



Hallo und willkommen zur 17. Wissensspritze.

Mein Name ist Christoph Steindl und die heutige Wissensspritze dreht sich um das Thema „**Architektur-Refactoring**“

Christoph Steindl hielt diesen Vortrag am 23.10.2009.

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

Zeitachse **Catalysts**

- Vor dem Projekt
 - Erste Architekturüberlegungen
- Zu Beginn des Projekts
 - Initiale Architektur erstellen
- Während der Arbeit an der Release
 - Architektur anpassen
- Am Beginn der neuen Release
 - Architektur für neue Release erarbeiten
 - Bestehende Architektur umbauen

Architektur-Refactoring

© www.catalysts.cc, 2009 Architektur-Refactoring 2

Bevor man an Architektur-Refactoring denken kann, muss man zuerst einmal „DIE“ Architektur haben.

Somit schauen wir uns einmal auf der Zeitachse an, wann in einem Software-Projekt welche Architekturarbeiten anfallen.

Schon vor dem offiziellen Projektstart stellt man typischerweise die ersten Architekturüberlegungen an. Man muss die Aufgabenstellung, das geschäftliche Umfeld und das IT-Umfeld verstehen, um eine adäquate Lösung konzipieren zu können.

Zu Beginn des Projekts erstellt man dann die initiale Architektur, die man während der Arbeit an der ersten Release natürlich noch anpassen muss.

Wenn das entstandene System weiterentwickelt wird und es weitere Releases gibt, kann es schon sein, dass die neuen Anforderungen mit der bestehenden Architektur nicht gut umgesetzt werden können.

In so einem Fall entwirft man dann die Zielarchitektur für das neue Release und muss dann die bestehende Architektur entsprechend umbauen.

Und genau um derartige größere Umstellungen der Architektur geht es uns in der heutigen Wissensspritze.

Vor dem Projekt: Patterns for e-business**Catalysts**

- Optionen aufstellen und diskutieren
- Vom primären Benutzerziel (Self Service, Data Mining, e-commerce, B2B usw.) zu Server-Strukturen und zur Produktauswahl

- IBM-Initiative
 - <http://www.ibm.com/developerworks/patterns/>
 - Adams et al. : [Patterns for e-business: A Strategy for Reuse](#)
 - Viele „Redbooks“ (<http://www.redbooks.ibm.com>)

© www.catalysts.cc, 2009Architektur-Refactoring3

Als ehemaliger IBMer fällt mir zu den ersten Architekturüberlegungen vor dem Projektstart die „Patterns for e-business“-Initiative der IBM ein.

Die waren sehr nützlich, wenn man Optionen aufstellen und diskutieren musste – innerhalb weniger Minuten konnte man die unterschiedlichen Szenarien mit einem technisch versierten Entscheidungsträger durchgehen.

Zuerst grenzte man das primäre Benutzerziel ein – die Frage „worum geht es aus Benutzersicht eigentlich“. Man hat dann gleich einmal Überblicksdiagramme mit den Server-Strukturen und kommt im nächsten Schritt zur Produktauswahl.

Es gibt zu dieser IBM-Initiative eine gute Webseite, ein gutes Buch und etliche Redbooks, das sind Bücher, die von Technikern für Techniker geschrieben wurden, meistens mit einem klaren Produkthintergrund, z.B. „Wie baut man serielle und parallele Prozesse mit dem WebSphere Process Server V6?“

Auf den nächsten drei Folien steht etwas mehr über diese „Patterns for e-business“, für die heutige Wissensspritze soll aber diese Mini-Einleitung auch schon wieder reichen.

Wir springen somit weiter zum Projektbeginn.

Patterns for e-business

Catalysts

- **IBM Patterns for e-business extend the domain of software patterns to earlier phases of the application development life cycle.**
 - These patterns help to us understand and analyze complex business problems and break them down into smaller, more manageable functions that can then be implemented using lower-level patterns.
- **Patterns e-business define the following assets:**
 - **Business patterns:** establish the primary business purpose of a solution; they identify the interaction between users, businesses, and data
 - **Integration patterns:** tie multiple business patterns together
 - **Composite patterns:** represent commonly occurring combinations of Business patterns and Integration patterns (such as Electronic commerce, Portal, Sell-Side Hub, Buy-Side Hub, and so on)
 - **Application patterns:** refine Business patterns so they can be implemented as computer-based solutions (in form of application components and data)
 - **Runtime patterns:** define the logical middleware structure supporting an Application pattern (in form of middleware nodes and interfaces)
 - **Product mappings:** identify proven and tested software implementations for each Runtime pattern
 - **Best-practice guidelines:** document benefits and pitfalls during design, development, deployment, and management

© www.catalysts.cc, 2009
Architektur-Refactoring
4

The IBM Patterns for e-business are structured into a layered set of Architectural Patterns that bridge the business and IT gap, providing new layers of abstraction that extend the domain of Software Patterns to earlier phases of solution design and development. These patterns also provide a basis for problem decomposition by helping the architect to understand and analyze complex business problems and break them down into smaller, more manageable functions.

These patterns are organized into a layered asset model that provides a pattern selection decision-making framework. Pattern selection begins by identifying patterns that characterize the business context. Then, selection proceeds to application and runtime topology patterns. This selection process provides traceability from business requirements to the technical aspects of the solution and also encourages the participation of both technical and non-technical stakeholders. This structure is referred to as the layered asset model.

The Patterns for e-business are composed of four basic types of patterns and a type of pattern that represents cases where multiple basic patterns apply. At the top layer of the layered asset model are business and integration patterns. Also at this layer are composite patterns, which represent frequently occurring cases where multiple business or integration applies. A group of Application patterns representing high-level application topology alternatives are linked to each Business and Integration pattern. Likewise, Application patterns have one or more Runtime Patterns associated with them to represent operational or runtime topology alternatives. Products are mapped to the runtime pattern, completing the picture. All of the pattern selection decisions are linked together, providing a way to trace technology decisions back to business and IT requirements and drivers.

Business Patterns highlight the most commonly observed interactions between users, businesses, and data of most e-business solutions, such as Self-Service, Collaboration, Information Aggregation, and Extended Enterprise.

Integration Patterns address more complex e-business applications that combine multiple Business Patterns at the front-end via Access Integration or at the back-end via Application Integration. Integration Patterns are differentiated from Business Patterns in that they themselves do not automate specific business problems. Rather, they combine Business Patterns to support more advanced functions or to make Composite Patterns feasible.

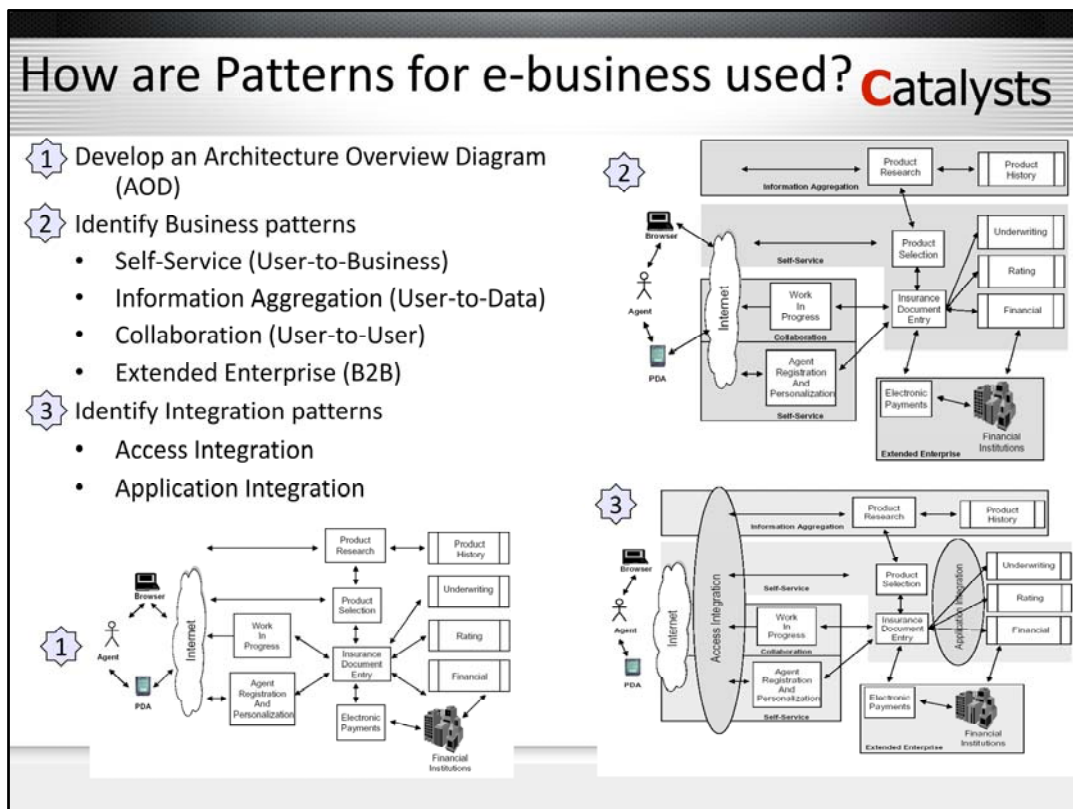
Composite Patterns combine Business Patterns and Integration Patterns to create advanced e-business solutions. When a frequently repeating combination is identified, it is documented as a composite pattern. Composite patterns typically signal the availability of sophisticated off-the-shelf products, such as Commerce and portal servers.

Application patterns refine Business patterns so they can be implemented as computer-based solutions (in form of application components and data).

Runtime patterns define the logical middleware structure supporting an Application pattern (in form of middleware nodes and interfaces).

Product mappings identify proven and tested software implementations for each Runtime pattern.

Best-practice guidelines document benefits and pitfalls during design, development, deployment, and management.

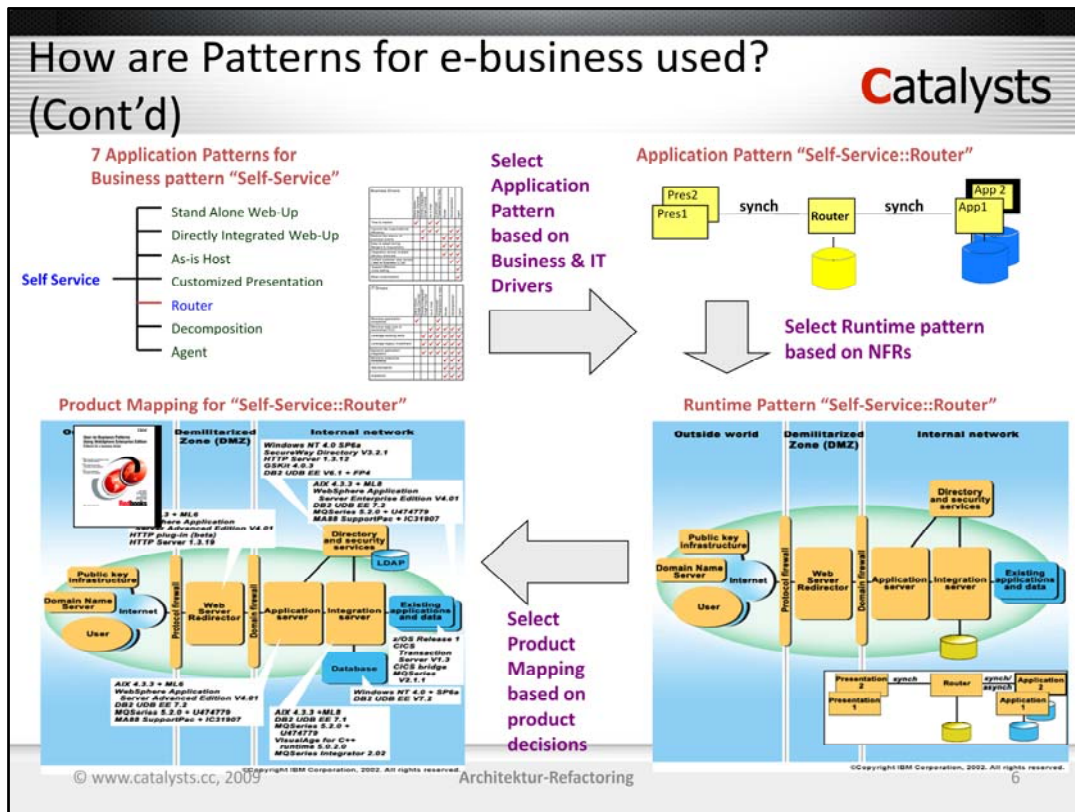


When faced with the challenge of designing a solution for a business problem, the first step is to take a high-level view of the goals you are trying to achieve. A proposed business scenario should be described. An Architecture Overview Diagram is an effective means to communicate the solution.

Next, you try to map the problem to one or several Business patterns.

It would be very convenient if all problems fit nicely into the four slots of Business patterns (Self-Service, Information Aggregation, Collaboration and Extended Enterprise), but reality is such that things will often be more complicated. The patterns assume that most problems, when broken down into their most basic components, will fit more than one of these patterns.

When a problem requires multiple Business patterns, Patterns for e-business provide additional patterns in the form of Integration patterns that tie together multiple Business patterns.



After the Business pattern is identified, the next step is to define the high-level logical components that make up the solution and how these components interact. This is known as the Application pattern. A Business pattern will usually have multiple possible Application patterns. You select the right application patterns by weighing the business and IT drivers as sketched in the tables.

Application patterns break the application down into the most basic conceptual components (for example, logical components that describe a presentation tier for interacting with users, an application tier, and a back-end application tier), identifying the goal of the application. In our example, the application falls into the Self-Service business pattern, and the goal is to build an application that provides access to existing back-end systems through different channels.

The Application pattern can be further refined with more explicit functions to be performed. Each function is associated with a runtime node. You select the right Runtime pattern based on the requested Non-Functional Requirements. In reality, these functions, or nodes, can exist on separate physical machines or can coexist on the same machine. In the Runtime pattern, this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure.

As an example, let's assume that our customer has determined that his solution fits into the Self-Service business pattern and that the Router pattern is the most descriptive of the situation. The next step is to determine the Runtime pattern that is most appropriate for the customer's situation.

By overlaying the Application pattern on the Runtime pattern, you can see the roles that each functional node will fulfill in the application.

The last step in defining the network structure for the application is to correlate real products with one or more runtime nodes. The Patterns Web site shows each Runtime pattern with products that have been tested in that capacity. The Product mappings are oriented toward a particular platform, though more likely the customer will have a variety of platforms involved in the network. In this case, it is simply a matter of mixing and matching.

Application Patterns represent the partitioning of application logic and data, and the styles of interaction between the logic tiers. Several Application Patterns are defined for each Business and Integration Pattern. Like the Business and Integration patterns, Application Patterns are technology-agnostic. Also like the Business and Integration patterns, functional requirements drive the Application pattern selection process. A difference between Application patterns and the Business and Integration patterns is that Application patterns introduce high-level topologies.

An Application Pattern can be implemented by using a Runtime Pattern, which documents technical nodes, such as Web servers or firewalls, necessary to implement the chosen Application Pattern. Non-functional requirements or NFRs are the primary consideration when you choose between Runtime Pattern alternatives for a given Application Pattern.

Each business and integration pattern has an associated set of business and IT drivers that can be assessed and prioritized to narrow the Application pattern selection process. In this example, assume that the highest priority requirements and drivers have identified the "Router" Application Pattern as a good choice. The Router provides intelligent routing and transformation of requests between multiple delivery channels and back-end applications. The Router Application pattern has a Runtime pattern topology, to which products can be assigned. The Patterns for e-business Web site documents product mappings for platforms, such as xSeries, pSeries, and zSeries, and operating systems, such as AIX, Linux, zOS, and Windows.

IBM Redbooks provide detailed documentation of how to implement solutions using these patterns, including technology scenarios, guidelines for design and development, and examples of actual implementations.

Zu Beginn eines Projekts Catalysts

- **Initiale Architektur erstellen**
 - Input
 - Current IT Environment
 - System Context
 - Non-Functional Requirements
 - Output
 - Architecture Overview Diagram
 - Component Model
 - Operational Model

© www.catalysts.cc, 2009Architektur-Refactoring7

Zu Beginn eines Projekts erstellen wir die initiale Architektur.

Wieder zu IBM-Zeiten hat es da geheißen, dass wir ein Architektur-Überblicks-Diagramm, ein Component Model und ein Operational Model erstellen.

Ausgangsbasis für diese Architektur-Dokumente ist oft mit dem „Current IT Environment“ die Beschreibung der Hardware und Software, die es im Unternehmen schon gibt, weil man ja selten auf der grünen Wiese anfängt.

Oft gibt es mit dem „System Context“ auch noch die Beschreibung, welche Akteure und externen Systeme mit dem neuen System interagieren, welche Daten rein und rausgehen, somit welche Schnittstellen es geben muss. Beim System Context geht es hauptsächlich um die Abgrenzung, was ist neu zu entwickeln und was gibt es schon und muss „nur“ mehr angebunden werden.

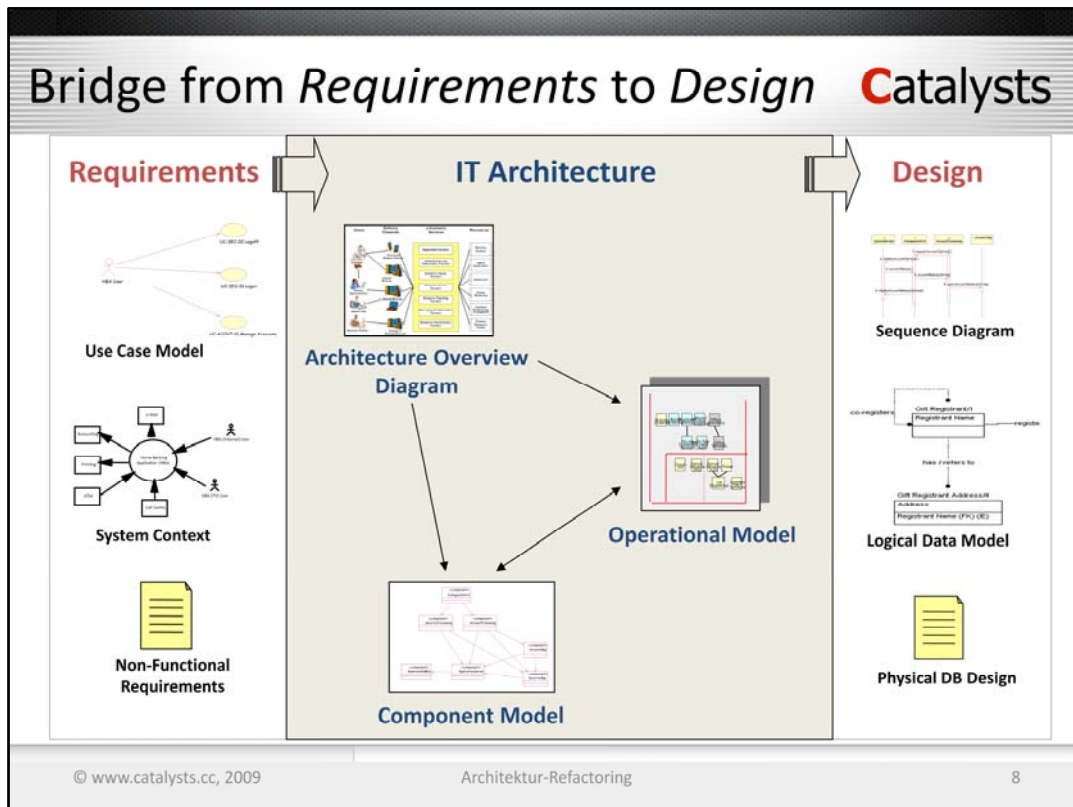
Und die Non-Functional Requirements beschreiben, wie verfügbar, performant, sicher usw. das neue System werden musste.

Das Architektur-Überblicks-Diagramm gibt es oft in zwei Varianten: einmal aus Geschäfts-Sicht und einmal aus technischer Sicht.

Das Component Model beschreibt, welche fachlichen Komponenten es im System geben soll und wie sie miteinander interagieren, um das Systemverhalten darzustellen.

Das Operational Model beschreibt, welches Stück Software auf welchem Stück Hardware laufen soll.

Das ist – wie gesagt – wie ich zu IBM-Zeiten als IT-Architekt die Software-Architektur für IT-Projekte erstellt habe.



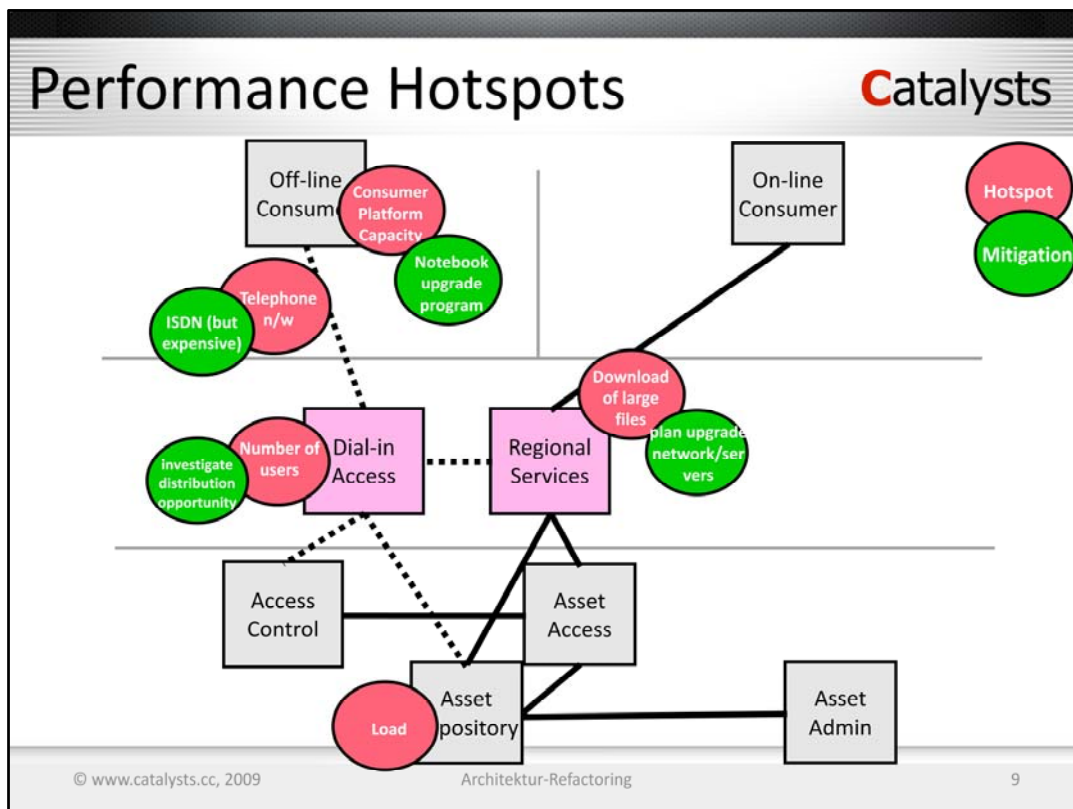
Die Architektur-Arbeit am Anfang des Projekts sollte somit die Brücke schlagen von den Anforderungen zum detaillierten Entwurf. Indem man sich die wichtigen Fragen frühzeitig stellt, kann man oft spätere Nacharbeiten und Änderungen vermeiden.

To be successful during infrastructure design, you must understand and balance requirements and constraints. Requirements and constraints are capabilities needed by a user to solve a problem or achieve an objective. They answer the question of what does the customer want versus how it is built. (Customers state the constraints imposed upon the desired solution.) Requirements and constraints help define measures of success or failure. They enable communication between user, developer, and architect.

Requirements and constraints are a means of defining client goals and needs. They are the foundation for managing scope, quality, and client expectations. They are the baseline for validating testing activities. Requirements and constraints also influence architecture development, such as structuring and placement decisions, solution sizing and costing, and product selection, deployment, and configuration framework.

All kinds of constraints on an IT system are important to understand. These include business constraints like geographic locations, risk willingness, volumes, and service levels. These also include IT standards and Vendor Preferences, such as J2EE™ compliance, and current infrastructure constraints (for example, must run on specified existing middleware; or what is the existing infrastructure hardware, software, or network).

The diagram on this slide shows how business requirements impact solution design. It also emphasizes how the development of the Component Model occurs in parallel to the development of the Operational Model.

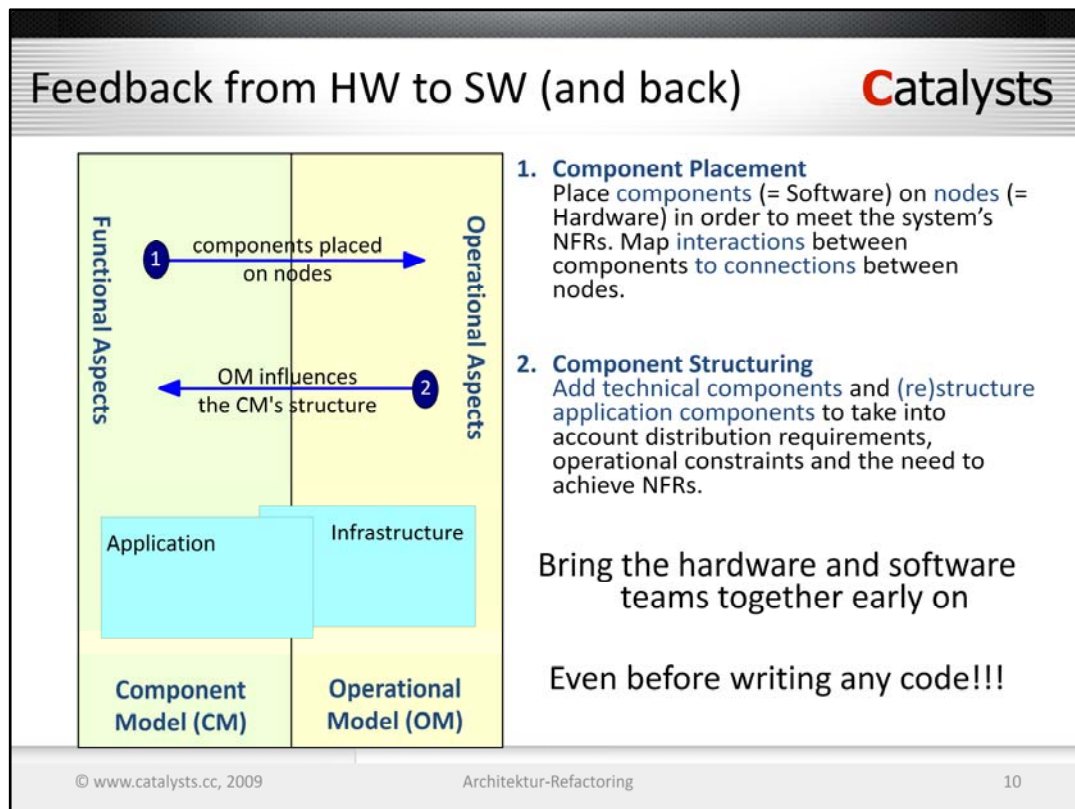


Schon anhand der einfachen Skizzen des ersten operationalen Modells kann man sich überlegen, welche Hotspots auftauchen können und was man dagegen tun kann. Das operationale Modell besteht aus Lokationen, Rechnern und Netzwerkverbindungen.

Im Diagramm sehen wir unterschiedliche Bereiche oder Zonen. Und in jeder Zone stehen ein oder mehrere Computer.

Die Striche zwischen den Computern deuten an, wie viel bzw. schnell Daten über eine Leitung drübergehen.

Wenn z.B. ein Hotspot der Arbeitsplatzrechner ist, weil die dortige Festplatte zu klein ist, kann man überlegen, ob nicht parallel zum Projekt die Notebooks aufgerüstet werden könnten. Oder ob eine Offline-Lösung überhaupt sinnvoll ist. Wenn man das gleich am Anfang diskutieren kann, ist es besser, als wenn man das erst später macht. Weil man später vielleicht schon die halbe Offline-Lösung implementiert hat.



Wenn man sich überlegt hat, welche Komponenten es geben wird, versucht man als nächstes gleich einmal, diese (angedachten) Komponenten auf der (angedachten) Hardware zu installieren.

Wenn eine Komponente eine andere braucht, weil sie z.B. ein Web-Service der anderen Komponente aufruft, muss es zwischen den beiden "Hardware-Knoten" natürlich auch eine Netzwerk-Verbindung geben.

Man kann Szenarien aus Benutzersicht durchspielen und überlegen, welche Software auf welchem Rechner am Szenario beteiligt ist.

Man kann sich überlegen, wie das System reagieren soll, wenn ein Rechner bzw. eine Verbindung ausfällt.

Man kann Szenarien aus Betriebssicht durchspielen, etwa wie man die Software verteilt und wie man neue Versionen ausliefert oder wie man die Daten sichert.

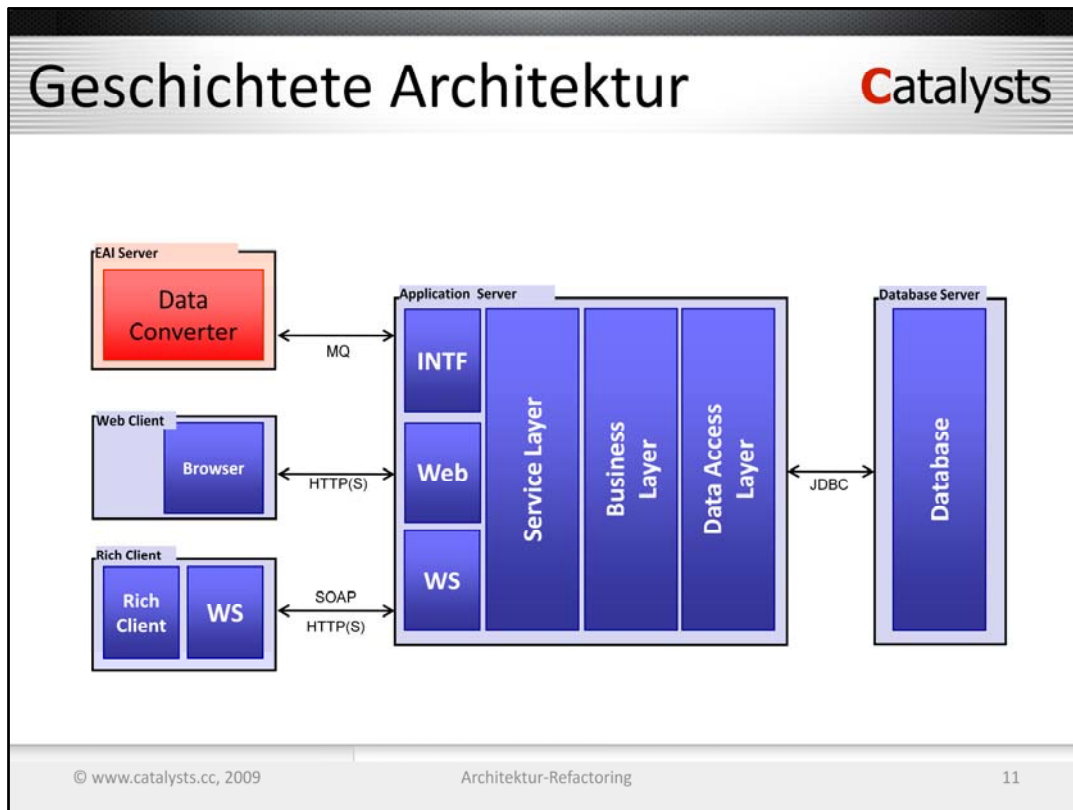
Man kann technische Komponenten wie z.B. einen Lastverteiler hinzufügen und schauen, welche Auswirkungen diese technischen Komponenten auf die fachlichen Komponenten haben.

Oft fallen Knackpunkte ja erst beim Systemtest auf, weil erst da die Betriebsabläufe durchgespielt werden.

Mit der Vorgehensweise, die Komponenten sehr früh schon gedanklich auf Hardware zu platzieren und die Auswirkungen durchzudiskutieren, kann man sich natürlich auch das nachträgliche Architektur-Refactoring in etlichen Fällen sparen.

Der wichtige Punkt ist, dass man all diese Diskussionen als Software-Techniker mit den Hardware-Technikern überhaupt erst einmal führen kann.

Und dass man das tun kann, noch bevor die erste Zeile Code geschrieben worden wäre.

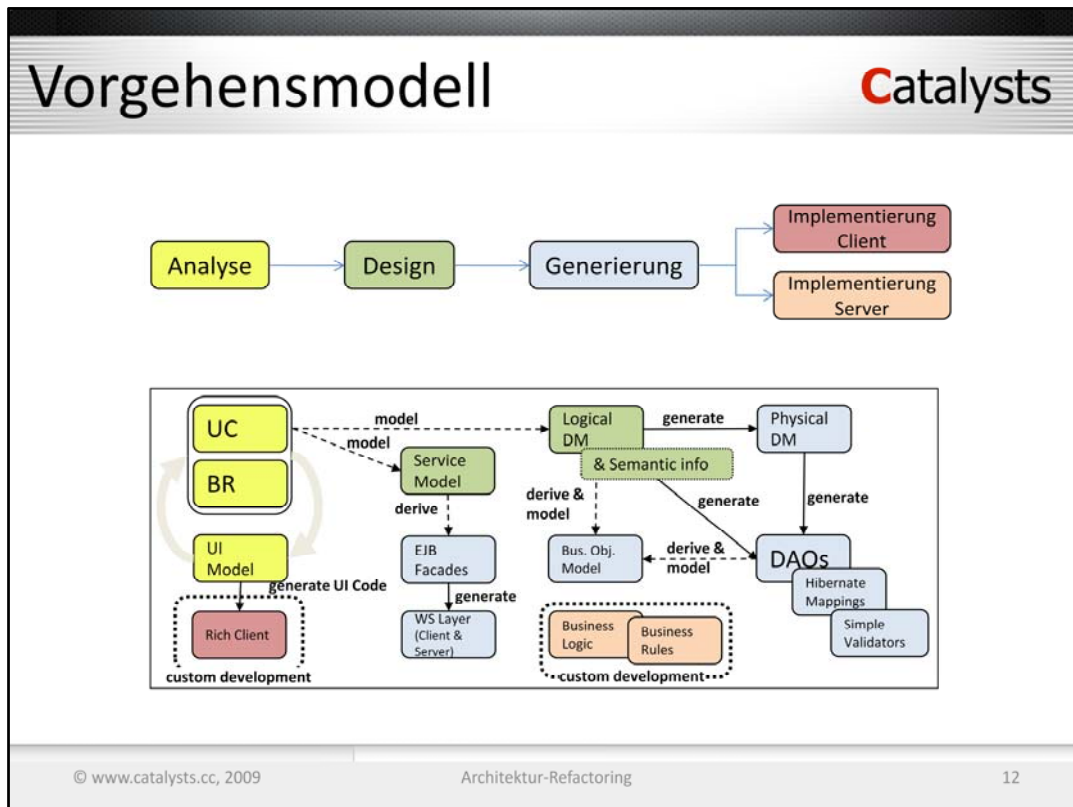


Wenn man jetzt einem Architekten freien Lauf lässt, dann entsteht sehr schnell eine schön geschichtete Architektur.

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.



Man kann jetzt als Architekt auch noch vorgeben, wie aus den Analyse-Ergebnissen, der Entwurf entstehen soll und wie daraus das Gerüst der Anwendung generiert wird.

Wo man als Client-Entwickler „nur noch“ hingreifen muss und wo man als Server-Entwickler „nur noch“ hingreifen muss.

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 generiert.

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.

Das wäre jedenfalls das Ziel – ah ja, und das Ergebnis sollte natürlich auch noch genau dem Architekturbild von der vorigen Folie entsprechen.

Aber Vorsicht!



Software-Architektur auf Papier veraltet schnell

Vorstellungen über Software-Architektur in den
Köpfen sind meist nicht deckungsgleich

?

Geplante Software-Architektur auf Papier

==

Faktische Software-Architektur im Code

?

© www.catalysts.cc, 2009Architektur-Refactoring13

Aber VOOOORRR-Sicht!

Die Software-Architektur auf Papier veraltet sehr schnell.

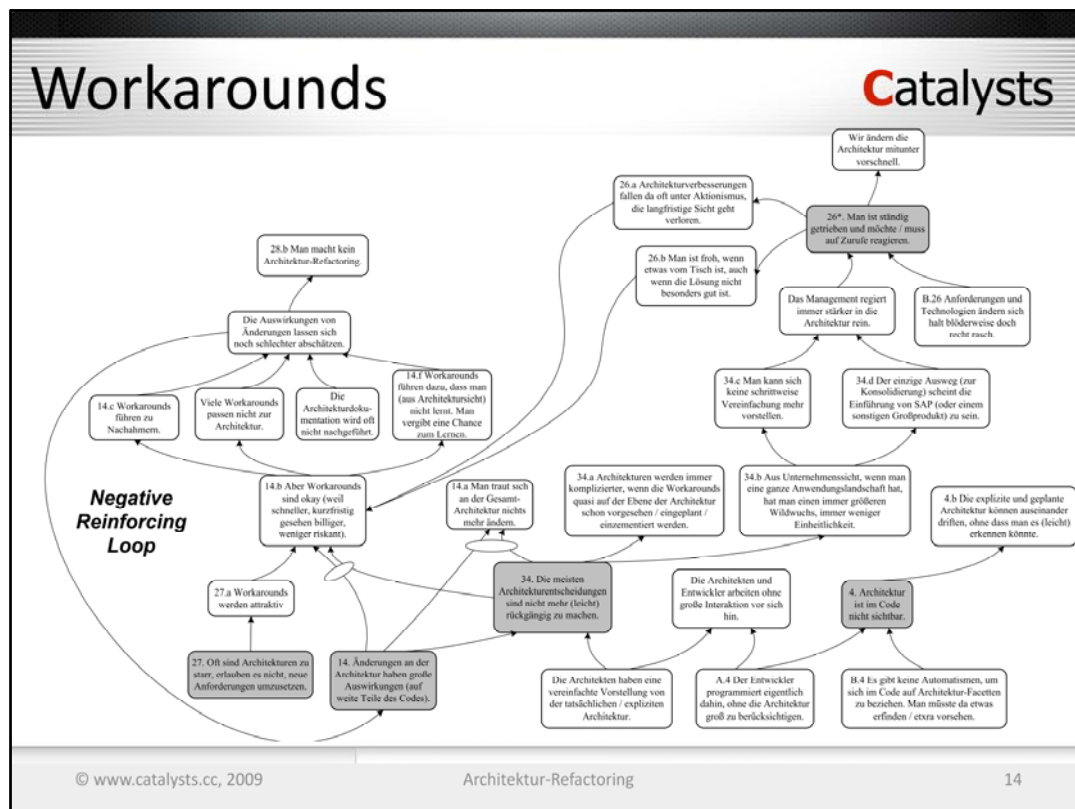
Nicht alles, was sich der Architekt ausgedacht hat, lässt sich dann auch leicht genau so realisieren.

Und nicht alles, was sich der Architekt gedacht hat, ist in der Software-Architektur sichtbar geworden, geschweige denn schriftlich festgehalten.

Somit gibt es schnell unterschiedliche Auffassungen, wie etwas im Detail zu verstehen und zu machen ist.

Die Vorstellungen in den Köpfen sind meist nicht deckungsgleich.

Und die geplante Software-Architektur auf Papier ist meist schöner als die faktische Software-Architektur im Code.



Wir haben uns vor ein paar Jahren angeschaut, welche Gründe es geben kann, warum die faktische Software-Architektur oft von der geplanten Architektur abweicht.

Ein wesentlicher Grund ist: Änderungen an der Architektur haben große Auswirkungen und werden deshalb als gefährlich empfunden.

Kleine Workarounds erscheinen attraktiver, weil sie schneller erledigt sind.

Workarounds führen allerdings zu Nachahmern: wenn der eine Workaround erlaubt war, wird der nächste Workarounds auch erlaubt sein.

Das Blöde ist, dass man aus Architektur-Sicht eine Chance vergibt zu lernen. Eigentlich war die aktuelle Architektur zu starr bzw. hat wo nicht ganz gepasst.

Hätte man sich die Zeit genommen für ein Architektur-Refactoring, hätte man die Architektur langfristig flexibler gemacht.

Weil man sich aber nicht die Zeit für ein Architektur-Refactoring genommen hat, wird die Architektur immer starrer und BRAUCHT dann sogar immer mehr Workarounds.

Rechts unten steht noch, dass die Entwickler oft dahinarbeiten, ohne sich groß um die Architektur zu kümmern.

Weil die faktische Architektur nicht sichtbar wird, fällt das auch meistens keinem auf.

Automatische Architekturüberprüfungen Catalysts

Mit Build Server Plugins wie ClassCycle können Sie Beziehungen zwischen Klassen, Paketen und Schichten festlegen und automatisch überprüfen lassen.

Datei: classcycle.ddf

<p>Server Schichtung festlegen:</p> <pre>layer api = [api] layer dao = [dao] layer event = [event] layer exception = [exception] layer service = [service] layer util = [util]</pre> <p>Mit z.B.:</p> <pre>[service] = #\${package}.*.*Service ... check layeringOf util exception dao service</pre>	<p>Client Schichtung festlegen:</p> <pre>layer command = [command] layer context = [context] layer mediator = [mediator] layer proxy = [proxy] layer repository = [repository]</pre> <p>Mit z.B.:</p> <pre>[command] = #\${package}.*.*Command ... check layeringOf context repository service proxy command mediator</pre>
--	---

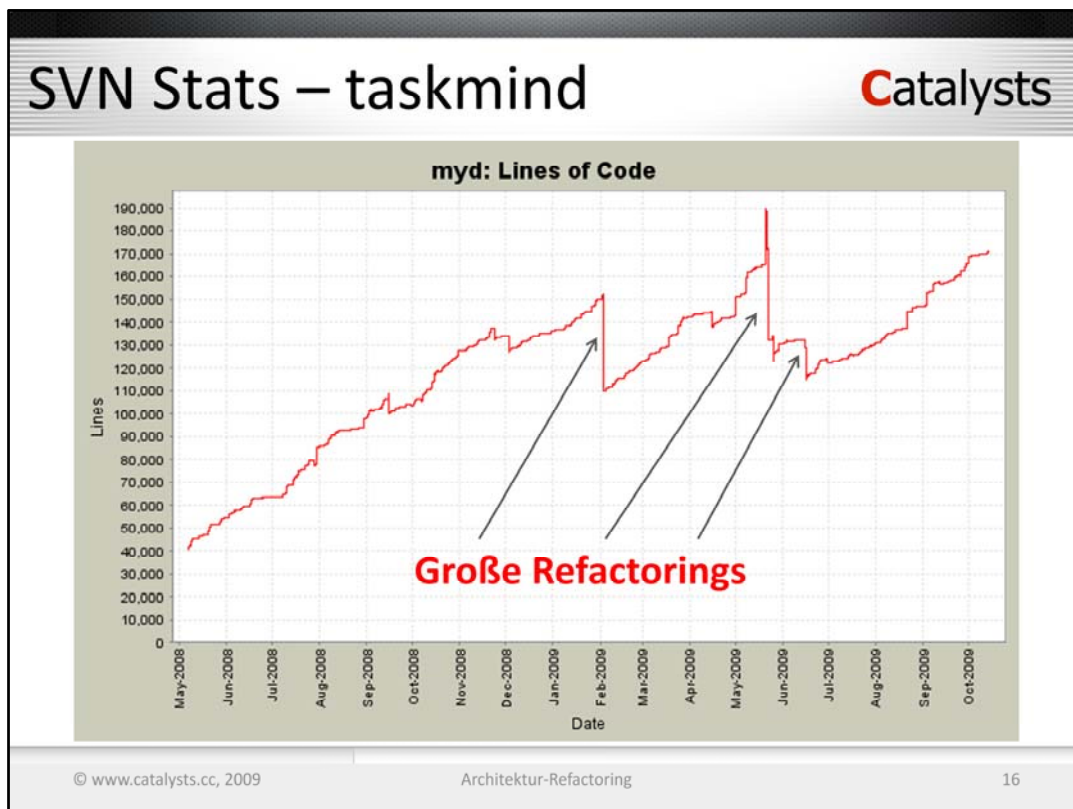
© www.catalysts.cc, 2009 Architektur-Refactoring 15

Praktisch kann man nicht leicht sicherstellen, dass die faktische Software-Architektur im Source Code mit der geplanten Software-Architektur übereinstimmt.

Aus unserer Erfahrung geht das nur, indem man die Architektur in Regeln beschreibt, die automatisch überprüft werden können.

Diese automatischen Architekturprüfungen laufen beim Continuous Build mit und schlagen Alarm, sobald irgendeine Architektur-Regel verletzt ist.

Wie das im Detail geht haben wir ja schon in der 15. Wissensspritze: Automatische Prüfungen im Build-Prozess am 25.9.2009 erzählt (<http://wissensspritze.catalysts.cc/themen/automatische-pruefungen-im-build-prozess/>).



Schauen wir uns ein konkretes Beispiel an – wann wir im letzten Jahr bei taskmind größere Umstellungen vorgenommen haben.

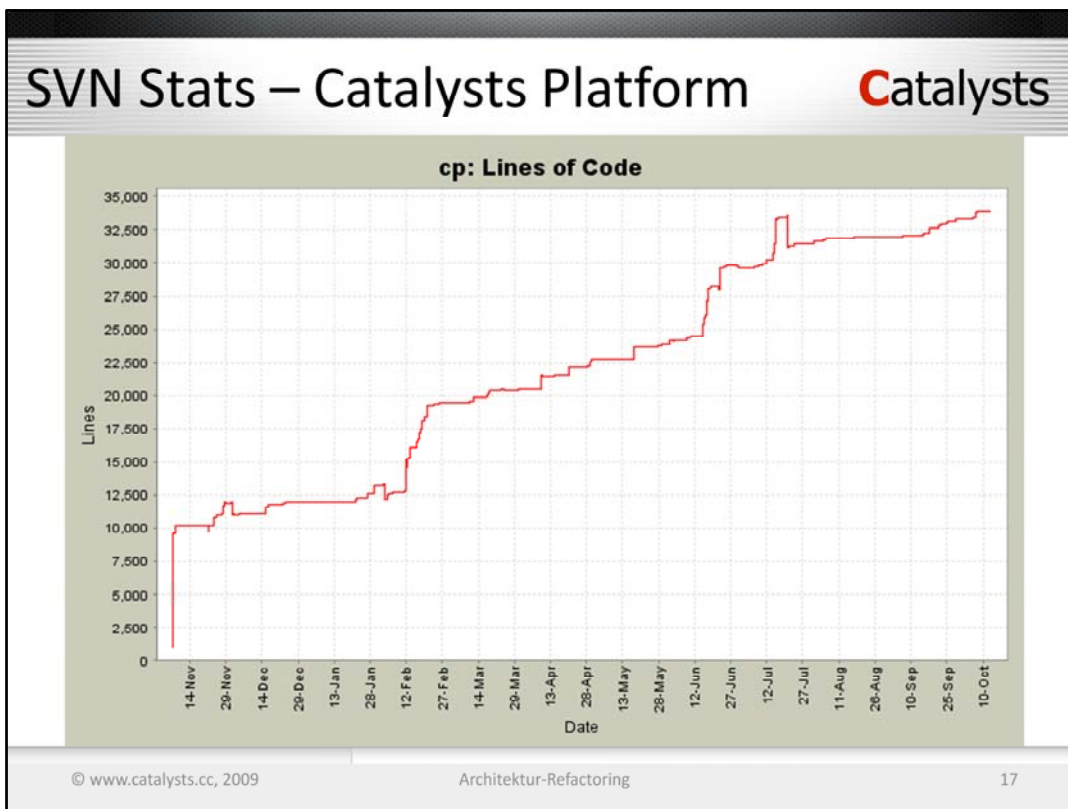
Da gab's im Februar 2009 eine große Umstellung mit dem Herauslösen der Catalysts-Plattform. Die Arbeiten an der Catalysts-Plattform haben schon im November 2008 begonnen, aber erst im Februar 2009 haben wir die Teile, die vom taskmind-Source Code in die Catalysts-Plattform gerutscht sind, in den taskmind-Projekten auch tatsächlich gelöscht.

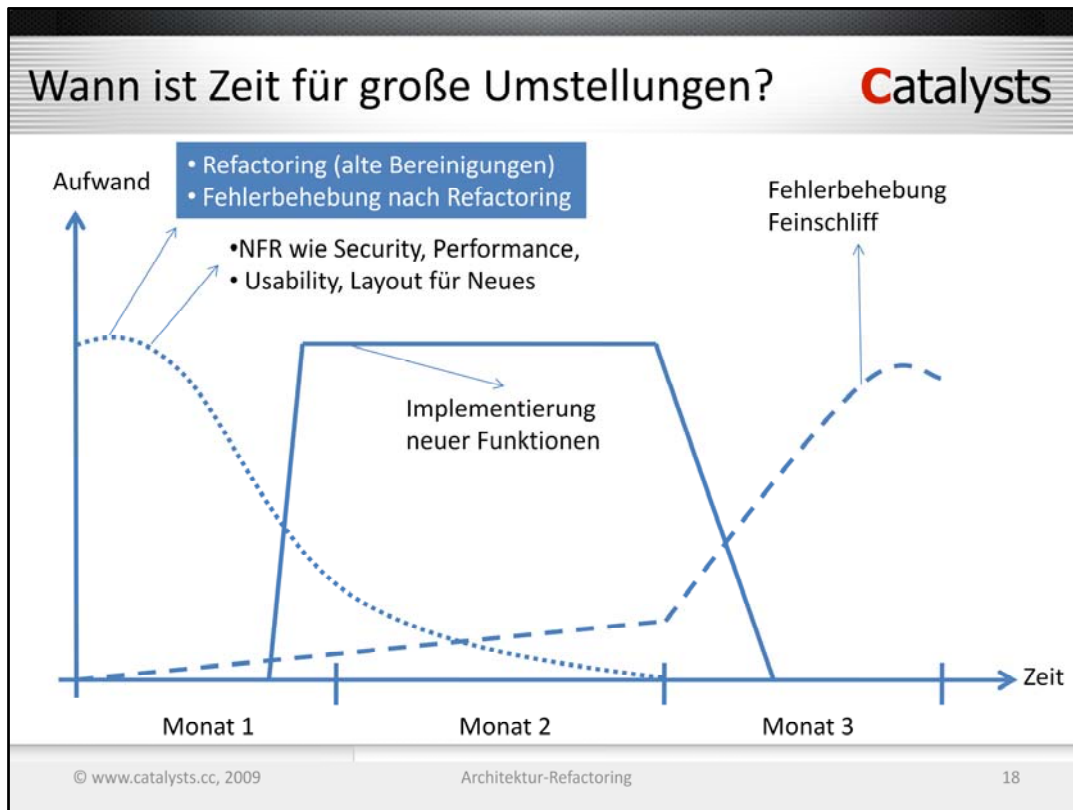
40.000 Zeilen sind in den taskmind-Projekten weggefallen, in der Catalysts-Plattform darunter sind 20.000 Zeilen übrig geblieben, 20.000 Zeilen fielen weg.

Im Mai 2009 gab's eine zweite große Umstellung, wo durch das Generalisierung von Aufgaben, Terminen und Dateien gut 40.000 Zeilen weggefallen sind.

Mitte Juni sind etwa 15.000 Zeilen in taskmind weggefallen, wo etwa 5.000 Zeilen in der Catalysts-Plattform dazu gekommen sind.

Wenn man diese großen Umstellungen nicht regelmäßig macht, veraltet die Architektur und wird so starr, dass sie Workarounds regelrecht BRAUCHT.





Die automatischen Architekturprüfungen führen dazu, dass Verletzungen gleich einmal aufgedeckt werden – und das ist gut so.

Ohne die automatischen Architekturprüfungen würden wir uns größere Architekturumstellungen wahrscheinlich auch nicht trauen.

Die Frage ist jetzt noch, WANN man größere Umstellungen prinzipiell machen möchte.


Aus unserer Erfahrung macht man größere Umstellungen am besten, wenn man nicht allzu sehr unter Druck steht.

Bei einem dreimonatigen Release-Zyklus würde man große Umstellungen somit eher im ersten Monat machen als im dritten, damit man zum Schluss ausreichend Zeit für die Fehlerbehebung und den Feinschliff hat.


Eclipse: im „Finish“ keine großen Änderungen mehr

Catalysts

How Is the Development Done?



- **release cycle 6 – 15 months**
 - rolling – 4-13 months
 - finish – 1 month
 - decompression – 1 month



Laser Summer School 2004

© www.catalysts.cc, 2009 Architektur-Refactoring 19

Sehr ähnlich hat uns das vor ein paar Jahren Erich Gamma erzählt, dass beim Eclipse-Release-Zyklus ein Monat vor der Release – quasi im „Finish“ keine großen Änderungen mehr erlaubt sind.

In dieser Endphase sind allerdings Workarounds explizit erlaubt.

Wenn die Release draußen ist, gehen viele Leute einmal auf Urlaub oder erholen sich von der Anstrengung.

In der Dekompressions-Phase bzw. kurz danach müssen dann die Workarounds wieder beseitigt werden. Hier können dann auch Architektur-Umstellungen erfolgen.

Zusammenfassung Catalysts

- Architektur-Refactoring am liebsten, wenn es das System noch gar nicht gibt
- Später mit dem Sicherheitsnetz der automatischen Architekturprüfungen
- Große Refactorings nur, wenn man nicht unter Druck steht
 - Eher am Anfang einer Release als am Ende
 - Eher am Anfang der Iteration als am Ende

© www.catalysts.cc, 2009Architektur-Refactoring20

Zusammenfassend heißt das:

Am liebsten ist es mir, wenn ich die Architektur umstelle, wenn es das System noch gar nicht gibt.

Ein Flipchart ist schneller umgezeichnet als Hunderttausende Zeilen Source Code neu strukturiert sind.

Später möchte man, dass die Architektur beschrieben ist, dass es Regeln gibt, die automatisch überprüft werden können.

Dadurch fallen Abweichungen und Regelverstöße sofort auf.

Man kann entweder die Architektur umdefinieren oder den Fehler beheben.

Und große Refactorings wird man nur machen wollen, wenn man nicht sehr unter Druck steht, weil ja doch immer was schief gehen kann.

Das heißt wohl eher am Anfang einer Release als am Ende.


Und weil die Welt fraktal ist, trifft das Ganze natürlich auch im kleineren Maßstab zu.

Auch innerhalb einer Ein-Wochen-Iteration wird man größere Umstellungen lieber am ersten als am letzten Tag machen.

Ausblick

Catalysts

- 6. November: Was bringt TDD wirklich?
- 20. November: Die Catalysts Platform
- 4. Dezember: PureMVC
- 18. Dezember: Server-Push über mehrere Plattformen hinweg



Was bringt mir Catalysts sonst noch?

Werbe-Trailer

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