System-Architektur und Software-Engineering

Andres Koch
Object Engineering GmbH
Birmensdorferstrasse 32
CH-8142 Uitikon Waldegg
Tel +41 (0) 44 400 47 00
Fax +41 (0) 44 400 47 07
email: info@objeng.ch

Erschienen in "Offene Systeme" (1996) Band 5/ Nr. 1, Februar 96, S. 30-34
Springer Verlag International Heidelberg

Das Entwickeln von neuen Systemen auf der ``grünen Wiese'' gehört heute eher zur Minderheit der Entwicklungs-Projekte. Viel realistischer erweist sich der Miteinbezug von bestehenden Systemen zur Problemstellung des heutigen Software- oder System-Ingenieurs. Damit grössere und komplexere Systeme speziell in der heute trendmässig verteilten System-Landschaft erfolgreich zu Ende gebracht werden können braucht es `Engineering' im wahrsten Sinne des Wortes, und darin ist die Architektur die wegweisende Komponente.

Einleitung

Der Ausdruck ``Engineering'' (Herkunft aus dem Lateinischen ingenium in etwa genial) ist erst angebracht, wenn es eben geniale Ideen braucht, um Altes mit Neuem optimal zu kombinieren. Die amerikanische, zweite Definition von Engineering lautet ``the planning, designing, construction, or management of machinery, roads, bridges, buildings, waterways etc.'' und zeigt auf, dass es dabei nicht nur um eine technische Disziplin geht. Betrachten wir erfolgreiche Migrations-Projekte, dann wurden diese in erster Linie durch geschicktes Führen und Organisieren und erst in zweiter Linie unter Miteinbezug von guten Techniken zum Erfolg gebracht.

Die Architektur wird durch den Architekten erstellt

Die Architektur regelt den konzeptionellen Zusammenhang zwischen den verschiedenen eigenständigen Komponenten eines Systems. Sie formt die logische und physikalische Struktur eines Systems mit allen strategischen und taktischen Entwurfsentscheidungen, welche während dem Entwicklungsprozess angewendet werden müssen ([1]).

Hier finden wir wiederum den engen Zusammenhang mit dem geschickten Führen und Organisieren eines Projektes und dem richtigen Einsatz der richtigen Komponenten für ein System.

Zum Vergleich im Bauwesen, wo Architektur doch schon seit Jahrhunderten eine Rolle spielt, findet man die personelle Aufteilung in Architekt und Bauführer sehen, was für Informatik-Projekte ebenso dem Architekten und Projektleiter gleichkommt. Es ist natürlich auch wichtig, die Architektur durchzusetzen, damit das entstehende Produkt der geistigen Illusion des Architekten entspricht. Wer könnte dies besser tun, als der Architekt selbst, was man im Bauwesen oft in Personunion von Bauführer und Architekt sieht. Dies zeigt uns, dass für ein Projekt nur ein Architekt angebracht ist, um der Volksweisheit ``viele Köche verderben den Brei'' Rechnung zu tragen.

Es ist entsprechend angebracht die erfahrenste, am Besten ausgebildete Person als ArchitektIn einzusetzen. Die Kontinuität ist besser gewährleistet, wenn eine Person die Erstellung der Architektur übernimmt. Ebenfalls

gilt, dass jede Architektur besser als keine Architektur ist. Und, eine Architektur ist nur so gut wie sie verständlich dokumentiert ist. Oder könnten sie sich den Bau eines Bürohochhauses vorstellen ohne einen Bauplan?

Die Architektur wird speziell bei objektorientierten Systemen am Anfang der Entwurfsphase erstellt. Dies macht insbesondere Sinn, da neben den logischen Aspekten auch die physikalischen Aspekte eine grosse Rolle spielen. Werden bestehende Systeme wiederverwendet und in eine Weiter- oder Neuentwicklung miteinbezogen, muss dies in der Architektur ebenfalls mitberücksichtigt werden.

Wie bereits angesprochen, kann aufgrund der Architektur auch organisiert werden. Im Bauwesen wird der Dachstock auch durch den Zimmermann und nicht durch den Elektromonteur gemacht. Das heisst, dass die Aufgaben aufgrund der in der Architektur definierten Subsysteme und Teilkomponenten verteilt werden können. So kann für einzelne Teile die Verantwortung an ein kleines Team oder sogar an eine Einzelperson abgegeben werden, welche sich um die Realisation unter Einhaltung der, durch die Architektur vorgegebenen Richtlinien kümmert. Dazu müssen natürlich vorgängig die Schnittstellen zwischen den Modulen definiert und festgelegt werden, deren Abstimmung noch einiges an Koordination und Teamarbeit erfordert. Dies ist überschaubar genauso wie die Realisation der einzelnen Subsysteme.

Dass bei grossen Systemen auch eine entsprechende grössere Anzahl von Subsystemen und Komponenten zu realisieren und zu überwachen sind, lässt sich ableiten. Gerade bei der objektorientierten Entwicklung hat man es u.U. mit Hunderten von Klassen zu tun, die man im Projektmanagement auf irgendeine Art im Griff bekommen sollte. Hier kann man sich gemäss dem Vorbild der Apparate- und Maschinen-Industrie dem Hilfsmittel ``Stückliste'' bedienen. Dadurch wird es auch einfacher den Fertigstellungsgrad festzustellen, in dem man dies pro Einzelkomponenten feststellt und aufkumuliert. Dadurch erhält der Projektleiter (Bauführer) auch besser einen Ueberblick, wie weit fortgeschritten der ``Bau'' ist.

Dokumentation der Architektur

Die Architektur ist gemäss obiger Aussage nur etwas wert, wenn davon eine sinnvolle, verständliche und zutreffende Dokumentation vorhanden ist.

Das Zielpublikum davon teilt sich in zwei Gruppen auf:

- Manager, Benutzer, Kunden, andere Projekte (Subsysteme), Betrieb u.a
- Projekt-Management und Entwickler des Zielsystems.

Diese beiden Zielgruppen haben natürlich einen unterschiedlichen Wissensstand und unterschiedliche Erwartungen. So kann die Architektur in eine

- informelle Architektur
- formelle Architektur

aufgeteilt werden.

Informelle Architektur

Diese Dokumentation dient zur Präsentation des Projektes und des Projektfortschritts. Was den Entwicklern oft schwer fällt, ist der ``Verkauf'' und die Präsentation ihrer oft genialen Arbeit. Dies gilt auch für den Architekten. Entsprechend müssen sie das Erreichte so präsentieren, damit der Rückhalt des geldgebenden Management oder des Kunden weiterhin gewährleistet wird. Für Einsteiger ins Projekt hilft das informelle und auf möglichst einfache, graphische Darstellungen basierte Dokument schnell das konzeptionelle Verstehen des

Systems zu erlangen. Eine formale Notation wird nur dann eingesetzt, wenn diese für den nicht technischen Betrachter einfach und ohne grosse Ausbildung verständlich ist. Die Namensgebung der einzelnen Komponenten sollte denen der Subsysteme der formalen Architektur entsprechen.

Formelle Architektur

Die formelle Architektur beschreibt den technischen Aufbau des Systems unter Verwendung einer eindeutigen formalen Notation. Besonders bei umfangreicheren Projekten muss das System in mehrere Komponenten (Subsysteme) unterteilt werden um:

- den Ueberblick zu behalten
- organisatorisch aufteilbare Einheiten zu erhalten
- die bessere Vergabe von Aufgaben an Teams zu unterstützen
- die momentane oder spätere Wiederverwendbarkeit anzustreben
- die Flexibilität des iterativen Entwicklungsprozesses sicherzustellen.
- klare Schnittstellen zu schaffen, was wiederum Flexibilität ergibt.

Eine formelle Methode und Notation ist nötig, weil mit einer informellen, oft zweidimensional dargestellten Architektur die Darstellung komplexer, oft mehrdimensional darzustellenden Systeme nicht gemacht werden kann.

Als formale Darstellung der Architektur können herkömmliche Notationen aber auch objektorientierte Notationen (wie z.B. Booch, vgl. [1]) verwendet werden. Unter Verwendung der Booch-Notation, kann wie folgt aussehen.

- logische Komponenten-Struktur -> Kategorie
- Prozessaufteilung -> Objektdiagramme
- Physikalische Modul-Struktur -> Modul-Diagramme
- Physikalische Systemstruktur -> Prozess-Modell

Von der formalen Architektur ausgehend, welche mittels einem CASE-Tool entsprechend der verwendeten Methode, aufgesetzt wird, kann dann der Entwurf und die Implementation fortgesetzt werden. Darum empfiehlt sich die Architektur formal aufzusetzen.

Da man sich beim Erstellen der Architektur intensiv mit den Schnittstellen befasst, soll als Notation für die Schnittstellen vorteilhaft die Schnittstellen-Sprache (IDL) der von OMG (Object Management Group) definierten Common Object Broker Architecture (CORBA) ([3]) verwendet werden. Dies kann auch gemacht werden, wenn in einer ersten Phase noch keine Verteilung von Objekten (Komponenten) geplant ist. Eine formale, Programmiersprachen-übergeordnete Spezifikations-Notation kann hier als Pseudosprache verwendet werden, welche dann die spätere Erweiterung in eine verteilte Architektur stark unterstützt.

Eigenschaften einer guten Architektur

Versucht man Qualitätskriterien für eine gute Architektur zu definieren so wird man auf folgende Attribute stossen ([4]):

- Einfachheit
- Funktionalität
- Erweiterbarkeit
- Kapsulierung

Diese stimmen übrigens auch für objektorientierte Systeme zu, das heisst es spielt keine Rolle für welche Technologie die Architektur erstellt wird.

Einfachheit der Architektur

Wenn die Architektur schon komplex (aus vielen zusammenhängenden Teilen) ist, kann man sich in etwa vorstellen wie komplex erst die Umsetzung im Entwurf und Programmierung ist. Einfachheit ist das häufigste übersehene Qualitätsmerkmal muss anderseits als das wohl wichtigstes angesehen werden.

Je einfacher die Architektur, um so einfacher kann sie realisiert, verstanden, wiederverwendet und gewartet werden. Je komplexer desto schwieriger und teurer wird die gleiche Unterfangen.

Funktionalität

Grundsätzlich ist das natürlich der üblichen Zweck, der durch eine Architektur verfolgt wird. Gerade in heutiger Zeit ist die Interoperabilität ein stark zu berücksichtigender Punkt. Der Informationsaustausch zwischen einzelnen Komponenten muss beim Definieren der Funktionalität und beim Unterteilen in einzelne Module gut in die Ueberlegungen einbezogen werden. Hier kommt uns natürlich der Client/Server-Ansatz entgegen, der uns eher eine offene Schnittstelle bietet. Die Funktionalität ihrerseits ist zwar fundamental und trotzdem sollte sie in einem angemessenen Verhältnis zum erwarteten Lebenszyklus (oder Lebenserwartung) des Systems sein. Das heisst dass Features einzubauen wenig Sinn macht, wenn diese eventuell einmal in Zukunft benötigt werden könnten. Die Wartung eines mit solchen ``Schönheiten'' vollgepackten Systems kann die Wartung und den Betrieb eines Systems sehr teuer werden lassen. Als Paradebeispiele davon, sollte man sich die heute angebotenen Benutzerschnittstellen betrachten. Jedes Windowsdetail muss, wenn auch nicht mehr unbedingt direkt programmiert, doch spezifiziert und im Programm als Event abgehandelt werden. Besonders hier gilt Einfachheit geht vor ``Flipperkasten''.

Als vertretbarer Ansatz sollte die Funktionalität umgesetzt werden, welche in einer Anforderungs-Analyse auch eindeutig ermittelt wurde.

Das was dann noch kommen könnte, muss durch das nächste Qualitätsmerkmal der Architektur abgefangen werden:

Erweiterbarkeit

Wenn das System flexibel aufgebaut ist, was bei einer einfachen Architektur angestrebt wird, dann wird es ebenfalls einfach sein später erst gebrauchte Funktionalitäten einzubauen. Dasselbe trifft auch für Schnittstellen. Eine Schnittstelle breiter (umfangreicher) zu machen, ist mit weniger Aufwand machbar, als wegen einer Fehlplanung später überflüssige Schnittstellenteile zu entfernen. Dann kann praktisch nicht mehr festgestellt werden, in welchen anderen Modulen diese Features schon verwendet wurden. Durch die objektorientierte Entwicklungsmethode wird die Ausbaubarkeit besonders berücksichtigt und unterstützt, was bei grossen Systemen besonders zu hoch einzuschätzende ist. Wir können ja nicht laufend neue Systeme bauen, sondern sollten bestehende und gut konzipierte Systeme erweitern können. Um bestehende Komponenten wiederverwenden oder weiterverwenden zu können, muss auf die Isolierbarkeit oder Kapsulierung von Systemen geachtet werden.

Kapsulierung

Die Isolierung von Funktionalität in Module ist der Grundstein, um überhaupt eine Architektur aufstellen zu können, muss doch eine gewisse Abstrahierung der aus der Analyse stammenden Anforderungen in Komponenten verlegt werden können. Auch hier bestimmen Schnittstellen das Bindeglied dazwischen, wie wir es durch die einfache Client/Server-Architektur heute bereits in der Realität antreffen.

Wenn bei der Mehrheit der heutigen Altlastsystemen (Legacy Systemen) die Benutzerführung von der Applikation abgetrennt worden ist, dann können sowohl neue wie alte Applikationen auf die bestehenden, oft robusteren ``Erbschaften'' zugreifen, deren Implementation durch Isolierung nicht durchscheinen. Dafür bietet OMG mit CORBA wiederum die zukünftige Basis, wo Komponenten (man nennt diese einfach Objekte) via den Object Request Broker (ORB) nach dem Vorbild der Computer-Bussysteme ins System eingeklinkt werden können. Ob eine solche Komponente nun in C++, C, PL/1, ADA, Fortran oder sogar COBOL geschrieben worden ist, spielt auf der Ebene der Kommunikation (Interoperabilität) keine Rolle. Der gemeinsame Nenner ist der ORB und die bereits oben erwähnte CORBA-IDL, in welcher die Schnittstellen spezifiziert werden.

Der richtige Schnitt bei der Modularisierung (Isolierung) kann bei Bertrand Meyer [2] abgeschaut werden. Module müssen folgenden Qualitätsmerkmalen gehorchen:

- Modulare Zerlegbarkeit
- grössere Komponenten müssen zerlegt werden können
- Modulare Kombinierbarkeit
- kleinere Komponenten können einfach zu grösseren kombiniert werden. Gute Beispiele bilden hier die Befehlsstruktur von UNIX mit der Kombinierbarkeit über die ``Pipe'' (|) aber auch das erfolgreichste Kinderspielzeug LEGO.
- Modulare Verständlichkeit

Wie oben bereits angesprochen eine für den Erfolg eines Produktes wichtige Eigenschaft, welche sich einerseits im Aufbau der Schnittstelle und der Semantik der abstrakten, internen Funktionalität ist. Wenn der Anwender (Entwickler) innere, komplexe eines Moduls verstehen muss, damit er es einsetzen kann, dann würde man nicht schlechter fahren, wenn man ihm kleinere Komponenten zur Verfügung stellen würde. Schnittstellen-Design wird in unseren heutigen objektorientierten und verteilten Systemen eine hochwichtige Disziplin. Sie entscheidet stark über die Wiederverwendung von Komponenten und auch hier gilt: In der Einfachheit liegt die Mächtigkeit.

Modulare Stetigkeit

wiederspiegelt die gegenseitige Abhängigkeit von einzelnen Modulen und der Architektur. Eine kleine Aenderung eines Moduls sollte die Architektur nicht beeinflussen und ebenfalls sollten Module in Systemen mit verschiedenen Architekturen ihren Einsatz finden. Damit ist wiederum die Wiederverwendung von Komponenten angesprochen. Die internen Eigenschaften eines Moduls darf nicht eine gesamte Sicht der eigentlichen Applikations-Architektur vorschreiben, wenn es in anderen Applikationen wieder verwendet werden soll. Auch hier spielt die Spezifikation der Schnittstelle eine massgebende Rolle. Wenn Argument-Namen einer Schnittstelle schon eine gewisse Applikation, für welche die dahinterliegende Komponente verwendet wird, verraten, dann liegt bereits ein Verständlichkeits-Handicap für die Wiederverwendung vor. Schnittstellen sind weiter durch folgende von Bertrand Meyer [2] und in einem früheren Artikel [5] erwähnten Eigenschaften geprägt:

- Wenige Schnittstellen: Jedes Modul sollte mit möglichst wenig anderen kommunizieren
- Schmale Schnittstelen: Wenn zwei Module überhaupt miteinander kommunizieren sollten sie so wenig Information wie möglich austauschen. Dies zeichnet sich dann auch in loser Kopplung aus.

Mit dem Geheimnisprinzip, welches sich sowohl bei Bertrand Meyers Modularisierungs-Richtlinien wie den davon abgeleiteten fünf Prinzipien vorkommt, schliesst sich der Kreis mit der Isolierbarkeit der Kapsulierung bei unserer Architektur wieder.

Zusammenfassung

Die Grundaussage ist, dass eine gute Architektur Zeit und Geld spart und eine bessere Funktionalität und Flexibilität dem Endbenutzer zur verfügung stellt.

Leider noch zu oft werden heute Systeme konzipiert, worin der Programmierung und Integration grössere Aufmerksamkeit beigemessen wird, als einer übergeordneten Architektur. Dabei werden oft die organisatorischen Vorteile der durch die Architektur bestimmten Komponenten und die Verwendbarkeit von bestehenden Komponenten unterschätzt.

Bewegen wir uns vermehrt in verteilten Systemen, wird eine Architektur unabdingbar, wenn man statt Chaos die Uebersicht behalten will. Als konzeptionelle Basis wird sich hier die `Common Object Request Broker Architecture' von OMG immer stärker positionieren, in der als Objekte'' gewisse ``Erbmassen'' von bestehenden Programmen agieren können. Dank der Interface Definition Language (IDL, OMG) werden übergeordnet zu den eigentlichen Implementationssprachen, die Schnittstellen spezifiziert.

Was es aber braucht ist ein guter System-Architekt dem der Ausdruck Engineering einverleibt ist.

Literaturverzeichnis

G. Booch, "Object Oriented Analysis and Design with Applications", The Benjamin/Cummings Publishing Company, Inc. New York, 2nd Ed, ISBN 0-80535340-2, 1994.

Meyer Bertrand, "Ojektorientiere Softwareentwicklung", Carl Hanser München, Prentice Hall, ISBN 3-446-15773-5, 1990.

T. Mowbray and R. Zahavi, "The Essential CORBA, System Integration Using Distributed Objects", John Wiley, New York 1995, ISBN 0-471-10611-9.

T. Mowbray, "Essentials Of Objectoriented Architecture", S.28-32, Object Magatzine, September 1995, SIGS Publications.

A. Koch, "Objektorientierte Entwicklung verteilter Applikationen", Offene Systeme (1995) Band 4, Nr. 2, Mai 1995, S. 70-75, Springer Verlag International Heidelberg.