

Catalysts

WWW – Warum Was Wie???

Dr. Christoph Steindl
Catalysts GmbH
steindl@catalysts.cc

Das Thema der 12. Wissensspritze sind unsere Lehren aus 10 Jahren Software-Entwicklung für das Internet.

Dr. Christoph Steindl hielt diesen Vortrag am 10.7.2009.

Die Aufnahme des Vortrags steht unter <http://wissensspritze.catalysts.cc> zur Verfügung.



Vor etwas mehr als 10 Jahren hab ich mein Informatik-Doktorat abgeschlossen. Natürlich habe ich damals geglaubt, dass ich mich in allen wichtigen Bereichen der Informatik sehr gut auskenne.

Dann habe ich 5 Jahre als IT-Architekt bei der IBM im Kundenprojektgeschäft gearbeitet. Da gab es viele Fettnäpfchen und etliche davon sind halt einfach nun einmal unvermeidbar. Das Gute an den Fettnäpfchen ist, dass man aus ihnen etwas lernen kann.

Im Jahr 2005 hab ich mich dann selbständig gemacht und Catalysts gegründet. In den letzten 5 Jahren war ich der technische Leiter bei einem großen Forschungsprojekt und bei einigen kleineren Kundenprojekten. Außerdem habe ich etliche Software-Entwicklungs-Teams beraten.

In den nächsten 20 Minuten möchte ich erzählen, welche technischen Lehren ich aus den letzten 10 Jahren ziehe. Manches würde ich heute wieder so machen wie damals. Vieles aber mache ich heute mit Überzeugung etwas anders als damals. Und darum geht's in der heutigen Wissensspritze und in den nächsten paar Wissensspritzen: Warum wir bei Catalysts was wie machen.



Ich finde die folgende Unterteilung von Projekten oft ganz hilfreich: die zwei Achsen sind mit „WAS“ und mit „WIE“ beschriftet.

Nach oben ist aufgetragen, wie klar bzw. wie unklar das WAS ist – kennt man die Anforderungen im Detail oder sind die Anforderungen noch völlig unklar.

Nach rechts ist aufgetragen, wie klar bzw. wie unklar das WIE ist – beherrscht man die Technologie im Detail oder kennt man sich überhaupt nicht aus.

Wenn man links unten ist kennt man die Anforderungen und beherrscht die Technologien – das wären dann zahme Projekte.

Wenn man rechts oben ist, kennt man die Anforderungen nur oberflächlich und hat eine steile Lernkurve bei den Technologien zu erwarten – das wären dann verhexte Projekte.

Wenn ich mir anschau, welche Projekte ich so miterlebt habe, dann waren die nie zahm.

Zur IBM sind die Kunden halt nicht wegen Pippifax-Projekten gekommen.

Somit waren damals die Projekte, die auf meinen Tisch gekommen sind, einmal sicher nicht links unten. Sie waren auch nicht immer ganz rechts oben, aber es gab immer große Fragezeichen.

Bei Catalysts habe ich in den letzten 4 Jahren hauptsächlich in Forschungsprojekten gearbeitet.

Bei Forschungsprojekten kennt man natürlich nie alle Anforderungen im Vorhinein. Und natürlich gibt's auch immer viele technologische Fragezeichen.

Die Lehren gehen jetzt also auf verhexte Kundenprojekte zurück, wo die Kunden die IBM beauftragten für hochqualitative und langlebige Lösungen ihrer Probleme.

Die Lehren gehen aber auch zurück auf Forschungsprojekte mit Teams von etwa 10 Personen.

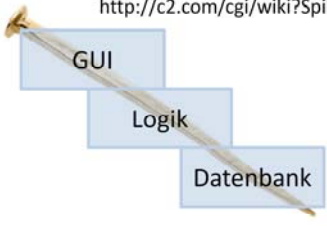
Außerdem haben wir als Unternehmensberater etliche Reviews von Software-Entwicklungsprojekten durchgeführt – das eine oder andere stammt auch von diesen Projekten, die wir begleitet haben.


Bei kleinen unkritischen Projekten ist vielleicht nicht alles wichtig, was ich heute so erzähle.


Bei etwas größeren Projekten, wo die Software dann einige Jahre in Betrieb sein soll und wo Qualität wichtig ist, sind mir die Lehren wichtig.

Spike ... Tracer Bullet ...


<http://c2.com/cgi/wiki?SpikeSolution>







<http://www.artima.com/intv/tracer.html>



© www.catalysts.cc, 2009
WWW - Warum Was Wie???
4

Oft steht man vor ein paar offenen Fragen und möchte sie schnell klären.

Bei Extreme Programming würden wir sagen, wir machen einen „Spike“.

Wir beginnen am User Interface am Client, schicken den Request zum Server, wo er verarbeitet wird, bis zur Datenbank geht und wieder retour.

Wir programmieren ein kleines Stück Funktionalität, das von oben bis unten bzw. von links bis rechts durch geht und wieder retour.

<http://c2.com/cgi/wiki?SpikeSolution>

"Spike" because [TopDown](#) is typically [BreadthFirst](#), but a Spike is [DepthFirst](#). From the top to the bottom, then [CloseTheLoop](#).

So-called because a spike is "end to end, but very thin", like driving a spike all the way through a log.

Oder wir schießen eine Tracer-Bullet ab, eine Leuchtmunition – wenn sich das Ziel schnell bewegt, trifft man leichter, wenn man bei jedem 10. Schuss sieht, wo er hingeht.

<http://www.artima.com/intv/tracer.html>

Dave Thomas: The idea of tracer bullets comes obviously from gunnery artillery. In the heavy artillery days, you would take your gun position, your target position, the wind, temperature, elevation, and other factors, and feed that into a firing table. You would get a solution that said to aim your gun at this angle and elevation, and fire. And you'd fire your gun and hope that your shell landed somewhere close to your target.

An alternative to that approach is to use tracer bullets. If your target is moving, or if you don't know all the factors, you use tracer bullets—little phosphorous rounds intermixed with real rounds in your gun. As you fire, you can actually see the tracer bullets. And where they are landing is where the actual bullets are landing. If you're not quite on target—because you can see if you're not on target—you can adjust your position.

Andy Hunt: Dynamically, in real time, under real conditions.

The software analog to firing heavy artillery by calculating everything up front is saying, "I'm going to specify everything up front, feed that to the coders, and hope what comes out the other end is close to my target." Instead, the tracer bullet analogy says, "Let's try and produce something really early on that we can actually give to the user to see how close we will be to the target. As time goes on, we can adjust our aim slightly by seeing where we are in relation to our user's target." You're looking at small iterations, skeleton code, which is non-functional, but enough of an application to show people how it's going to hang together.

Basically, it all comes down to feedback. The more quickly you can get feedback, the less change you need to get back on target.

„Wir wollen rasches Feedback“, heißt es da in Extreme Programming. Wir wollen Klarheit schaffen, wo Verwirrung ist.

Und weil das oft zu zweit leichter ist als alleine, machen wir Pair Programming.

Nur müssen wir verdammt aufpassen, dass das, was als Spike begonnen hat, nicht zu einer großen Mistkugel wird.

Weil Mistkugeln haben es so an sich, dass sie immer größer und größer werden, wenn man sie wälzt.

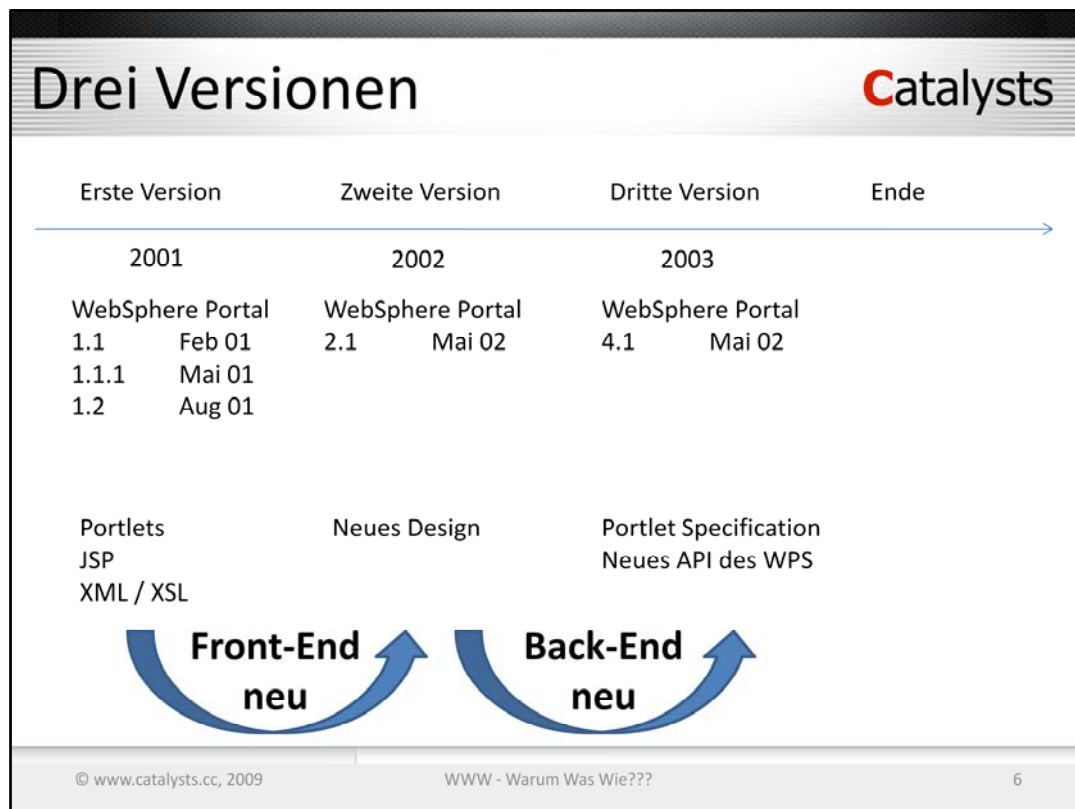
Und dann ist es vorbei mit der Wartbarkeit und Erweiterbarkeit.

Nicht jedes Problem kann man „ad hoc“ lösen.

© www.catalysts.cc, 2009 WWW - Warum Was Wie???

Im Jahr 2001 habe ich mit meinem Team ein Portalprojekt für eine Bank umgesetzt. Die technologische Basis sollte damals der WebSphere Portal Server werden – unser Projekt war eines der ersten 3 Projekte auf der ganzen Welt, die den WebSphere Portal Server produktiv eingesetzt haben. Das war damals technologisches Neuland – wir haben viel mit dem Labor in Böblingen Kontakt gehabt.

Was auf der Oberfläche schön aussieht, hat allerdings intern seine Tücken gehabt.

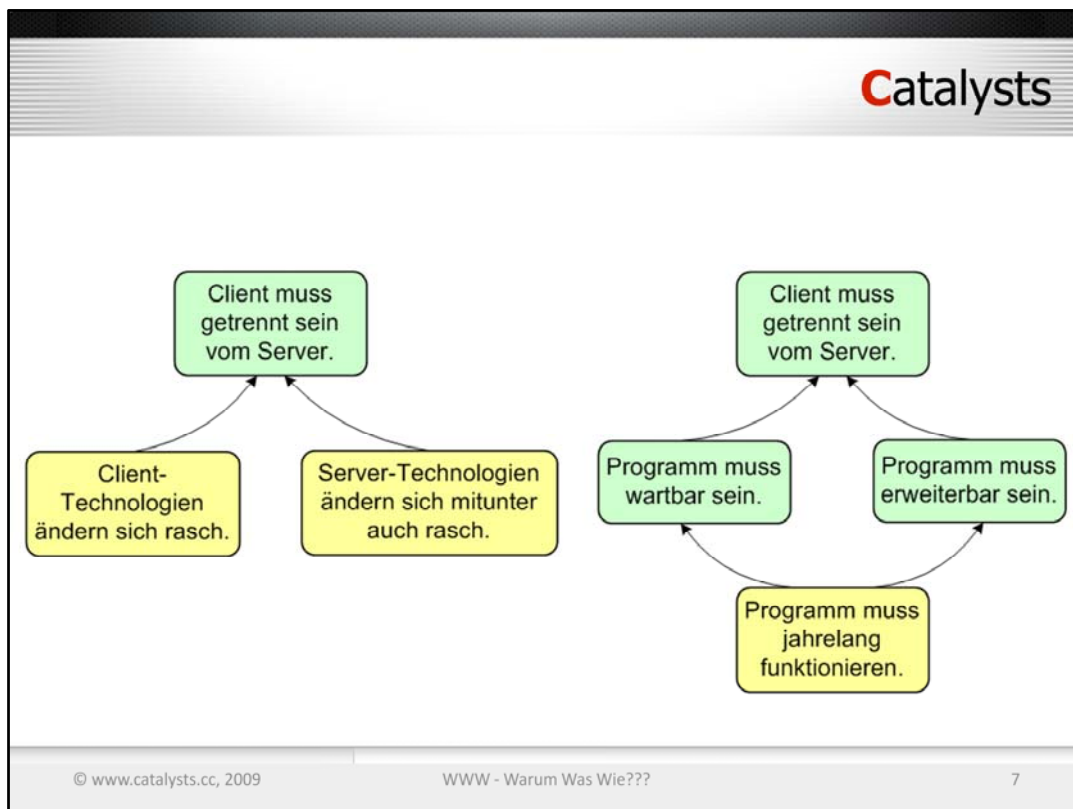


Im Jahr 2001 haben wir die erste Version des Kundenportals entwickelt. Wir haben mit Vorabversionen des Portalservers gearbeitet. Dort, wo der Portalserver noch funktionale Lücken hatte, haben wir Sachen dazu programmiert – in Abstimmung und mit Unterstützung vom Labor in Böblingen.

Ein Jahr später hatte die Bank ein neues Corporate Design, das heißt wir durften (oder mussten) das Front-End neu machen. Dazu mussten wir alle JSPs angreifen und alle XSLs. Und natürlich war die Visualisierung nicht immer komplett getrennt von der Programmlogik. Man hat schon sehr aufpassen müssen, dass man da durch grafische Änderungen nicht auch die Programmlogik ändert.

Wieder ein Jahr später war klar, dass die Bank auf die neue Version des Portalservers umsteigen musste. Das hat zwar aus Benutzersicht nichts geändert, aber intern war vieles anders. In der Zwischenzeit hatte sich die Portlet-Programmierung emanzipiert, d.h. es gab jetzt eine Portlet-Spezifikation. Viele Schnittstellen des WebSphere Portal Servers waren anders. In Summe hat das bedeutet, dass wir große Bereiche des Back-Ends neu schreiben mussten.

Und vielleicht wieder ein Jahr später wurde die Bank von einer größeren Bank übernommen. Durch die Neupositionierung war auch das Kundenportal obsolet und wurde stillgelegt.



Eine wichtige Lehre von diesem Portalprojekt war, dass uns eine bessere Trennung von Client und Server geholfen hätte.

Bei klassischen Client-Server-Programmen ist die Trennung eh ziemlich klar.
Bei Web-Anwendungen, wo die Anzeige im Server vorbereitet wird (mit JSPs, XSLs, Struts oder Ähnlichem), ist es schon wieder viel leichter, große Mistkugeln zu produzieren.


Wenn man einige Zeit „im Geschäft“ ist, weiß man, dass sich Client-Technologien rasch ändern.

Bei dem Portalserver-Projekt habe ich gelernt, dass sich auch Server-Technologien mitunter sehr rasch ändern können – speziell wenn ein Produkt neu auf den Markt kommt.

Man hat es somit leichter, wenn der Client und der Server gut voneinander getrennt sind.

Wenn ein Programm jahrelang funktionieren soll, muss es wartbar und erweiterbar sein.
Ein Programm ist umso leichter wartbar und erweiterbar je besser Client und Server voneinander getrennt sind.

1. Klare Trennung zwischen Client und Server



- Möglichst **schlanke Clients**
- Möglichst **viel Logik am Server** (für alle Clients gleich), dadurch Konsistenz über alle Clients hinweg
- **Clients in mehreren Programmiersprachen** möglich
- Mit **typisierter Schnittstelle** – sauberes API mit Methoden und Value-Objects
- **Effiziente Kommunikation** zwischen Client und Server (mit Delta-Übertragung)
- **Client-Updates** während laufendem Server möglich

© www.catalysts.cc, 2009WWW - Warum Was Wie???8

Für uns ist somit die 1. Lehre, dass es eine klare Trennung zwischen Client und Server geben soll.

Dann kann man den Client ändern, ohne dass der Server davon betroffen ist.
Und man kann den Server ändern, ohne dass der Client davon betroffen ist.

Man kann weitere Clients schreiben, die alle mit demselben Server kommunizieren.
Wenn man mehrere Clients hat, möchte man diese wahrscheinlich möglichst schlank halten.

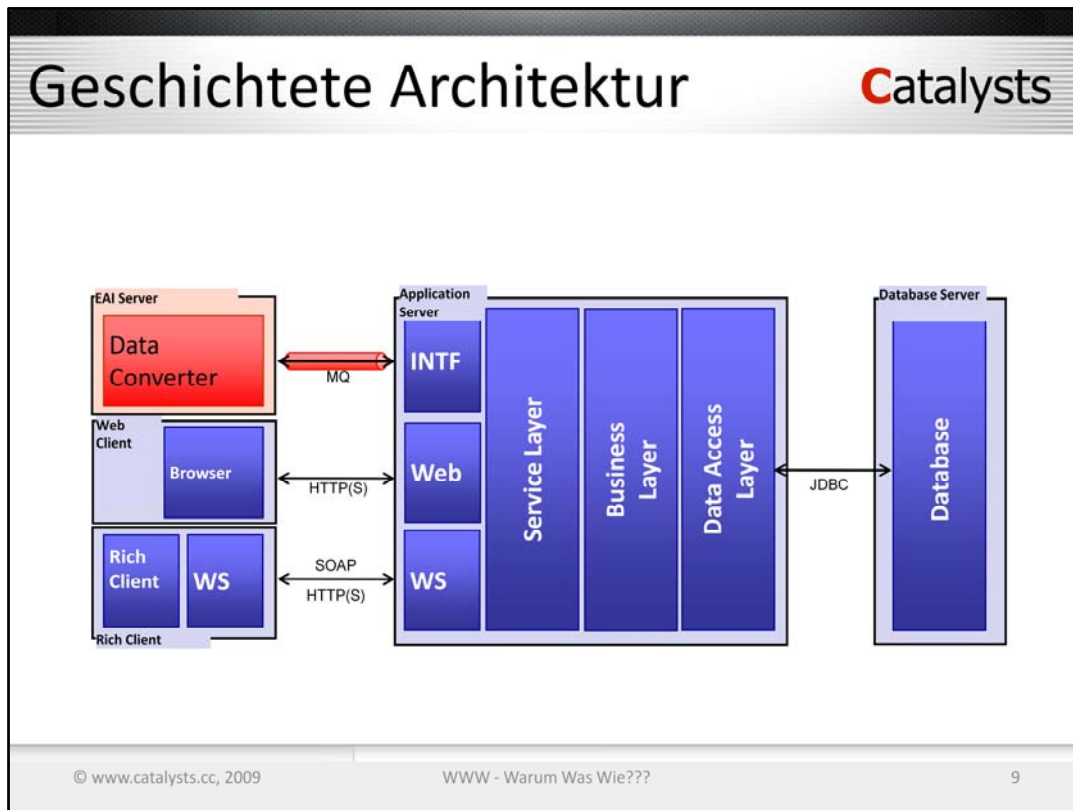
Mit möglichst viel Logik am Server, die für alle Clients verwendet wird, damit sich die Clients gleich verhalten.

Mit einer sauberen Trennung von Client und Server wird es dann auch möglich sein, die Clients in verschiedenen Programmiersprachen zu entwickeln – z.B. in Java, Flex/Action

Natürlich möchte man vom Client über eine typisierte Schnittstelle auf den Server zugreifen. Man möchte ein sauberes API mit Methoden und Value-Objects.
Auch wenn der Client und der Server nicht in derselben Programmiersprache geschrieben sind.

Dann sollte die Kommunikation zwischen dem Client und Server natürlich effizient erfolgen – immer nur die Änderungen.

Und wenn es einen neuen Client gibt, soll dadurch natürlich der Server nicht neu gestartet werden müssen.



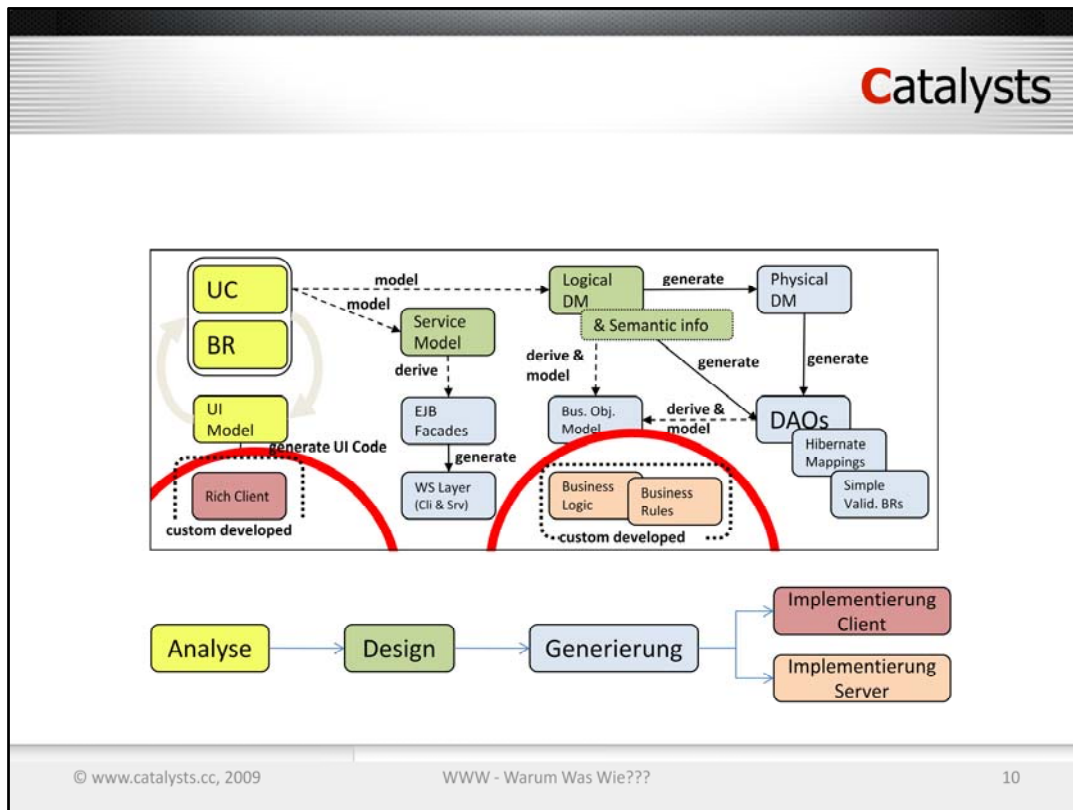
Wenn man jetzt einem Architekten freien Lauf lässt, dann entsteht sehr schnell eine schön geschichtete Architektur.

Mit einem Rich Client, einem Web-Client und einer Anbindung an beliebige Systeme über einen EAI-Server.

Mit einem Application-Server mit einer Service-Schicht, Business-Schicht und Datenzugriffsschicht.

Und mit einem eigenen Datenbank-Server.

Aber wer soll das dann noch alles im Detail verstehen und beherrschen?



Da muss man dann wieder hart arbeiten, um die Komplexität für die Entwickler zu reduzieren.

In einem großen Kundenprojekt habe ich kürzlich die folgende Situation vorgefunden:

Die Business-Analysten erarbeiten die Use Cases mit den Geschäftsregeln und dem User-Interface-Modell.

Die Designer machen daraus ein Service-Modell und ein logisches Datenmodell.


Daraus wird dann der Service-Layer, das Business-Objektmodell und die Persistenzschicht erzeugt.

Der Client-Entwickler muss sich nur mehr um die Programmierung der Darstellung im Rich Client kümmern – mit einem recht engen Korsett.

Der Server-Entwickler muss nur mehr die Geschäftslogik und Geschäftsregeln, die sich aus den Use-Cases ergeben, programmieren – auch wieder mit einem recht engen Korsett.

2. Saubere Software-Architektur Catalysts

- Am Client und am Server
 - Model-View-Controller am Client
 - Schichtung am Server
- Genaue Vorgaben an Entwickler
 - So einfach und verständlich wie möglich
 - So flexibel wie nötig
- Rollen-Trennung
 - Client-Entwickler brauchen kein Server-Wissen
 - Server-Entwickler brauchen kein Client-Wissen
 - Dadurch leichtere Skalierung des Teams
- Automatische Überprüfung der Architektur



© www.catalysts.cc, 2009 WWW - Warum Was Wie??? 11

Daraus ergibt sich die 2. Lehre: ab einer gewissen Größe brauchen Projekte eine saubere Software-Architektur.

Am Client braucht man eine Trennung von Model, View und Controller.

Am Server braucht man eine gute Schichtung – in Service-Schicht, Business-Schicht und Persistierungs-Schicht

Für die Entwickler braucht es genaue Vorgaben, welcher Code wo hin gehört.

Die Architektur muss so einfach und verständlich wie möglich sein – sie muss „adäquat“ sein.

Die Architektur muss aber auch so flexibel wie nötig sein.

Die Architektur sollte eine Rollen-Trennung unterstützen:

Client-Entwickler sollten kein detailliertes Verständnis über den Server brauchen.

Server-Entwickler sollten kein detailliertes Verständnis über den Client brauchen.


Wenn man das erreicht hat, kann man das ursprünglich kleine Kernteam zu einem größeren Team wachsen lassen.

Damit die Architektur-Vorgaben auch auf Dauer eingehalten werden, müssen sie automatisch überprüft werden.

In Summe führt eine saubere Software-Architektur zu Erlernbarkeit, Erweiterbarkeit und Wartbarkeit.

3. Testgetriebene Entwicklung Catalysts

- Dichtes Netz an Unit-Tests am Server
 - Kleine orthogonale Testfälle
 - In jeder Schicht der Server-Architektur
 - ohne Container, oft ohne Datenbank
- Unit-Integrationstests
 - über mehrere Komponenten / Schichten
 - vom Client bis zum Server
 - mit Container und Datenbank
- Clientseitige Testfälle
 - Zumindest für Model und Controller
 - Auch für Kommunikation mit Server
- Außerdem manuelle Testfälle (Akzeptanz)
- Außerdem Capture & Replay Tests am GUI



© www.catalysts.cc, 2009 WWW - Warum Was Wie???

Unsere 3. Lehre ist, dass testgetriebene Entwicklung sehr wichtig ist für die Qualität.

Wir wollen ein dichtes Netz an Unit-Tests am Server.

Dieses Netz besteht aus vielen kleinen orthogonalen Testfällen, die unabhängig voneinander sind und in einer beliebigen Reihenfolge laufen können.

Diese kleinen Unit-Tests gibt es in jeder Schicht der Server-Architektur.

Um die serverseitige Logik überhaupt gescheit unit-testen zu können braucht es eine gute Schichtung.

Man möchte ja z.B. für die Tests nicht immer wirklich in eine Datenbank schreiben oder wirklich E-Mails verschicken. Da muss man ganze Schichten zu Testzwecken austauschen können.

Diese kleinen Unit-Tests laufen sehr schnell, weil typischerweise außerhalb des Application Servers und oft ohne Datenbank.

Dann möchte man auch Unit-Integrationstests – das sind Tests, wo mehrere Komponenten zusammenspielen, oft über mehrere Schichten hinweg bzw. sogar vom Client bis zum Server.

Diese Tests dauern dann schon etwas länger, weil der Server-Code dann schon in einem Application-Server ausgeführt wird und auf eine wirkliche Datenbank geschrieben wird.

Dann können wir auch im Client noch viele Sachen testen – zumindest Model und Controller lassen sich gut automatisiert testen. Natürlich auch die Kommunikation mit dem Server.

Alle Tests bis hierher schreibt der Entwickler selbst.

Dazu kommen aber jetzt auch noch manuelle Testfälle – da definiert man Szenarien, wie der Benutzer mit dem System arbeitet, welche Daten er dabei eingibt und welches Ergebnis er sich erwartet.

Einen Teil dieser Testfälle wird man versuchen zu automatisieren – und da kommen dann oft Capture & Replay-Werkzeuge ins Spiel. Da bedient man das Programm und lässt die Tastatur- und Mauseingaben aufzeichnen, die dann erneut abgespielt werden.

Ein dichtes Netz an automatisierten Testfällen hilft einem zu garantieren, dass alles, was bereits einmal funktioniert hat, für immer funktionieren wird.

Und das lieben die Benutzer und Kunden.

4. Entwicklung im Team Catalysts

- **Zentrale Ablage** für Source-Code und Dokumente (Subversion)
- **Definierter Build** (Ant), optional mit lokalen Anpassungen (Mock für E-Mails; In-Memory-Database usw.)
- **Verwaltung der Modulabhängigkeiten** (Maven, Ivy)
- **Verschiedene Entwicklungsumgebungen** (Eclipse, IntelliJ IDEA, Visual Studio, FlexBuilder)
- Ständiges **Refactoring im Kleinen**, seltener **Architektur-Refactoring**
- **Automatischer und ständiger Build** (TeamCity, Hudson, CruiseControl)
 - Mit automatischen Regressionstests
 - Mit statischen Code-Analysen und Architektur-Analysen
 - Mit Benachrichtigung bei Fehler
- **Mit mehreren Umgebungen** (Integration, Test, Produktion)
- **Auswertungen und Monitoring**

© www.catalysts.cc, 2009
WWW - Warum Was Wie???
13

Für eine effiziente Entwicklung im Team muss man Einiges im Griff haben.

Zuerst einmal muss man natürlich den gesamten Source-Code, die wichtigen Projektunterlagen und Dokumente zentral ablegen – im Subversion oder CVS oder einem ähnlichen Source-Code-Control-System.

Jeder im Team muss das Gesamtsystem bauen können. Dafür hat sich Ant bewährt. Mit lokalen Property-Files kann der Entwickler den definierten Build-Prozess anpassen, wenn er den Build lokal ausführt – zum Beispiel dass man keine E-Mails verschickt und eine Hauptspeicher-Datenbank verwendet.

Ärgerlich ist es, wenn man Phantom-Fehlern nachläuft, weil unterschiedliche Versionen von Bibliotheken verwendet werden. Für die Verwaltung von Modul-Abhängigkeiten und externen Bibliotheken hat sich Maven oder dessen Nachfolger Ivy bewährt. Ivy kann man so konfigurieren, dass auch automatisch der Quellcode von Fremdbibliotheken in die Entwicklungsumgebung eingebunden wird. Das erleichtert das Debugging.

Wenn's Clients in mehreren Programmiersprachen gibt, braucht man auch die jeweiligen Programmierumgebungen. Das heißt oft, dass man im Team nicht nur Eclipse hat, sondern auch IntelliJIDEA (von JetBrains) oder das Visual Studio (von Microsoft) oder den FlexBuilder (von Adobe).

Kleinere Umstellungen muss man ja sowieso ständig machen – da heißt's ja bei testgetriebener Entwicklung „Red – Green – Refactor“.

Größere Umstellungen wird man nicht kurz vor einer Release machen und außerdem im Team gut abstimmen.

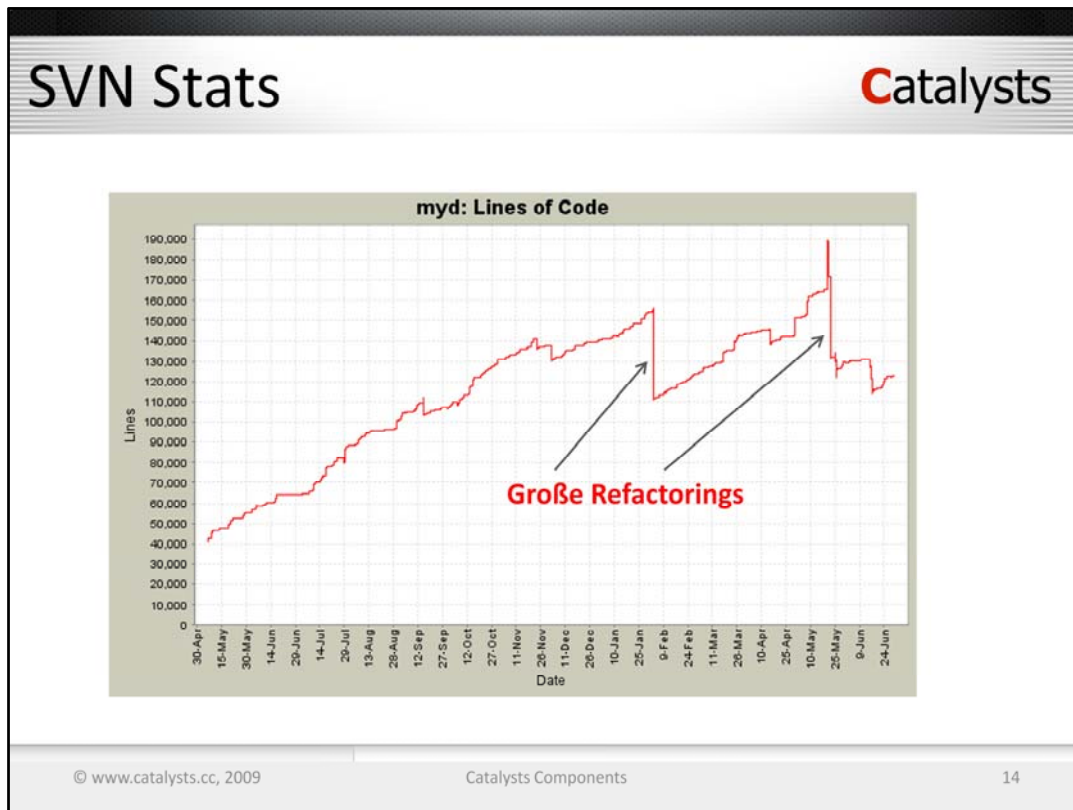
Den (eh schon definierten) Build sollte man dann natürlich auch noch automatisieren und zu einem ständigen Build ausbauen. Wir verwenden dazu TeamCity, aber Hudson oder CruiseControl gehen da natürlich auch).

Beim Continuous Build wird man natürlich die automatisierten Tests als Regressionstests mitlaufen lassen, sowie statische Code-Analysen und Architektur-Analysen.

Wenn der Build fehlschlägt, gibt es automatische Benachrichtigungen an die Entwickler, die das Problem verursacht haben.

Dann wird man den Build-Prozess so ausbauen, dass man mehrere Server-Umgebungen bedienen kann – eine Integrationsumgebung, eine Testumgebung und eine Produktionsumgebung.


Und wenn man dann noch in die Auswertungen hineinschaut und das Monitoring ernst nimmt, hat man eine effektive Qualitätskontrolle und fokussierte Qualitätsverbesserung.



In diesem Diagramm sehen wir das Wachstum eines Projekts – mit den ständigen kleinen Refactorings. Große Refactorings gibt's natürlich viel seltener.

5. Flexibilität und Unabhängigkeit Catalysts

- Minimale Anforderungen an Container / Server
 - einfacher Servlet-Container
 - einfache Datenbank
- Modulares System
 - von außen konfigurierbar
 - Dependency Injection
- Austauschbarkeit einzelner Komponenten



IBM Universal Business Adapter

© www.catalysts.cc, 2009
WWW - Warum Was Wie???
15

Die 5. Lehre haben wir weiter oben schon anklingen lassen – da geht es uns um Flexibilität und Unabhängigkeit.

Zu IBM-Zeiten habe ich mich immer gehütet, IBM-Erweiterungen zum Java-Standard zu verwenden. Den IBM WebSphere Application Server gab's in der normalen Version und in der Enterprise Version. Die IBM DB2 gab's in der normalen Version, in der Enterprise Version und in der Extended Enterprise Edition – „Triple E“ wurde die genannt.

Immer wenn man auf die Enterprise Edition – oder gar die Extended Enterprise Edition gesetzt hat, hat das bedeutet, dass man diese Entscheidung in einem halben Jahr oder ganzen Jahr bereut hat.

Die Lehre war somit: mit den einfachsten Mitteln auskommen.

Wenn man eine Java-Server-Anwendung schreibt, sollte die auch in einer einfachen Servlet-Engine laufen können.

Und mit einer einfachen Datenbank.

Vorsicht beim Einbinden externer Bibliotheken.

Vorsicht vor den großen Mistkugeln – lieber die eine große Mistkugel in drei kleinere Mistkugeln aufteilen.

Mit einem modularen System kommt man besser durch die ungewisse Zukunft.

Das System von außen konfigurierbar machen und „Dependency Injection“ verwenden.

Dann kann man hoffentlich einzelne Komponenten austauschen.

Vor ein paar Jahren hat es übrigens eine ganz tolle Werbung von IBM über die Idee eines „Universal Business Adapters“ gegeben – findet man noch auf YouTube.

6. Keine Web-Krücken mehr Catalysts

- Alle Browser, unabhängig von Browser
- Usability wie bei installierter Anwendung
 - ohne diese ständigen Wartezeiten
 - natürlich mit Drag & Drop UND Tastenkürzel
- Offline-Fähigkeit
 - zwischenzeitliche Verbindungsprobleme überleben
 - automatischer Re-Connect
 - lokale Ereignis-Warteschlange
 - automatische Datensynchronisierung
- Immer aktuelle Daten
 - auf allen Clients
 - auch ohne Neuladen
 - Server-Push

© www.catalysts.cc, 2009
WWW - Warum Was Wie???
16

Und die letzte Lehre ist, dass die Benutzer die Web-Krücken schon satt haben.

Es nervt, wenn eine Web-Anwendung nur im Internet Explorer funktioniert.

Es ist eine Zumutung, wenn man weiterhin den IE6 verwenden muss (der 2001 herausgekommen ist), weil eine wichtige Anwendung nur mit dieser IE-Version funktioniert.

Ich möchte außerdem, dass die Web-Anwendung unabhängig vom Browser ist.

Browser sind einfach immer noch instabil – stürzen halt hin und wieder ab.

Und dann sind alle Web-Seiten und alle Web-Anwendungen auf einen Schlag weg.

Das geht doch nicht so – das ist doch nicht brauchbar.

Und dann sind viele Web-Anwendungen aus Sicht der Usability noch immer eine ziemliche Frechheit: immer wieder diese Wartezeiten von 1 bis 5 Sekunden. Selten funktioniert Drag & Drop gescheit. Und wenn es Drag & Drop gibt, dann gibt es wieder keine Tastenkürzel.

Ich bin viel mit dem Zug unterwegs – von Linz nach Wien oder von Linz nach Salzburg.

Da ist – auch im Jahr 2009 – noch nicht durchgängig Internet verfügbar. Die Verbindung reißt immer wieder ab.

Und was machen dann die allermeisten Web-Anwendungen – sie sagen, dass der Server gerade nicht erreichbar ist – na super.

Oder sie tun so, als ob sie die Aktion ausgeführt hätten, ohne dass der Server was davon mitbekommen hat.

Der Google Calendar macht das übrigens so: da kann man – ohne Serververbindung – einen Kalendereintrag eingeben. Man glaubt, dass er abgespeichert ist – man sieht ihn ja auch im Kalender. Und wenn man den Browser schließt und das nächste Mal in den Kalender schaut, ist der Termin weg.

Was ich haben möchte, ist eine Offline-Fähigkeit – zumindest in einem gewissen Umfang:

Die Anwendung sollte zwischenzeitliche Verbindungsprobleme überleben.

Sie sollte sich automatisch neu verbinden.

Ich möchte lokal weiterarbeiten können und die lokalen Daten sollten mit dem Server abgeglichen werden, wenn der Server wieder erreichbar ist.

Und dann bin ich es auch schon leid, dass die angezeigten Daten nur vermeintlich aktuell sind.

Eine Nachrichtenseite wie „orf.at“ muss man immer wieder „neu laden“. Das nervt.

Bei einer wichtigen Anwendung möchte ich nicht immer wieder neu laden müssen. Ich möchte automatisch immer die aktuellen Daten angezeigt bekommen.


Natürlich auf allen Clients – im Web-Browser, im installierten Programm, auf dem Handy, im Office-Plugin, wo auch immer.

Da kann man doch nicht immer auf F5 drücken oder die „Web-Seite“ neu laden.

Das Web hat sich ja emanzipiert von Web-Seiten zu Web-Anwendungen. Und Anwendungen kann man nicht immer neu laden müssen.

Das muss auch intelligenter gehen – so wie die Nachrichten automatisch auf den Blackberry kommen.

Zusammenfassung




1. Klare Trennung zwischen Client und Server
2. Saubere Software-Architektur
3. Testgetriebene Entwicklung
4. Entwicklung im Team
5. Flexibilität und Unabhängigkeit
6. Keine Web-Krücken mehr

© www.catalysts.cc, 2009 WWW - Warum Was Wie???

Tja, das waren die 6 Lehren.

Quintessenz



- Vorteile von Web-Anwendungen
 - (fast) keine Installation, automatische Updates
 - skalierbare Infrastruktur
 - von allen Plattformen aus verwendbar
 - immer aktuelle Daten (Online-Zugriff; Server-Push)
 - überall dabei (Notebook, Handy)
- Vorteile von installierten Anwendungen
 - funktionieren auch, wenn Server nicht erreichbar
 - Flüssiges Arbeiten mit Drag & Drop und Tastenkürzel
 - Office-Integration
- Catalysts Plattform
<http://www.catalysts.cc/solutions/platform/>

© www.catalysts.cc, 2009 WWW - Warum Was Wie???

Die Quintessenz ist, dass man die Vorteile von Web-Anwendungen und von installierten Anwendungen haben möchte.

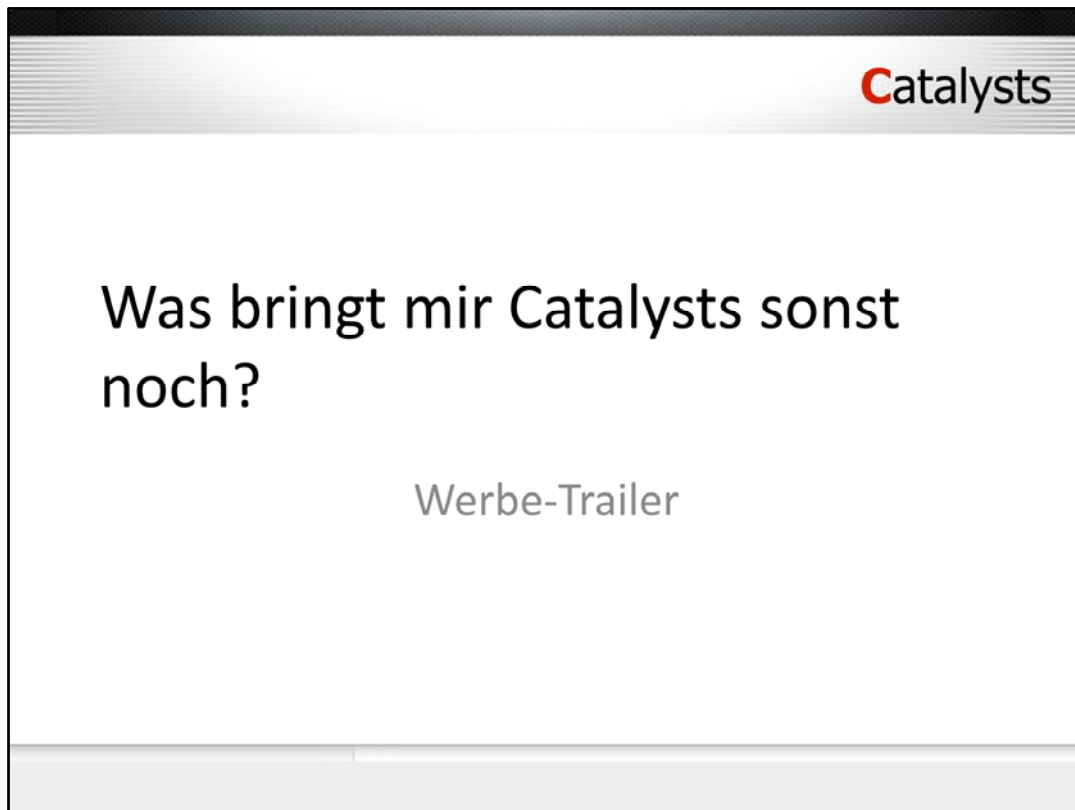
Ohne die Nachteile der beiden.

Wir haben uns eine technologische Plattform und Entwicklungsumgebung geschaffen, die diese Vorteile hat.

In den nächsten paar Wissensspritzen werden wir einige interessante Bereiche davon beleuchten.

Ein bisschen was darüber steht auf unserer Homepage.

Das war's für heute – danke fürs Interesse.



Effizient-Katalysator fürs Team

Catalysts

Nur perfekt funktionierende Teams arbeiten
hocheffizient.



hilft Teams bei der Planung und Organisation.

Registrieren Sie sich gratis unter
<http://www.taskmind.net>

Gesundheitscheck für Projekte

Catalysts

- „Wenn ich vorher gewusst hätte, dass all die Probleme auftreten, dann wären wir das Projekt anders angegangen...“
- Sind Ihnen zu Projektbeginn alle technischen und organisatorischen Risiken bewusst?
- Oder gibt's auch bei Ihnen immer wieder viel Unvorhergesehenes?
- Fordern Sie heute noch ein Experten-Team von Catalysts für eine Vorsorgeuntersuchung für Ihr Projekt an. 400 Euro, die sich auszahlen!

Wir setzen Ihre Ideen um

Catalysts

- Sie wissen was – wir wissen wie
- Mit gewohnter Catalysts-Qualität
- Zu vernünftigen Preisen
- Schnell
 - erster Prototyp nach wenigen Tagen
 - Wochenweise mehr Funktionalität
- Probemonat – Ausprobieren und nichts riskieren!

- Wir entwickeln **Software nach Ihren Bedürfnissen**
 - für den Büroarbeitsplatz,
 - für unterwegs am Notebook,
 - für Ihr Handy.
- Wir entwickeln **Software auf agile Art.**
- Dadurch gibt's
 - frühzeitige und regelmäßige Auslieferung,
 - rasches Feedback und
 - ausschließlich wertvolle Funktionen im Produkt, keine Schnörkel.

Karin S. über Catalysts

Catalysts

Karin S. (Geschäftsführerin eines kleinen Dienstleistungsunternehmens, Nicht-IT)

“Ich leite ein kleines Dienstleistungsunternehmen (16 Mitarbeiter). Wir brauchen zuverlässige Simulationssoftware nach Maß, um unsere Aufträge schneller abwickeln zu können. Wir haben selbst keine Software-Entwickler. Ich kenne mich mit Software nicht aus, verwende sie nur. Über unseren bisherigen Softwarelieferanten ärgere ich mich, weil er für jede kleine Änderung Länge mal Breite verrechnet.”

Erfahrungen von Karin S. mit Catalysts:

- [Günstiger](#) als andere Firmen in OÖ
- „Ausprobieren und nichts riskieren!“ ([Probemonat](#))
- [Wertvolles zuerst, Unwichtiges später, Schnörkel gar nicht](#)
- [Papier-Prototyp nach einer Woche, erste Version nach einem Monat](#)
- [Monatliche Auslieferung](#) und monatliche Kurz-[Retrospektiven](#)
- [Änderungen gratis](#)
- [Von Profis geführt, kritische Denker](#)

Hans M. über Catalysts



Hans M. (Abteilungsleiter IT in einer größeren Firma)

“Ich leite die Softwareentwicklung (25 Entwickler) in einer größeren Firma. Meine Leute sind mit der Wartung der alten Programme schon ausgelastet. Sie kommen bei den neuen Technologien allerdings nicht mehr mit. Aus den Fachabteilungen kommen immer mehr Anforderungen, die wir nicht erfüllen können. Wir suchen ständig nach guten Software-Entwicklern, finden die aber nicht.”

Erfahrungen von Hans M. mit Catalysts:

- [Misch-Stundensatz](#) unter unseren internen Stundensätzen
- [Kreativere und bessere Lösungen](#)
- [Scrum, XP, TDD, laufende Integration, Personas, User Stories](#)
- [Stabile Technologie-Plattform](#)
- Modulare und zyklenfreie Architektur, [ausführlich getestet](#)
- [Unternehmensberatung inbegriffen](#)
- Exzellentes Team