. Die geschäftlichen Rollen identifizieren, die sich mit einem Prozess assoziieren lassen.
4. Optional: Einen User-Interface-getriebenen Prototypen einer Applikation erzeugen, um das Design der Anwendung bei der Zielgruppe zu testen.
5. Die Schritte 1-4 wiederholen, bis man vollkommene Einigung erzielt hat.

Software-Architekturen als Kombination von Diensten

Software-Architekturen müssen aber noch einem weiteren wesentlichen Sachverhalt Rechnung tragen, denn es geht immer mehr auch darum, externe Funktionalitäten mit einzubinden. Die kleine festverdrahtete Anwendung wird seltener und macht lose gekoppelten Diensten Platz. Für Entwickler und Software-Architekten kommt es angesichts dessen immer mehr darauf an, sich jeden Vorgang als Kombination von Services zu denken, egal wie banal er auch klingen mag. Im Kern geht es dabei vor allem um die Informationsverwaltung. Services laufen als aufeinanderfolgende Prozesse ab und erzeugen dabei Daten. Ein Prozess muss sich dabei aber auch graphisch darstellen lassen und es muss deutlich werden, was der Vorgang selbst ist und auf welche Master-, Produktionsdaten er zurückgreift. Aber es geht nicht nur um formatierte Daten, sondern auch um unformatierte Daten z.B. auf Dokumentenbasis wie Anfragen oder Angebote und andere Informationen. Am Ende entsteht so ein Geschäftsartefakt, das seinerseits wieder als eine Vorlage für andere Dienste und Geschäftsvorgänge fungieren kann.

Auf dem Weg von der IT in die Fachabteilungen

Die Software-Architektur hängt im Wesentlichen von den geschäftlichen Anforderungen an die Softwaresysteme ab. Generell sollen Architekturkonzepte helfen, das Vorgehen bei der Implementierung auf eine höhere Abstraktionsebene zu bringen. Weg von den Begriffen der IT-Technologie hin zu den gelebten Geschäftsmodellen der Fachabteilungen. Diese sollen die Architektur nach ihren Anforderungen modellieren können. So betrachtet wandert also einiges an Kompetenz weg von der IT in Richtung der Sachbearbeiter-Ebene. Eine Tatsache, die man nicht in allen IT-Ressorts gerne sieht, weil sich damit natürlich auch eine Verschiebung von Kompetenzen abzeichnet. Zudem wird damit alles deutlich transparenter. Und das führt zu einer besseren Kontrollierbarkeit der Entwicklungs-Budgets.

Damit wird die IT aber natürlich nicht überflüssig, sondern sie verändert sich: Zwar soll die IT nicht mehr die Geschäftsvorfälle und Abläufe komplett implementieren, jedoch muss sie die Plattform und vor allen Dingen die dahinter laufenden Services bereitstellen. Und zwar so, dass die Fachabteilungen daraus eine Anwendung zusammenstellen können. Im Endeffekt arbeiten also beide Seiten – IT und Fachabteilung – an der gleichen Plattform und erweitern sie kontinuierlich. Die Grundlage dazu muss jedoch aus der IT herausgelegt werden: Denn in den Fachabteilungen kann man kaum entscheiden, ob bei einem zusammengesetzten Service zwei Bedingungen mit ODER verknüpft sind oder eben nicht. Dort setzt man vielleicht einfach Alternativen in einem Angebot.

Die Anwender wollen natürlich nicht wissen, wie das funktioniert, sondern sie wollen die Daten „auf den Tisch bekommen“. Genauso wie der Gast nach seiner Pizzabestellung nicht fragt, wie die Pizza gemacht wird, sondern sie eben nur vor sich stehen haben will. Der Anwender muss letztlich jedoch immer wissen, welche Dienste zur Verfügung stehen, um die Funktionen richtig aufzurufen. Genau wie der Gast in der Pizzeria aus der Karte wählen kann.

Was macht eine Architektur agil?

Software-Architekturen müssen heute mehr denn je flexibel oder agil sein. Das heißt, sie müssen das Geschäft des Anwenders unterstützen und dabei außerdem noch in der Lage sein, sich rasch an dessen Veränderungen anzupassen. Forderungen wie diese sind nicht neu. Sie führten in der jüngeren Vergangenheit dazu, dass der SOA-Gedanke (serviceorientierte Architekturen) einen immensen Aufschwung erfuhr. Dieser Aufschwung geriet jedoch ins Stocken. Denn statt einem Managementkonzept vermutete man dahinter vielfach eine „neue“ Technologie. Eine Fehleinschätzung, die unter anderem dazu führte, dass viele SOA-Projekte mit irgendwelchen technischen Baukästen angegangen und dann „abgebogen“ wurden, weil sie die Zielsetzung eben doch nicht erfüllen konnten. Der SOA-Kerngedanke hat jedoch keineswegs an Richtigkeit verloren. Vor allem wenn man agile Software-Architekturen aufbauen will. Denn die benötigen in erster Linie eines: flexible Dienste, die sich einfach koppeln und entkoppeln lassen.

Die Agilität entsteht bei einer solchen Architektur dabei ausschließlich über die Serviceorientierung. Denn man greift immer wieder auf bereits bestehende Dienste zurück. Natürlich muss man vermehrt auch externe Dienste einbinden können. Die Aufgabe der IT ist dabei die des Vermittlers. Eine serviceorientierte Architektur braucht nicht auf einem SOA-Bus zu laufen, sondern sie kann alle Arten von Kommunikationswegen adaptieren, z.B. Webservices auf der Basis von Internetprotokollen benutzen.

Starre Software verhindern

Software darf letztendlich aber nicht zu starr werden. Das beginnt bei der Wahl der Methodiken, die man während der Entwicklung einsetzt. Lange Zeit spielten bei der Software-Entwicklung zum Beispiel Klassen und deren Datenstrukturen, die sehr eng miteinander verzahnt waren, eine zentrale Rolle. Die Funktionalität war dabei ganz eng an den Datenstrukturen aufgehängt. Die Unified Modeling Language (UML) war das große Zauberwort. Die Auswirkungen der UML beschränken sich jedoch nur auf die Ebene der traditionell statischen objektorientierten Programmierung. Doch genau diese Herangehensweise macht Software über kurz oder lang unflexibel und insbesondere schwer änderbar. Denn durch starre Ableitungshierarchien können Änderungen gravierende Auswirkungen auf abgeleitete Klassen haben. Wenn es aber so ist, dass aufwändige Anpassungen durchgeführt werden müssen, kostet das Implementieren und Verifizieren der Software dann deutlich höheren Aufwand. Aber schlimmer noch: Diese oft nur aus der Implementierungstechnik heraus nötigen Softwarebearbeitungen führen die Applikationen aus der Synchronisation mit dem eigentlich abzubildenden Modell heraus. Damit steht sie dann abseits der eigentlich zu unterstützenden Geschäftsfunktionen. Das kann aber natürlich eben nicht das Ziel bei der Applikationsentwicklung sein. Stattdessen ist eine vollkommen andere Art zu denken erforderlich. Ziel muss es dabei sein, ein dynamisches Organisations-beziehungsweise Funktionsmodell und dessen Transformation ganzheitlich in ein IT-System zu konzipieren und implementieren. Grundlage dafür ist das dynamische Objektmodell. Entscheidend dabei ist vor allen Dingen, der mit dem Modell verknüpfte Ansatz, Ablaufaspekte des Systems über die Zeit zu betrachten. Denn alle Beteiligten der systemischen Organisations- und Funktionseinheiten verändern sich im Laufe der Zeit – abhängig von Ereignissen der Umgebung.

Die Stärken von agilen Software-Architekturen

Grundsätzlich kommt es darauf an, autonome Funktionseinheiten so früh wie möglich zu identifizieren und dementsprechend als autonome Komponente mit ihren statischen und dynamischen Beziehungen zu anderen Komponenten in das Systemmodell zu implementieren. Dies beschleunigt die Entwicklung und führt zudem zu deutlich robusteren Ergebnissen, weil sich – wie oben bereits angeklungen – System-Änderungen nur lokal auswirken. Zudem fällt der Aufwand für Implementierung und Test deutlich geringer aus. Darüber hinaus erlaubt es dieser Ansatz, etwaige Änderungen einfacher vornehmen zu können. Schließlich entwickelt man dann innerhalb der jeweiligen Projektgruppe nicht mit den gleichen Abhängigkeiten wie bei herkömmlichen Verfahren.

Prinzipiell muss es auch ein Ziel für jedes Unternehmen sein, eine IT-Architektur zu schaffen, die seine Informationen, Prozesse und Geschäftsregeln als dynamische Komponenten abbildet und als Dienste für andere Komponenten verwendbar macht. Richtig sinnvoll wird diese Architektur jedoch nur dann, wenn sie sich auch unkompliziert anpassen lässt. Dies

gelingt nur, wenn die Implementierungssprache möglichst nahe am semantisch-fachlichen Modell orientiert ist. Denn nur allzu oft liegt das Hauptaugenmerk auf der IT-Technik, also auf syntaktischen Aspekten. Diesen Graben gilt es zu schließen.

Die sonst in der Engineering-Phase erforderlichen Modelltransformationen über mehrere Ebenen hinweg, vom fachlich-semantischen Modell bis zur untersten Ebene des Implementierungsmodells, sind überflüssig. Ebenso bedarf es keiner plattformspezifischen Implementierung mehr, die sonst das Gros der Implementierungsphase in Beschlag nahm. Einer der Hauptvorteile von agilen Architekturen besteht jedoch in ihrer Ausdehnbarkeit: Man kann mit einem beschränkten Ausschnitt starten und sie dann auf weitere dynamische Komponenten ausdehnen.

Agilitätsbremsen

Permanenter Preisdruck bestimmt natürlich auch in der Software-Entwicklung das Geschäft. Doch wer nur auf den vermeintlich billigeren Preis schaut, handelt sich damit sehr oft schwerwiegende Probleme ein: Das gilt insbesondere auch, wenn es um die Agilität von Software-Architekturen geht. Denn hier benötigt man Top-Kompetenz, die man einfach nicht für Low-Cost einkaufen kann.

Offshore-Entwicklung mag sich beispielsweise durchaus für einfache Funktionalitäten in einem Massenmarkt eignen. Wenn es aber darum geht, Geschäftsmodelle so abzubilden, dass das Anwenderunternehmen nicht nur sauber dargestellt ist, sondern auch noch die Möglichkeit hat, seine Anwendungen (mit nur wenigen Handgriffen) selbst zu erstellen, erfordert dies sowohl ein tiefgreifendes Verständnis der Technologie als auch des Unternehmens, also eher die Kompetenzen eines Business Analysten – und das lässt sich nun einmal nicht einfach so aus dem Ärmel schütteln. Die gute Nachricht dabei ist, dass ein solches Invest sich langfristig auszahlt, weil es ein Unternehmen in die Lage versetzt, sich schnell und ohne großen Aufwand an veränderte Rahmenbedingungen anzupassen.

Fazit

Eine Software-Architektur muss vor allen Dingen das Geschäftsmodell einer Organisation abbilden. Denn nach wie vor gilt die Regel: „IT follows business“. Soll das Ganze von Bestand sein, benötigt man Agilität. Diese Agilität kann jedoch nur dann entstehen, wenn die Fachabteilungen ihre Anwendungen aus bestehenden Bausteinen selbst zusammensetzen und an ihre Bedürfnisse anpassen können. Jahrelang verbrachte man auf der Suche nach der passenden Applikation seine Zeit damit, nach Standards zu suchen sowie aufwändige Lasten- und Pflichtenhefte zu verfassen. Das war genauso zeitraubend wie letztlich nutzlos. Jedenfalls dann, wenn es um Agilität geht. Und so stand dann am Ende eher: „Business follows IT“.

Damit Agilität gelingen kann, benötigt man eine Möglichkeit der Implementierung der Geschäftsvorgänge mit Serviceorientierung. Die Anwendungserstellung wandert dabei in die Fachabteilung, während sich die IT-Ressorts künftig vorwiegend um die Bereitstellung von Services, Schnittstellen und ein entsprechendes Metadatenmanagement kümmern müssen. Programmierung und Implementierung treten zunehmend in den Hintergrund, gleichzeitig tritt die Modellierung immer weiter in den Vordergrund. Applikationen werden künftig eher modelliert als programmiert.

Autoreninformation

Knut Lünse ist CTO und Gesellschafter der SAPHIR Gesellschaft für Software Systeme mbH. Mehr als 30 Jahre Berufserfahrung machen den diplomierten Mathematiker zu einem kompetenten Ansprechpartner für die Leitung von Software-Engineering-Projekten.

Zu den Arbeitsschwerpunkten von Knut Lünse zählen:



- **Service Oriented Architecture** (Veröffentlichungen)
- **Model Driven Architecture** (Veröffentlichungen)
- IT-Governance, ITIL, eTOM, TAM (TMF)
- TMN-Informationsmodell, CMIP
- (Tele-)Kommunikationsarchitekturen (SDH, STM, ATM, GSM, UMTS, Q.900, T.500; TCP/IP)
- IP-Architekturen (Routing)
- Komplexe Datenhaltung und Datengewinnung
- SW-Engineering (Agile SW-Entwicklung, Veröffentlichungen)

Über SAPHIR

Die SAPHIR Gesellschaft für Software Systeme mbH ist ein Systemhaus mit den Sparten Software Engineering, Netzwerke und Kommunikation sowie Services und Schulungen. Als unabhängiger Partner bietet SAPHIR seinen Kunden durchgängige Systemlösungen für technische und kommerzielle Anwendungen. Zum Leistungsspektrum von SAPHIR gehören neben dem Realisieren von Softwaresystemen und -komponenten auch das Erstellen von Analysen, Studien und Spezifikationen sowie die Systembetreuung. Darüber hinaus übernimmt SAPHIR für seine Kunden Projektmanagement und Mitarbeiterschulung.

1995 gegründet unterhält SAPHIR neben dem Stammsitz in München eine weitere Niederlassung in Friedrichshafen. Zu den Kunden des Systemhauses zählen unter anderem Siemens, ICS, BMW, Deutsche Telekom sowie Rohde & Schwarz.

Weitere Informationen unter www.saphirgmbh.de.

SAPHIR GmbH München
Herzogspitalstr. 11
80331 München

Email: saphir.muc@saphirgmbh.de

Tel: +49 89 2601 98 16
Fax: +49 89 2601 98 27

SAPHIR GmbH Friedrichshafen
Friedrichstr. 37
88045 Friedrichshafen

Email: saphir.frh@saphirgmbh.de

Tel: +49 7541 26 248
Fax: +49 7541 21 262