Warum ist die einleuchtende Idee "Trennung von Fachlichkeit und Technik" in der Praxis so schwer umzusetzen? Wie bekommen Sie endlich die "Separation of Concerns", die Ihnen seit 40 Jahren versprochen wird?

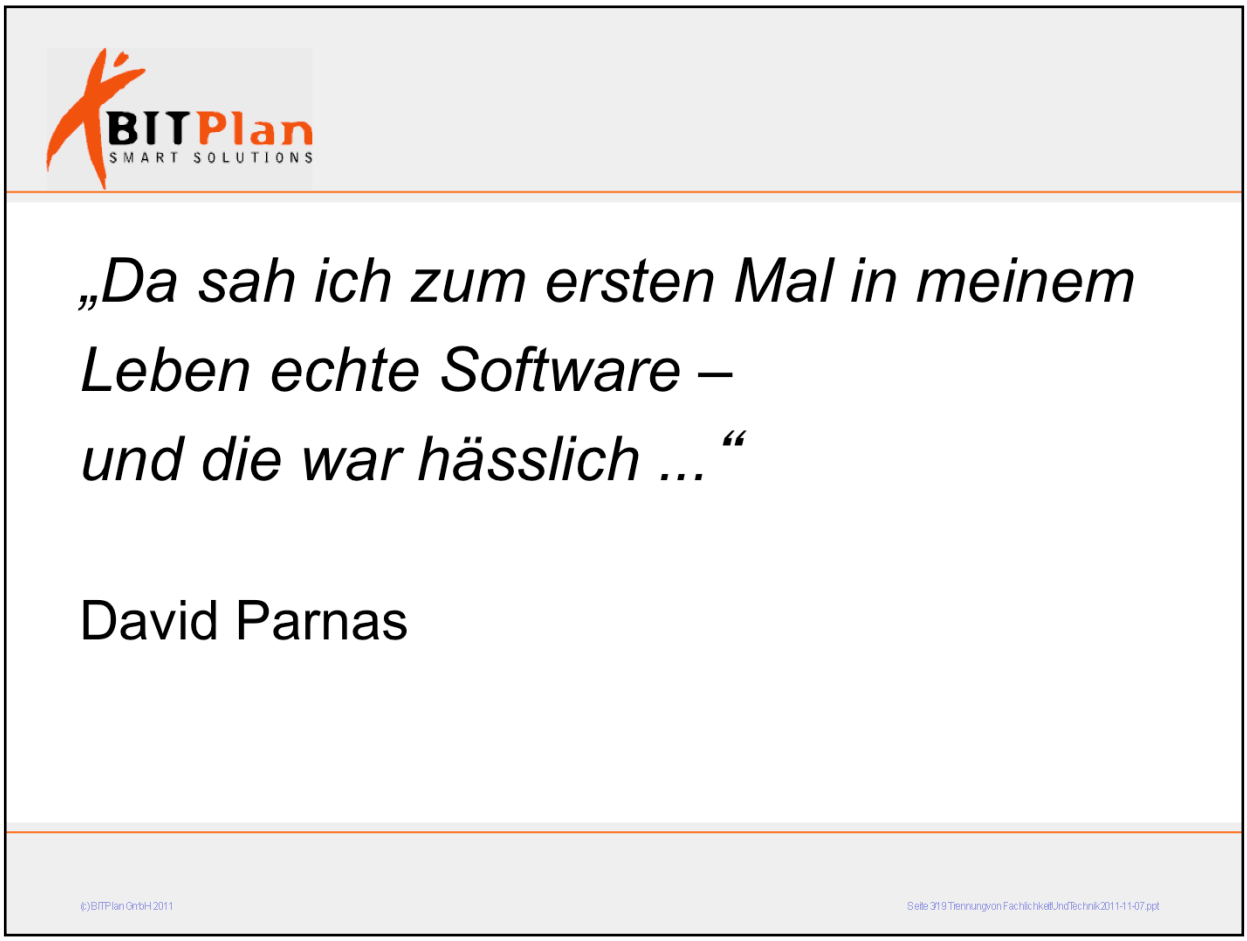
Lernen Sie "Architekturdrache", "Blutgruppe" und "Y-Prinzip" kennen.

Erfahren Sie wie mit zunehmender Größe und Komplexität der Vorteil des Trennungsvorgehens außerordentlich steigt. Sehen Sie, wie in konkreten Projektbeispielen die "Reverse-Y"-Analyse zeigt, wo der größte Nutzen liegt.

Geschäftsführender Gesellschafter BITPlan GmbH

Wolfgang Fahl ist Diplom-Informatiker und Geschäftsführender Gesellschafter der BITPlan GmbH. Er ist seit 20 Jahren im Software-Engineering tätig.

Softwarequalität liegt ihm besonders am Herzen, seit dem er Software für lebenserhaltende Systeme mitentwickelt hat. Seit 1996 ist Wolfgang Fahl als Berater und Unternehmer aktiv um Organisationen zu unterstützen, Software so zu entwickeln, dass das Ergebnis "passt".



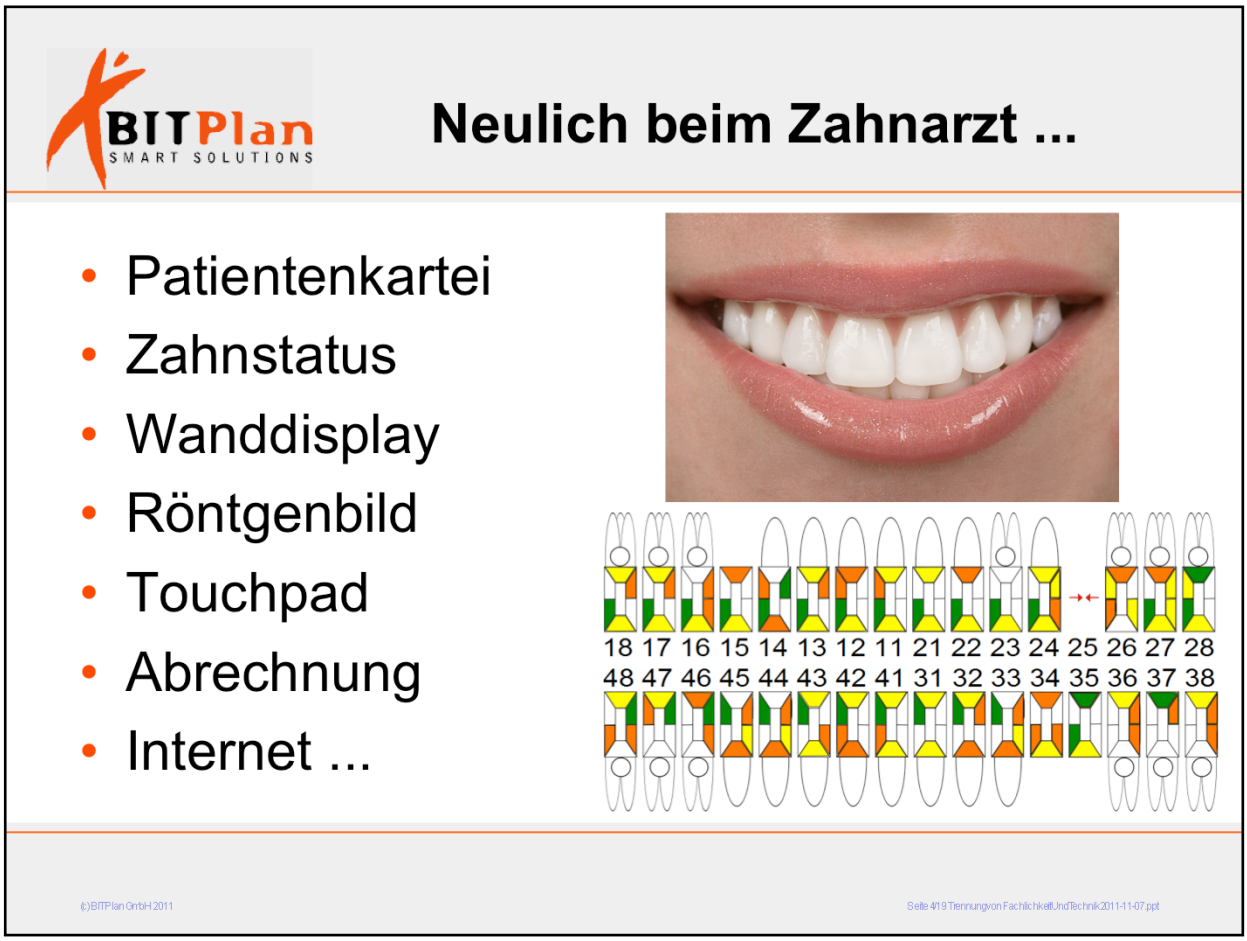
Parnas ist durch die Konfrontation mit hässlicher Software motiviert worden nach Gestaltungsprinzipien zu suchen, die Software „schöner“ macht. Zunächst war dabei gar nicht klar, dass dabei nicht nur eine ästhetische Verbesserung herauskommen würde. Der Nutzen von systematisch nach dem Prinzip der Trennung von Fachlichkeit und Technik entwickelter Software liegt nicht nur in einer hohen Einsparung von Aufwand. Die Software ist verständlicher und viel leichter zu ändern.

In diesem Vortrag geht es darum was Separation of Concerns bzgl. Fachlichkeit und Technik ist, mit welche Mitteln sie erreicht werden kann und was die Kriterien für die Anwendung sind.

Anhand des Beispiels einer Zahnarztsoftware werden Architekturdrache, Software-Blutgruppen und Y-Prinzip vorgestellt. Mit einer Formel wird der Vorteil der Trennung von Fachlichkeit und Technik gegenüber der Kombination aufgezeigt. Anschließend wird die Kosten-Nutzen Betrachtung mit Bezug zur allgemeinen Entwicklung der Kostenfaktoren in der Softwareentwicklung durchgeführt. An einigen konkreten Projektbeispielen zeigt sich, welche Entscheidungen sich aus unterschiedlichen Ausgangssituationen ergeben.

Mit diesem Wissen sollten Sie motiviert sein, Ihre eigenen Softwareprojekte darauf hin zu analysieren, wo sie bzgl. der Trennung von Fachlichkeit und Technik stehen und welches Gewicht Sie diesem Thema in Zukunft geben wollen.

Zum Schluss gibt es Gelegenheit zur gemeinsamen Diskussion.



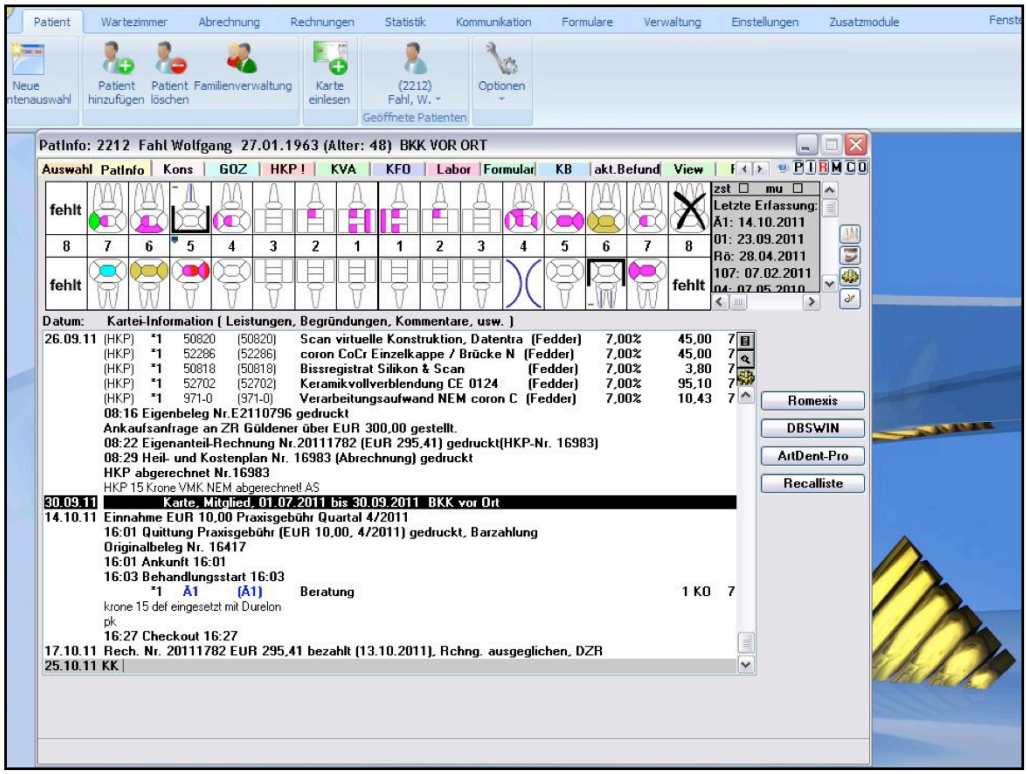
Mein letzter Zahnarztbesuch war erfreulicher als die drei Termine davor. Es musste nur noch einmal überprüft werden, dass die Krone sitzt und sie dann richtig einzementiert werden. So hatte ich Zeit mit meinem Zahnarzt ein nettes Gespräch über seine Software zu führen.

Er hat da ja so ein schönes großes Display an der Wand hängen wo ich auch selbst gucken kann wie es meinen Zähnen geht ☹

Nun hat mein Zahnarzt sich neulich ein iPad gekauft und meinte er fände es ja nun echt schick, wenn er das auch in seiner Praxis benutzen könnte. Er hat beim Hersteller seiner Software nachgefragt was denn da geplant sei und die sagten ihm da gäbe es schon eine nette App. Die hat mein Zahnarzt gleich runtergeladen und Geld dafür bezahlt. Anschließend war er ziemlich enttäuscht, weil die App ganz anders aufgebaut war als die eigentliche Patientensoftware. Der Zahnstatus war z.B. nicht nur optisch anders dargestellt – es fehlten auch wichtige Angaben. Der Hersteller meinte es wäre ja nur die erste Version und in wenigen Monaten wäre man ja ein Stück weiter ...

Solche oder ähnliche Geschichten über die Differenz zweier Softwaresysteme die das gleiche Thema behandeln und vom gleichen Hersteller stammen und „lediglich“ auf unterschiedlichen Systemen laufen kennen Sie sicher auch. Anhand des Zahnarztbeispieles will dieser Vortrag betrachten, was es mit der Separation of Concerns insbesondere der Trennung von Fachlichkeit und Technik auf sich hat.

Softwareentwicklung die passt

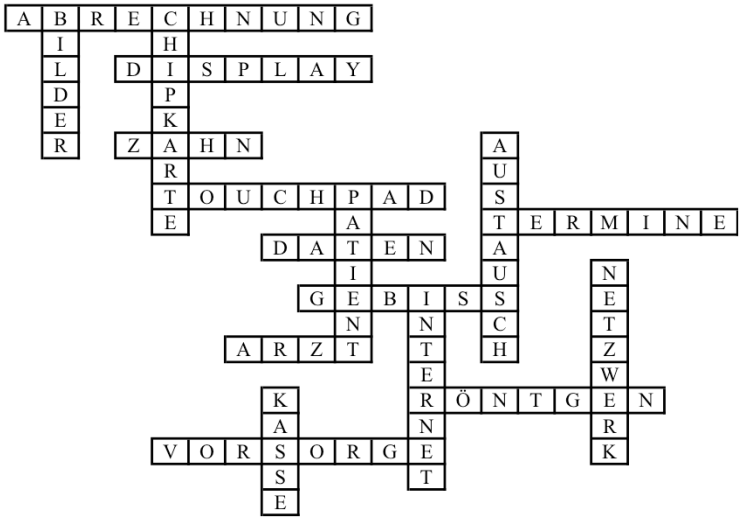


Rätselhafte Anforderungen

Anforderungen wirken

- rätselhaft
- komplex
- untereinander stark verwoben
- ungegliedert
- schwer analysierbar

wenn sie nicht nach Fachlichkeit und Technik getrennt sind

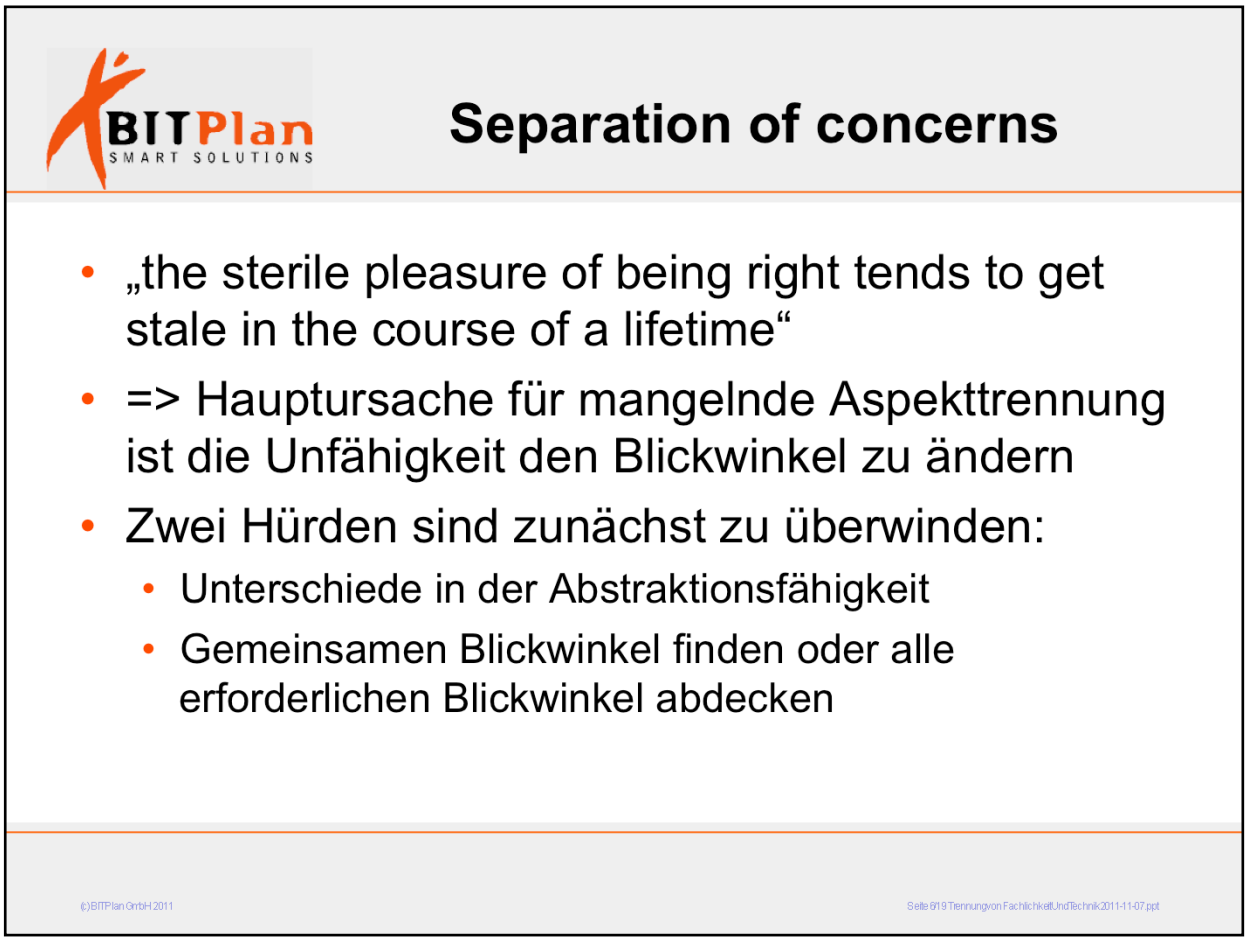


Schon bei der Anforderungsaufnahme ist die Gliederung in technische und fachliche Aspekte einer Lösung sinnvoll. Die Frage ist nach welchen Kriterien die Trennung erfolgen soll.

Wann ist ein Thema technisch? Wann ist es fachlich? Was ist mit gemischten Themen? Wie erfolgt die Bestimmung in welches „Töpfchen“ eine Anforderung gelegt werden soll?

Mit Intuition und Erfahrung ist es leicht, die Einteilung richtig vorzunehmen. Leitlinie ist, dass fachliche Anforderungen aus dem inhaltlichen Themenbereich des Auftraggebers kommen. Technische Themen gehören zur Umsetzung. So sind „Zahn“, „Kasse“ , „Arzt“ und „Patient“ der fachlichen Seite zuzuordnen. „Touchpad“, „Display“ und „Netzwerk“ gehören zur technischen Seite. Bei „Röntgen“ und „Bilder“ ist es nicht sofort klar was fachlich und was technisch ist. Hier kommt es darauf an, zu hinterfragen worauf sich die Begriffe beziehen und welche Blickwinkel gemeint sind.

Auf diesen Wechsel der Blickwinkel kommt es bei der „Separation of Concerns“, wie Dijkstra sie 1974 formuliert hat, an.



Dijkstra spricht davon, dass es natürlich und intelligent ist, sich den verschiedenen Aspekten eines Themas separat von einander zu nähern. Die Erhöhung der Flexibilität im Denken und Lösen von Problemen beginnt für ihn dabei mit dem Akzeptieren, dass es eine andere, bessere und ggf. schon existierende Lösung geben könnte.

In der Ausbildung von Softwarearchitekten hat sich gezeigt, dass die Zusammenarbeit von Menschen mit hohem Abstraktionsvermögen (etwas was von Softwarearchitekten erwartet wird) mit denjenigen, die nicht abstrahieren können oder wollen, schwierig ist. Diese Hürde kann erfolgreich genommen werden, indem konkrete Beispiele und Szenarien diskutiert werden und erst dann der Abstraktionsschritt erfolgt.

Die Einigung auf einen gemeinsamen Blickwinkel kann schwierig sein, selbst wenn es sich um einen scheinbar selbstverständlichen (natürlichen) Blickwinkel handet. Es kann notwendig sein, alle Beteiligten von Ihrem Blickwinkel “abzuholen”. Dies ist eine wichtige Aufgabe und soziale Fähigkeit des Softwarearchitekten.

Softwareprojekte, in denen die Probleme zu früh durch die “Lösungsbrille” betrachtet werden tendieren eher dazu einen Mangel an Separation of Concerns zu zeigen.

Siehe: Edsger W.Dijkstra;On the role of scientific thought;1974; http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF

aus:

Edsger W. Dijkstra, Selected Writings on Computing: A Personal Perspective, Springer-Verlag, 1982.

Struktur der Zahnarztsoftware

Sourcecode von „Tooth.cs“:

```
///<summary>Takes a user entered string
/// and validates/formats it for the database.
/// Throws an ApplicationException
/// if any formatting errors.
/// User string can contain
/// spaces, dashes, and commas, too.
/// </summary>
    public static string FormatRangeForDb
    (string toothNumbers){
      if(toothNumbers==null) {
        return "";
      }
      //remove all spaces
      toothNumbers=
        toothNumbers.Replace(" ","");
      if(toothNumbers=="") {
        return "";
      }
      //some items will contain dashes
      string[] toothArray=toothNumbers.Split(',');
```

Klasse „Patient“:

So oder ähnlich wie hier als Sourcecode und UML Modell gezeigt wird die Zahnarztsoftware „hinter den Kulissen“ aussehen. Die Ersteller der Software sind fest davon überzeugt, dass ihr Entwurf gut und richtig ist. Hält denn dieser Entwurf dem Kriterium „Separation of Concerns“ stand? Ist dieser Entwurf gut und richtig?

Eine ganze Reihe von Softwarearchitekten werden unverzüglich mit „schlecht und falsch“ antworten.

•Was hat „FormatRangeForDB“ als Funktion in „Tooth.cs“ zu suchen?

•Wird „patienanzeigen“ in der Klasse „Patient“ umgesetzt oder handelt es sich wenigstens um eine Delegation?

•Die Anwesenheit der Attribute „letzteScreenPositionX“ und „letzteScreenPositionY“ lässt darauf schließen, dass das technische Thema „GUI“ hier direkt in der fachlichen Klasse Patient umgesetzt werden soll.

Welche Ansätze gibt es eine solche Themenmischung zu vermeiden und welchen Vorteil hat dies?

Sourcecode aus:
https://70.90.133.65:23793/svn/opendental/opendental11.0/OpenDentBusiness/Logic/Tooth.cs

Architekturdrache

BISIT© Model / AXA Winterthur

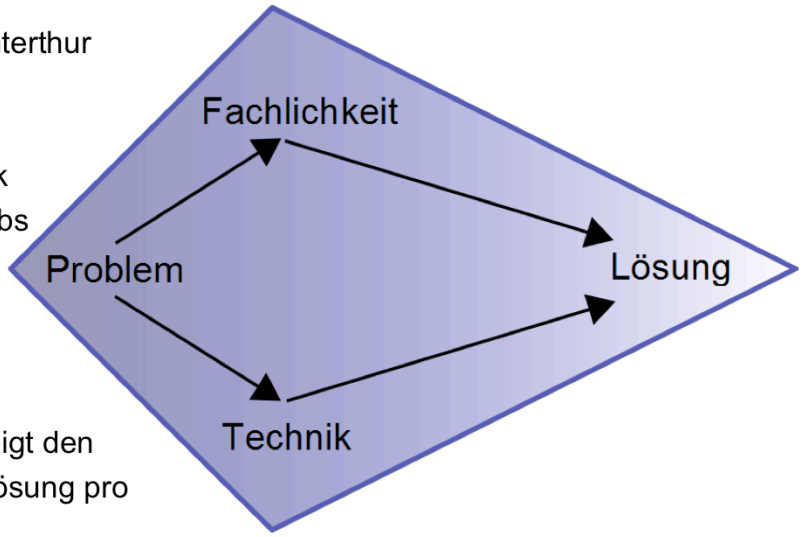
“Solution Architecture”:

- Fachliche Sicht
- Sicht der Informatik
- Sicht des IT-Betriebs

Zeitlich:

- Strategisch
- Taktisch
- Operational

Der Architekturdrache zeigt den Weg vom Problem zur Lösung pro Projekt.



Den Architekturdrachen habe ich 2005 bei der AXA Winterthur kennengelernt.

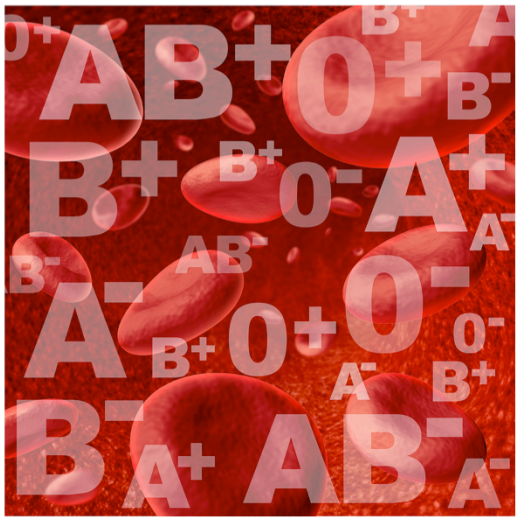
Fiorenzo Maletta hat mich beim Anblick der Grafik für das Y-Prinzip darauf hingewiesen, dass die Metapher des “Architekturdrachen” ebenfalls das Thema Trennung von Fachlichkeit und Technik aufgreift. Wenn man den Architekturdrachen wie oben quer abbildet dann kann man die Problem-Lösungsachse als zeitliche Achse sehen. Es stellt sich dann die Frage, in welcher Reihenfolge fachliche und technische Themen bearbeitet werden sollen und wann/wie häufig eine Integration der Ergebnisse sinnvoll ist.

Je getrennter die Verfolgung der Pfade Fachlichkeit und Technik erfolgt desto mehr ist es erforderlich, die Verbindungen zwischen beiden Seiten zu klären und Schnittstellen zu vereinbaren. Dazu ist es hilfreich, wenn die fachliche Seite mit einer “pseudotechnischen Sicht” arbeitet, welche vereinfacht die technischen Annahmen und Schnittstellen darstellt und die technische Seite arbeitet mit einer “pseudofachlichen Sicht”, welche die fachlichen Annahmen und Schnittstellen darstellt. Im Detail brauchen beide Sichten gar nicht vollständig oder korrekt sein. Die pseudofachliche Sicht sollte jedoch die aus technischer Sicht architekturrelevanten Punkte, wie z.B. das Mengengerüst, die Komplexität bzgl. Datenstrukturen und GUI und ähnliche Punkte wiedergeben. Die pseudotechnische Sicht sollte die aus fachlicher Sicht notwendigen techischen Themen wie z.B. GUI, Datenbank, Netzwerk, Security usw. so wiedergeben, dass die vorgesehenen Schnittstellen einfach und klar sind.

BITPlan SMART SOLUTIONS

Software-Blutgruppen

- Quasar (Siedersleben)
- Softwarekategorien
  - **A: Anwendung**
  - **T: Technik**
  - 0: Globale Basis
  - R: Repräsentationswechsel
- Vermeidung „unreiner“ AT-Software!



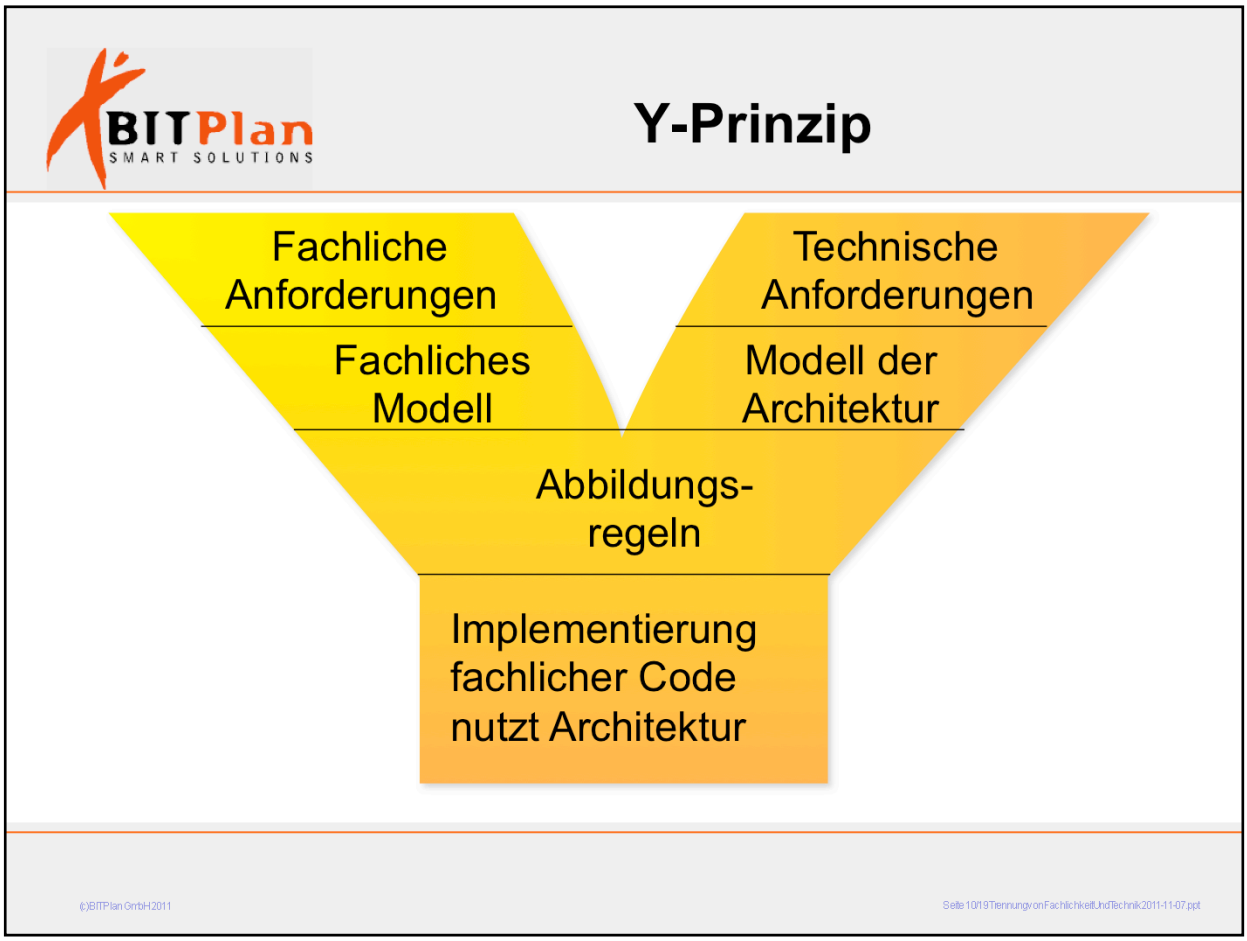
Der Gedanke der Blutgruppen sieht eine „saubere“ Trennung der Software in klare Kategorien vor. Die fachlichen Themen gehören zur Blutgruppe „A“ wie Anwendung. In unserem Zahnarztpraxisbeispiel sind „Zahnstatus“, „Patient“ und Arzt Themen der Blutgruppe A. Die technischen Themen gehören zur Blutgruppe „T“ wie Technik – d.h. im Beispiel sind „Netzwerk“, „Display“ und „Touchpad“ Themen der Software-Blutgruppe T.

0-Komponenten sind die globale Infrastruktur für beide Themen jedoch ohne A und/ oder T Inhalt nicht einzeln verwendbar. Im Beispiel ist „XML“ ein 0-Thema wenn es um die grundsätzliche Nutzung von XML-Strukturen geht. Die XML-Datenhaltung dagegen ist ein T-Thema. Die XML-Inhalte sind ein A-Thema.

R-Komponenten binden A und T Themen und sind die einzige erlaubte Mischung von A und T Themen. R steht für Repräsentation. Wenn also z.B. eine XML-Liste von Patienteninformationen in eine SQL-Datenbank zu überführen ist dann kann eine R-Software den Repräsentationswechsel von XML-Knoten zu SQL-Records übernehmen. R-Software hat in der Regel eine systematische Struktur und ist daher oft generierbar.

Siehe auch: Johannes Siedersleben;Moderne Softwarearchitektur;dpunkt.verlag 2004

Die weiteren Ansätze: Aspektorientierung, Modell Driven Architecture (MDA), DRY (Do not repeat yourself), usw. werden in diesem Vortrag nicht vertieft.



Das Y-Prinzip ist im Jahr 2001 bei BITPlan entstanden und basiert auf der Objects 9000® Idee von Martin Rösch. Dieses Prinzip ist in einer Reihe von Projekten angewendet worden, die plattformübergreifende Lösungen erfordern.

Das Tempo und der Rhythmus der Änderungen der Anforderungen unterscheidet sich für fachliche und technische Anforderungen. Es gibt Anwendungsfelder in denen die fachlichen Anforderungen sich über Jahrzehnte kaum ändern – insbesondere wenn es inhaltlich um Verträge mit jahrzehntelangen Laufzeiten geht wie bei Lebensversicherungen und Darlehen. Auch bei Zahnärzten ist der Kern der Anforderungen an die Patientensoftware ziemlich stabil – der Zahnstatus mit dem Zahnschema der Fédération Dentaire Internationale ist ein Beispiel für eine stabile fachliche Anforderung.

Andererseits ändern sich die fachlichen Vorgaben für die Abrechnung im Gesundheitswesen laufend und oft sogar mit einer Frequenz von mehreren Änderungen pro Jahr.

Der Gesetzgeber lässt sich halt ständig etwas Neues einfallen ☺

Im technischen Bereich werden durch die bunte Welt der Computer, des Internets von Web 2.0 und Social Media ständig neue Wünsche geweckt. Die Änderung der technischen Anforderungen ist davon abhängig wie schnell diese Wünsche befriedigt werden sollen und können. Zahnärzte gelten zwar als zahlungskräftig sind aber auch konservativ ☺

Das Y-Prinzip ermöglicht es das Tempo der Änderungen auf der fachlichen und technischen Seite unabhängig voneinander zu bestimmen.

BITPlan SMART SOLUTIONS

„Ausmultiplizieren“

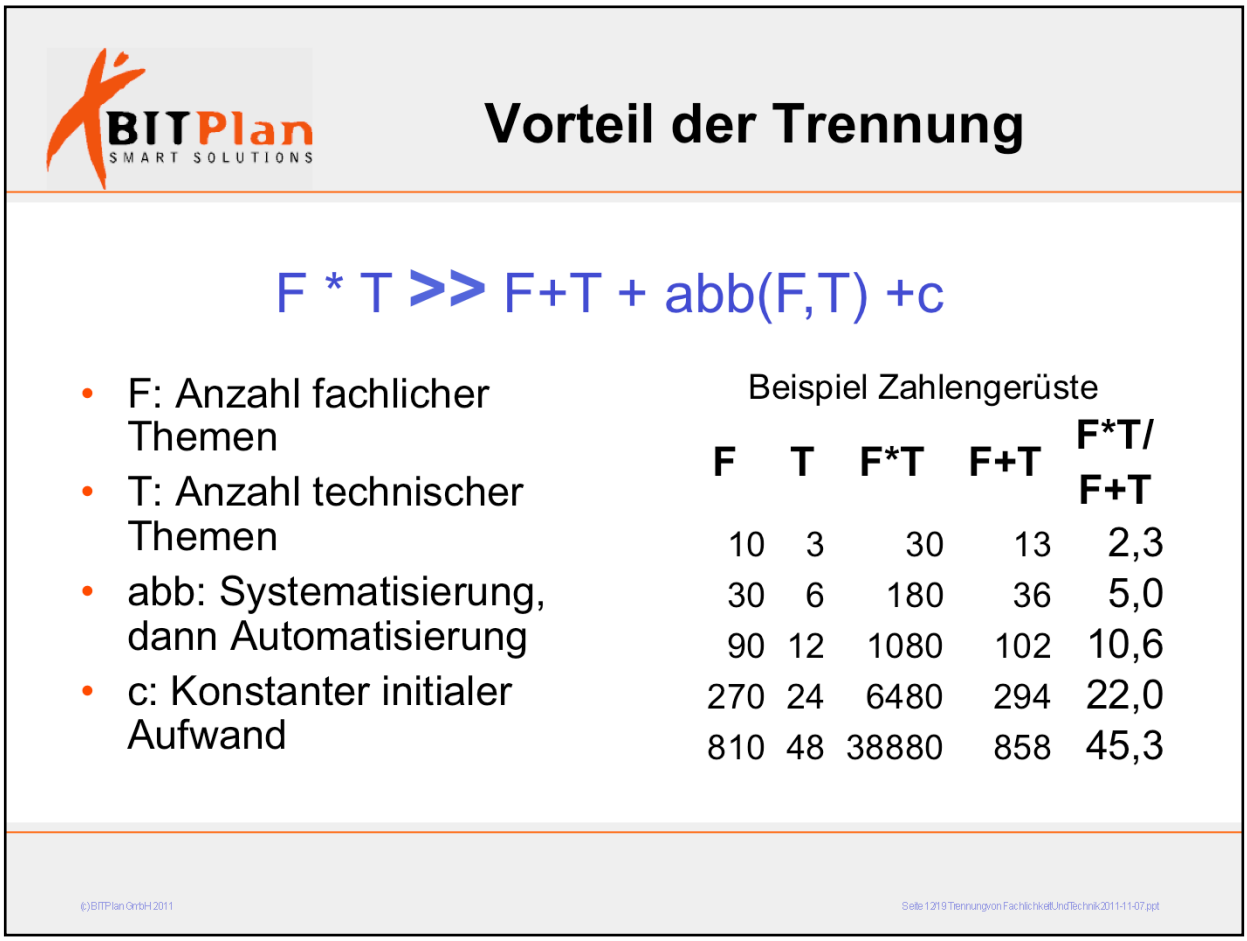
Fachlichkeit	Technik: GUI – Katalog	GUI – Formular	GUI – Bild	Datenhaltung – Tabellen	Datenhaltung – DAO	Transport XML-Austausch	...
Patient	Patienten-katalog	Patienten-formular	Patienten-bild	Patienten-tabelle	Patienten-DAO	Patient-XML	
Arzt	Arzt-katalog	Arzt-formular	Arzt-bild	Arzt-tabelle	Arzt-DAO	Arzt-XML	
Zahnstatus	Zahnstatus-katalog	Zahnstatus-formular	Zahnstatus-bild	Zahnstatus-tabelle	Zahnstatus-DAO	Zahnstatus-XML	
Zahn	Zahn-katalog	Zahn-formular	Zahn-bild	Zahn-tabelle	Zahn-DAO	Zahn-XML	
Abrechnung	Abrechnungs-katalog	Abrechnungs-formular		Abrechnungs-tabelle	Abrechnungs-DAO	Abrechnung-XML	
Kasse	Kassen-katalog	Kassen-formular		Kassen-tabelle	Kasse-DAO	Kasse-XML	
...							

Die obige Matrix zeigt am Beispiel der Zahnarztsoftware wie sich aus der Trennung von fachlichen und technischen Themen die Kombinationen „ausmultiplizieren“ lassen. Fast alle Kombinationen sind in diesem Fall sinnvoll und anwendbar. Insbesondere ergibt sich für praktisch alle Zeilen und Spalten eine systematische Abbildung. Somit lassen sich Regeln aufstellen, wie die Verbindung und von Fachlichkeit und Technik erfolgen kann. Je klarer diese Regeln formuliert werden können umso einfacher ist später auch die Automatisierung der Abbildung zu erreichen. Beispiele für Regeln sind z.B.:

- „Alle Eigenschaften eines fachlichen Themas sind im zugehörigen GUI-Formular anzuzeigen.“
- „Alle Eigenschaften eines fachlichen Themas sind in der zugehörigen Datenhaltungsstabelle und im Data Access Objekt sowie dem XML Austausch abzulegen. In der Tabelle als Spalten, im Data Access Objekt als Attribute und im XML Austausch als Knoten.“

Die Regelbeispiele zeigen bereits die Grenzen der Automatisierbarkeit auf. Während die Datenhaltung noch gut automatisierbar ist wird im GUI Bereich die Regel alleine noch nicht zu einer ergonomischen - geschweige denn ästhetischen – Darstellung führen.

Insgesamt stellt sich die Frage des Kosten- Nutzenverhältnisses für die Trennung.



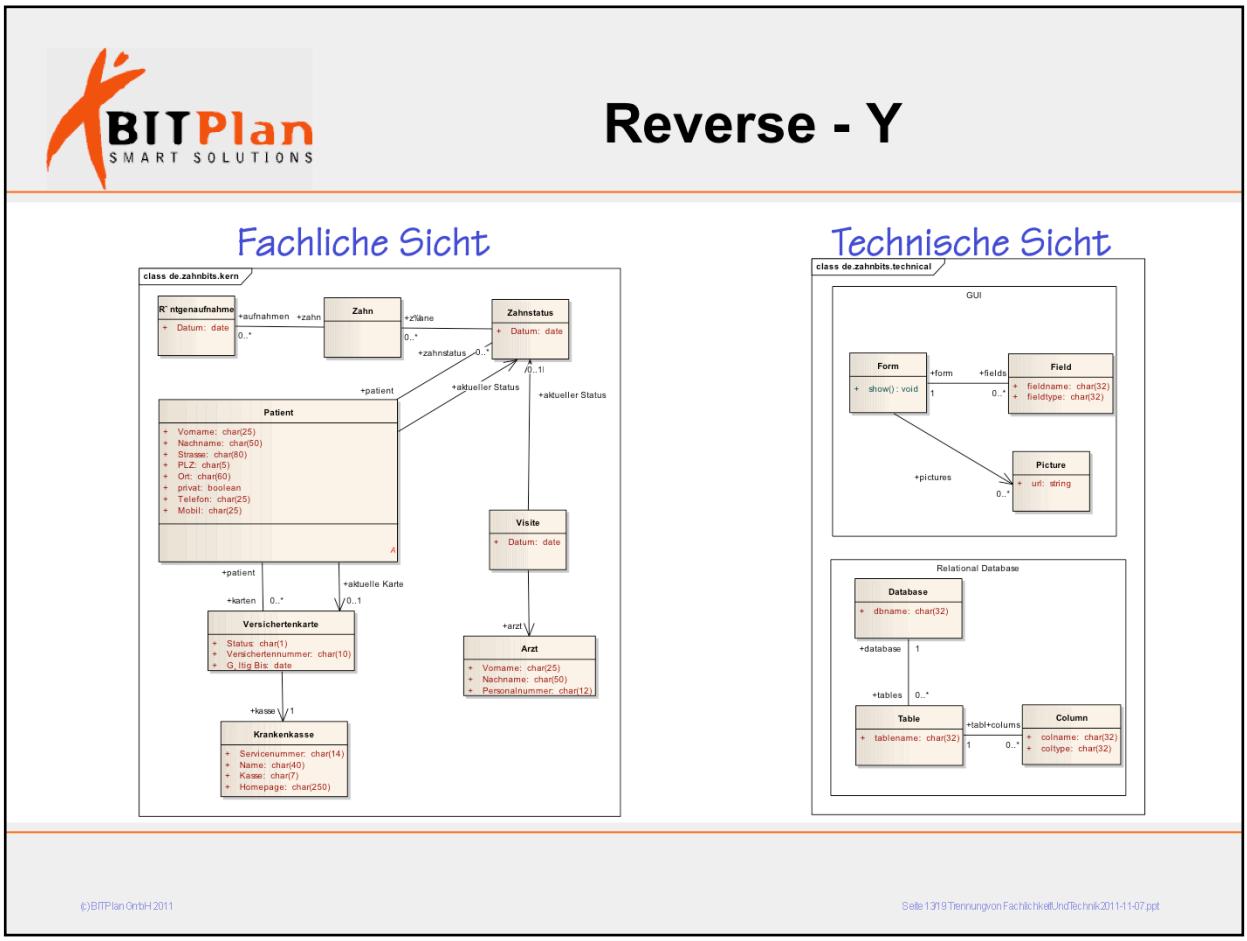
Die obige Formel zeigt, dass die theoretisch möglichen Einsparungen bei der systematischen Trennung von Fachlichkeit und Technik sehr hoch sein können.

Bei 320 Fachthemen und 40 technischen Themen gibt es 12800 Mögliche Kombinationen. Fachthemen und technische Themen getrennt zu betrachten führt zu 360 Arbeitspaketen. Das Verhältnis aller möglichen Kombinationen zu der Anzahl Arbeitspakete beträgt 35,6:1.

Voraussetzung für eine sinnvolle Einsparung ist eine hohe Anzahl von fachlichen und technischen Themen. Zusätzlich muss die Matrix „voll besetzt sein“. D.h. damit die Formel gilt muss annähernd immer die sinnvolle Verbindung von Fachlichkeit und Technik gegeben sein.

Der Abbildungsaufwand abb(F,T) muss gering genug sein und der initiale Aufwand c darf nicht zu hoch sein. Genau an diesen beiden Punkten scheitert in der Praxis jedoch häufig der konsequente Einsatz der Trennungstechnik mit all den sich daraus ergebenden Vorteilen für die Wiederverwendbarkeit, Testbarkeit und Automatisierung. Selbst wenn der Aufwand sich für ein Projekt bzw. für eine Organisation mit mehreren Projekten rechnet ist ein häufig anzutreffendes Hindernis die Zeitverzögerung die erfolgt, wenn erst einmal investiert werden muss in Wissen und Infrastruktur.

Ein weiteres Folgeproblem ist die Änderung der Verteilung des Wissens. Typisch für Innovationen die Arbeitskraft einsparen ist, das wenige Spezialisten benötigt werden. Das ist auch bei der konsequenten Nutzung von Trennung von Fachlichkeit und Technik mit dem Ziel der Systematisierung und Automatisierung so. Im Extremfall gibt es dann in kleinen Organisationen jedoch nur noch einen oder zwei Spezialisten und damit entsteht eine Risikosituation. Ein hoher Anteil der Softwareentwicklung ist nun von der Verfügbarkeit und Fähigkeit von wenigen Spezialisten abhängig.



Die obigen UML Sichten zeigen eine „saubere“ Trennung von fachlichen und technischen Themen für das Zahnarztsoftwarebeispiel.

Die konzeptionelle Durchgängigkeit von der Anforderungsaufnahme bis hin zur Implementierung zu erreichen ist nicht einfach. Es macht die Softwareentwicklung deutlich leichter und verständlicher, wenn die Begriffe und Konzepte von der Anforderungsaufnahme bis in die Implementierung unverändert durchgehalten werden. Diesem Ziel widersprechen ggf. andere Ziele die ein Softwareprojekt hat.

Auf der fachlichen Seite ist es heutzutage häufiger üblich, dass ein Begriff wie „Patient“ tatsächlich auch in dieser Form als Tabellenamen, Klassenname, Name eines Formulars usw. auftritt und nicht umgewandelt werden muss in einen kryptischen Bezeichner wie „HPAT3750“ wie es noch vor 20-30 Jahren durchaus üblich war.

Auf der technischen Seite ist die Durchgängigkeit erheblich schwerer zu erreichen. Eine Vielzahl von vorhandenen Technologien ist beladen mit Begriffen, die von den Lieferanten geprägt werden. Leider passen diese Begriffe nicht immer zum technischen Inhalt der Software und häufig genug sind die Begriffe schlicht unverständlich bzw. nur als Jargon innerhalb der Gemeinde der Anwender der Technologie üblich.

Immer Trennen?

- Es kommt darauf an ...
- Es kommt vor allem auf das Kosten/ Nutzen Verhältnis an
- Heute wird anders entschieden als früher ...

Wenn der Hersteller der Zahnarztsoftware zu Beginn der Softwareentwicklung gewusst hätte, dass diese eines Tages auch auf einem Touchpad laufen soll, dann hätte er möglicherweise die Software anders gestaltet.

Die wirtschaftliche Motivation eine Software plattformunabhängig zu entwickeln hängt direkt von den Verbreitungszahlen ab, die sich auf den unterschiedlichen Plattformen realisieren lassen.

Nehmen wir an ein Entwicklermonat kostet 20.000 EUR und pro verkaufter Software lassen sich 100 EUR erzielen. 200 Softwarelizenzen finanzieren dann einen Entwicklermonat Aufwand. Der Anreiz ist groß den Aufwand nun in eine „Speziallösung“ zu stecken, das nachträgliche Ändern der eigentlichen PC-Software lohnt sich finanziell nicht unbedingt.

Anders sieht die Rechnung aus, wenn sie früher angestellt wird und langfristiger angelegt ist. Wenn die Zahnarztsoftware von vornherein als Produktfamilie geplant ist und die Verbreitung auf unterschiedlichen Plattformen zur Produktstrategie gehört dann ist es erforderlich auch die Software von vornherein so zu gestalten, dass der Plattformwechsel leicht fällt. Die konsequente Trennung von Fachlichkeit und Technik ist dann fast selbstverständlich.

Welchen Einfluss das Kostenverhältnis Computer/Entwickler auf diese Entscheidung hat schauen wir uns als Nächstes an.

Kostenentwicklung in der IT

Barry Boehm (1955):

"We're paying this computer 600 hundred dollars an hour and we're paying you two dollars an hour and I want you to act accordingly."

Barry Boehm berichtet von seinem ersten Arbeitstag 1955. Sein Chef sagte zu ihm:

"We're paying this computer 600 hundred dollars an hour and we're paying you two dollars an hour and I want you to act accordingly."

Siehe auch: Software Pioneers, with DVD, Springer 2001

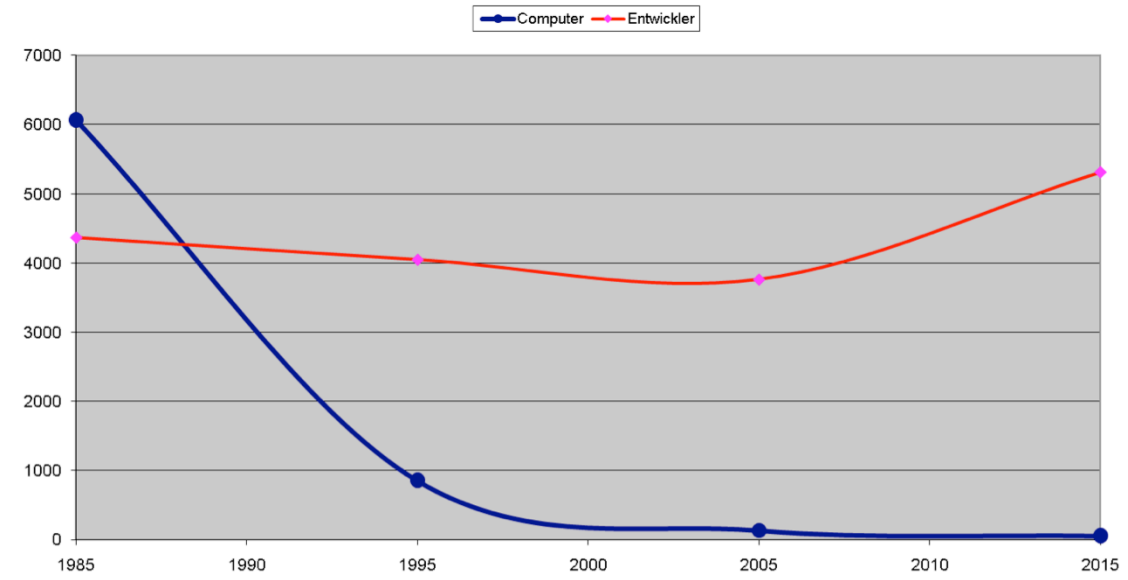
Wie war die Kostenentwicklung in der IT/Softwareentwicklung seit 1955?

Die Kurve auf der nächsten Seite zeigt die Entwicklung der Kosten pro Stunde für Computer bzw. Entwickler. Die Angaben sind aus dem Zitat von Barry Bohm mit Hilfe der amerikanischen Mindestlohntabelle extrapoliert. Annahme ist, dass Software-Entwickler über die Jahre immer etwa das 6,5 fache des amerikanischen Mindestlohns verdient haben.

Die Computerpreisentwicklung ist unter der Annahme abgeschätzt, dass ein Computer 9000 Stunden Nutzungszeit hat und beruht auf Preisschätzungen.

BITPlan SMART SOLUTIONS

Kostenentwicklung



War 1955 das Verhältnis Computerkosten zu Entwicklungskosten etwa 100:1 so ist das Verhältnis vermutlich schon bald 1:100.

Der „Tipping Point“ lag Ende der 80er Jahre bis dahin war die Computerstunde teurer als die Entwicklerstunde danach hat sich das Verhältnis umgekehrt.

Die dramatische Entwicklung um einen Faktor 10.000 hat gravierende Folgen auf die Entwurfsentscheidungen für Software. Die Software, die heute vielfach in Einsatz ist, wurde unter dem Aspekt optimiert, die Computerkosten zu minimieren. In der Regel hat heute jedoch der Entwicklungskostenaspekt Vorrang. Selbst hohe Vervielfältigungsfaktoren können diesen Effekt kaum noch mindern.

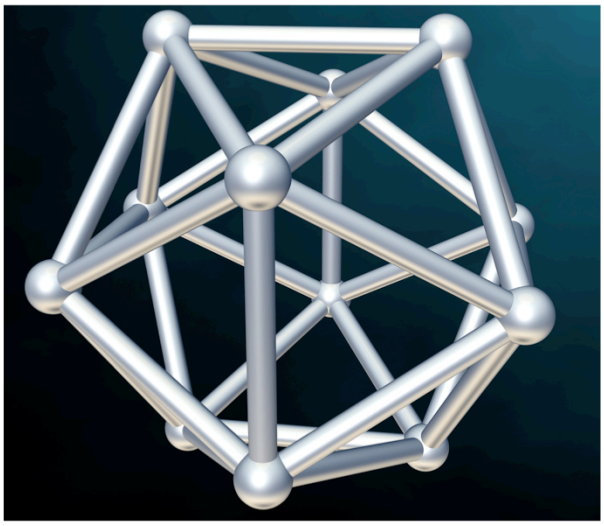
D.h. erst wenn eine Software häufig genug eingesetzt wird, sind die Entwicklungskosten wieder nachrangig.

Die Software so zu gestalten, dass die Entwicklungskosten gering sind, wird zunehmend wichtiger.

BITPlan SMART SOLUTIONS

Projektbeispiele

- Tilgungsplan
- Datenbank zur Untersuchung von Radiotherapiespätfolgen (GENEPI)
- Migration eines Tourenplanungssystems und Einführung Onboard Computer System
- Optimierung Paketversand
- Produktfamilie Embedded Software
- ...



Für die oben genannten Projektbeispiele ergeben sich folgende Bewertungen zur Nützlichkeit der systematischen Anwendung des Y-Prinzips:

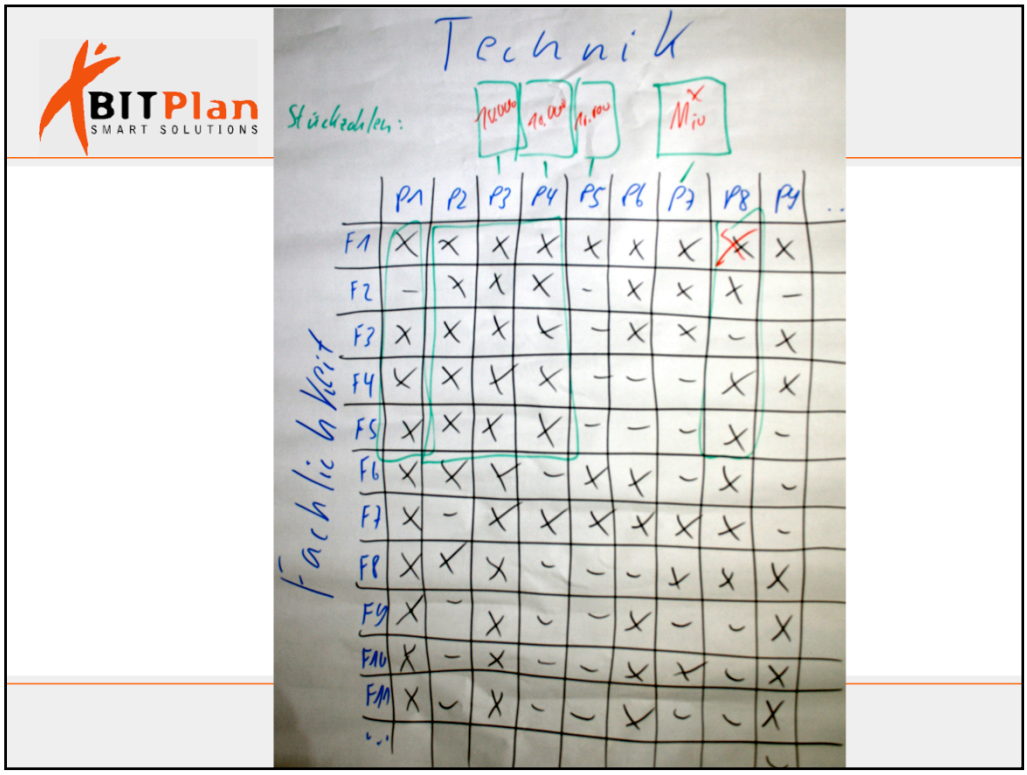
Im Tilgungsplanprojekt hat sich der Einsatz gelohnt, weil in der langfristigen Betrachtung vor allem Design for Testability im Vordergrund gestanden hat und die Wartbarkeit der Software über mehr als 10 Jahre deutlich verbessert wurde.
In GENEPI hat sich der Ansatz gelohnt, weil die Nutzung von UML2PHP zu geringen Gesamtkosten und einer schnelleren Entwicklung geführt hat.

Die Migration des Tourenplanungssystems hat profitiert da es vieler beteiligte Länder gab. Die Internationalisierung wurde als Technikaspekt betrachtet und die Möglichkeit des Wechsels des Tourenplanungssystems bei gleichzeitiger Einführung der Onboard Computer Systeme ermöglicht.

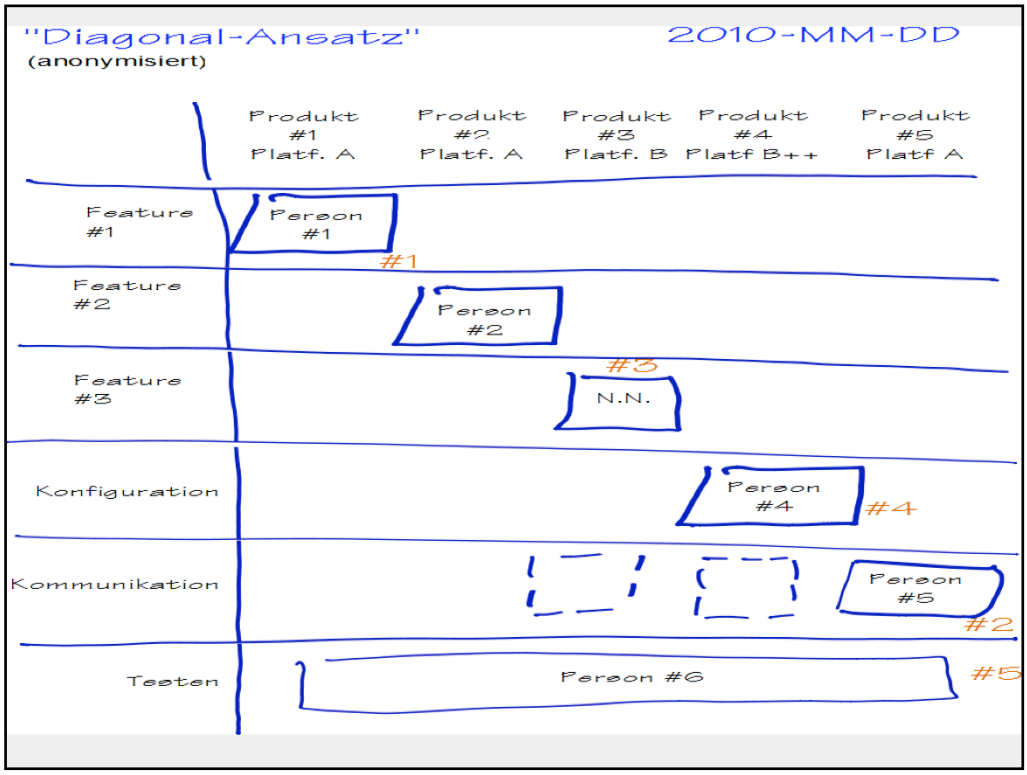
Die Optimierung des Paketversandes für ca. 1 Milliarde Pakete pro Jahr war ein Beispiel bei dem die Reverse-Y Analyse gezeigt hat, dass der Nutzen einer strikten Trennung von Fachlichkeit und Technik im Verhältnis zu den anderen Themen des Projektes nur gering ist, da im Wesentlichen ein einzelner Aspekt zu optimieren war und es um den Ersatz existierender Lösungen ging.
Bei dem betrachteten Embedded Software Projekt ergab die Reverse-Y Analyse ein differenziertes Bild. Besonders in den Produktlinien mit geringen Stückzahlen macht sich die Einsparung von Entwicklungsaufwand bemerkbar. Entscheidender ist aber die Möglichkeit der Rekombinierbarkeit von “Features” für ähnliche Produkte um ein breiteres Spektrum an Lösungen anbieten zu können.

Im Zahnarztsoftwarebeispiel hatten wird die Betrachtung ja bzgl. der Stückzahlen für die Touchpad-Lösung ja bereits durchgeführt.

Softwareentwicklung die passt



Softwareentwicklung die passt



BITPlan SMART SOLUTIONS

Zusammenfassung

- Trennung von Fachlichkeit und Technik ist ein Schlüssel zum Projekterfolg
- Der Vorrang ergibt sich aus dem Kosten- Nutzenpotential
- Bei Produktfamilien und plattformübergreifenden Systemen ist der Ansatz besonders lohnend
- Erst systematisieren, dann automatisieren!

Bei der Entwicklung von Softwarearchitekturen geht es darum, eine geeignete Zerlegung der zu entwickelnden Software zu finden. Die Trennung von Fachlichkeit und Technik ist ein wichtiges Zerlegungsprinzip. Das Kosten- und Nutzenpotential darf nicht außer acht gelassen werden, denn es macht keinen Sinn mit missionarischem Eifer als Prinzipienreiter aufzutreten, wenn eine spezielle Projektsituation nun doch nicht den höchstmöglichen Kosten- Nutzeneffekt bietet.

Bei langfristig zu pflegenden Softwaresystemen ist es wichtig die gesamte Laufzeit in die Betracht zu ziehen.

In Situationen in denen Produktfamilien entwickelt werden gilt grundsätzlicher, dass ein aspektorientierte Ansatz der sich von der Trennung von Fachlichkeit und Technik leiten lässt sinnvoll ist.

Plattformübergreifende Entwicklung lässt sich deutlich leichter realisieren, wenn die technischen Aspekte gekapselt sind. Konsequenterweise sollte schon die Anforderungsaufnahme die technischen Aspekte möglichst getrennt berücksichtigen.

Die konzeptionelle Durchgängigkeit von der Anforderungsaufnahme bis zur Implementierung unter Beibehaltung von Trennung von Fachlichkeit und Technik zu erreichen ist in der Praxis nicht einfach. Kompromisse sind kaum vermeidbar.

Wichtig dabei ist möglichst den Vorteil der Systematisierung nicht zu verlieren. Denn Systematisierung ist die Voraussetzung von Automatisierung.

Die nötigen Einigungsprozesse für die Systematisierung gehören an den Anfang. Dann erst kann auf dieser Basis automatisiert werden.

Diskussion

BITPlan
SMART SOLUTIONS

www.BITPlan.com

Wolfgang.Fahl@BITPlan.com