Jenseits des Tellerrands, Teil 3: Warum es Softwareentwicklungsprojekte gibt

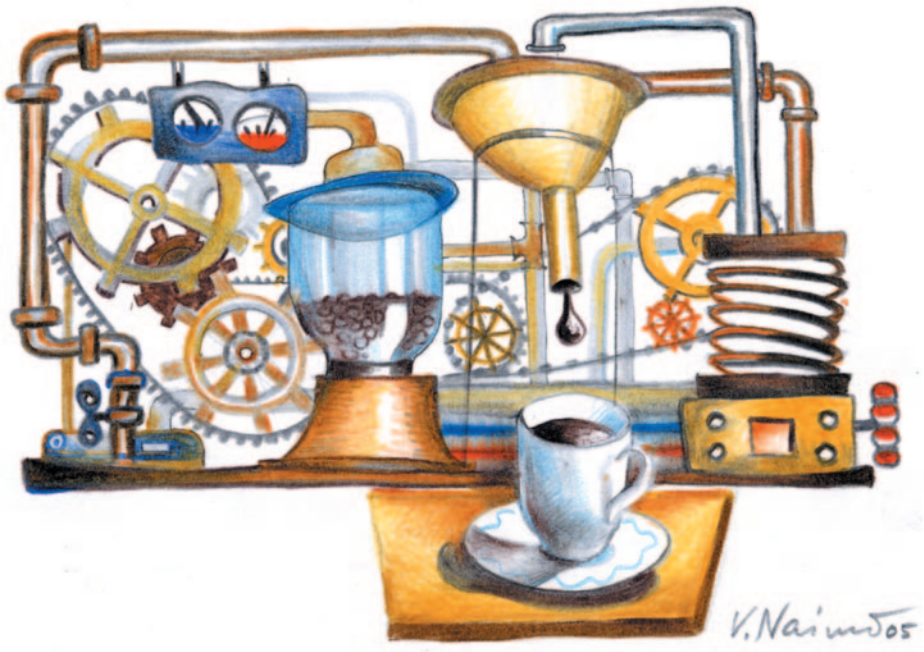
Ist die Technologie das geringere Problem?

VON MARTIN LIPPERT, STEFAN ROOCK UND HENNING WOLF

Speaker @ W-JAX

Wir Entwickler beschäftigen uns gerne mit handfester Technik, obwohl vermutlich jedem mit ein wenig Projekterfahrung klar ist, dass Projekte an ganz anderen Problemen scheitern. Tatsächlich beschleicht mich nach all den Jahren immer häufiger das Gefühl, dass ich meine Energie in etwas stecke, das sich treffend als „optimizing the idle loop" bezeichnen lässt. Die drei Autoren des folgenden Artikels gehen sogar noch einen Schritt weiter und vertreten die These, unsere Technikverliebtheit sei zwar verständlich, schade dem Projekterfolg jedoch mehr, als sie nütze. Ich kann ihnen aus vollem Herzen zustimmen.

JOHANNES LINK



Heute gibt es so viele gute Informationsquellen über Techniken, Technologien und Tools, dass niemand mehr sagen kann, er hätte nicht gewusst, wie man etwas technisch umsetzt. Zusätzlich gibt es nach unserer persönlichen Beobachtung zum Glück viele gut ausgebildete Entwickler, die ihr Handwerkszeug verstehen und sehr guten Code produzieren können. Sicherlich gibt es auch andere, aber selbst der guten Gruppe ist es nicht vergönnt, dass ihre Projekte zwangsläufig erfolgreich sind. Wir diskutieren im Folgenden die Ursachen und begründen, warum es viel wichtiger ist, sich in Entwicklungsprojekten mit den fachlichen Aspekten zu beschäftigen als mit diversen Technologien oder den neuesten APIs.

Warum werden viele Softwareentwicklungsprojekte überhaupt ins Leben gerufen? Da wollen Kunden (interne oder externe) softwaretechnische Unterstützung für ihre Probleme. Da versprechen sich Manager entscheidende Einsparungen. Da erhofft der Abteilungsleiter, eine organisatorische Änderung über ein neues Softwaresystem zu erreichen. Und Anwender wollen mehr Komfort und eine höhere Integration ihrer Aufgaben. Alle diese Leute wollen kein Java, kein Java EE, kein Swing, keine JSPs und nicht einmal Eclipse! Sie haben kein technisches Problem (außer vielleicht mit den Anwendungen, die unsere Vorgänger bei ihnen hinterlassen haben), sie haben ein *anwendungsfachliches* Problem. Dafür suchen sie nach ei-

ner Lösung. Wir Entwickler kommen ins Spiel, weil jemand vermutet, dass man das Problem softwaretechnisch lösen oder die Lösung mit Software unterstützt werden kann.

Es gibt auch noch andere Motive, die in Projektentscheidungen eine Rolle spielen: Manager wollen sich über Projekte profilieren. Projekte verleihen Wichtigkeit und laufende oder zumindest angefangene Projekte täuschen vor, es würde sich etwas (womöglich sogar in die richtige Richtung) bewegen. Diese Aspekte betrachten wir hier nicht detailliert, sie stellen aber für viele Projekte insofern ein reales Problem dar, als dass auch Entscheidungen während des Projektverlaufs dieser Logik und nicht einem Verständnis vom eigentlichen, eingangs genannten Ziel jedes Entwicklungsprojektes folgen.

Was passiert?

In vielen Projekten haben wir beobachtet, dass die größte Aufmerksamkeit zunächst der verwendeten Technologie geschenkt wird. Lange bevor das fachliche Problem verstanden ist, wird diskutiert, ob und vor allem welchen Application Server man benötigen wird, welche Sprache die geeigneten Hilfsmittel zur Verfügung stellt und welches Persistenz-Framework man wohl nehmen sollte (um nur einige der gängigen Diskussionen zu nennen). Die grundlegende technologische Architektur (SOA, 3-Tier, EJB oder Ähnliches) wird skizziert und es werden Prototypen für die unterschiedlichen technologischen Szenarien implementiert. Dann und wann kommen auch noch emotional vorbelastete Diskussionen um eine geeignete Entwicklungsumgebung hinzu. Im Rahmen dieser technologisch geprägten Diskussionen gerät das eigentliche Problem der Anwender schnell in den Hintergrund.

Ja, ja, dafür machen wir Analysen

Nun könnte man denken, dass wir genau deshalb eine Analyse der Probleme des Anwendungsbereichs machen oder sie von unseren Experten machen lassen. Die Analyse kriegt dann schon raus, was das Problem ist und mit ein bisschen Glück sogar gleich die Lösung als UML-Diagramm. Danach müssen dann nur noch die technischen Probleme gelöst werden.

So einfach ist es leider nicht und, zugegeben, so einfach machen es sich viele auch gar nicht. Das vermutlich schwierigste und damit gleichzeitig auch kritischste Problem jedes Entwicklungsprojektes besteht in der Analyse des Anwendungsproblems, der Konzeption der Lösung und einer geeigneten Umsetzung der Fachlichkeit in das Softwaresystem. Dabei geht es nicht darum, irgendwelche oder besonders viele Probleme zu finden, sondern das eine oder die wenigen essenziellen Probleme zu identifizieren, fachlich zu durchdringen und in einen geeigneten softwaretechnischen Entwurf zu überführen. Dazu ist es elementar notwendig, dass sich Entwickler so weit wie nur möglich in die Anwendungswelt der Kunden hineindenken. Es ist schon nicht wenig erwartet, dass hochgradig spezialisierte Entwickler immer wieder neue anwendungsfachliche Welten erschließen sollen. Es führt aber kein Weg daran vorbei.

Auch die Vermittlung der Konzepte durch einen Business-Analysten von einem Fachexperten an den Entwickler erspart es Entwicklern nicht, letzten Endes diese Konzepte zu durchdringen, um sie sinnvoll abzubilden. Business-Analysten können überdurchschnittlich gut Anwendern solche Konzepte entlocken, aber je weiter sie selbst von der Entwicklung entfernt

Jenseits des Tellerrands

Softwareentwicklung ist weit mehr als Programmierung und Programmierung ist weit mehr als das bloße Beherrschen von Programmiersprachen und Werkzeugen. Die Artikel dieser Serie möchten wichtige Themen, aktuelle Fragen und auch grundsätzliche Probleme aufgreifen, beleuchten und „die gängigen Antworten" immer wieder in Frage stellen. Subjektivität ist dabei keineswegs verpönt, sondern das notwendige Salz in der technischen Einheitssuppe.

Was bisher geschah und Sie zukünftig erwartet

Teil 1: Das nächste Java?
Teil 2: Typisierung in Java und darüber hinaus
Teil 3: Warum gibt es eigentlich Softwareentwicklungsprojekte?
Teil 4: Softwarearchitektur: eine kritische Bestandsaufnahme
Teil 5: Textuelle domänenspezifische Sprachen
Teil 6: Moderne System- und Abnahmetests

sind, umso schwieriger wird es, diese Konzepte an die Entwickler weiterzugeben.

Aber wozu dann überhaupt Technologie?

Die Technologien und Tools wenden wir nicht zum Selbstzweck an, sondern zum Nutzen unserer Projekte. Sie helfen uns, effizienter, effektiver, günstiger, schneller und besser zu werden. Sie verbessern das Verhalten der Software im Lebenszyklus, indem sie Wartbarkeit, Erweiterbarkeit und Verständlichkeit erhöhen. Technologien machen heute Lösungen möglich, über die man vor wenigen Jahren gar nicht ernsthaft nachdenken konnte: Anwender kooperieren über große Distanzen, sehen auf dasselbe Arbeitsergebnis, können im Medium Computer fast alle Arbeitsschritte erledigen. Systeme skalieren für gigantisch hohe Benutzerzahlen. Dieselbe Logik ist über verschiedene Frontends verfügbar. Das sind alles wichtige Errungenschaften der Technologie – aber …

Gewichtung beibehalten

Technologie ist „nur" Mittel zum Zweck. In Systemen, die relevante Mengen an Anwendungslogik abbilden, sollte also der Aufwand für die Realisierung der Fachlichkeit den Aufwand übersteigen, der in Technologien fließt. Das ist aber in den allerwenigsten Projekten der Fall.

Dafür sind zum einen die Entwickler verantwortlich und zum anderen die Technologien selbst. Auf der einen Seite kümmern sich viele Entwickler lieber um Technologien als um fachliche Konzepte. Die Kenntnis möglichst komplizierter Technologien erhöht den eigenen Marktwert und so muss jedes Projekt auch als Weiterbildungsmaßnahme herhalten. Außerdem fühlen sich Entwickler durch ihre Ausbildung natürlich eher im Bereich der Technologien wohl als im unbekannten Terrain fachlicher Konzepte. Hier drängt sich EJB förmlich auf: Die meisten größeren Java-Projekte verwenden heute EJB. Faktisch ist EJB aber in vielen Fällen gar nicht notwendig, weil im Grunde keine verteilte Anwendung benötigt wird oder deutlich einfachere Varianten existieren, um die Verteilung herzustellen. (Die anderen Vorteile von Java EE funktionieren auch wunderbar ohne EJB [1].)

Andererseits lösen viele der heute verfügbaren Technologien technisch relevante Probleme, sind aber viel zu umständlich in der Anwendung. Das Paradebeispiel ist wieder EJB. Es löst das technische Problem, ein verteiltes System zu schreiben, ist aber sehr schwierig zu benutzen. Außerdem legt EJB eine Softwarearchitektur nahe, die nicht für alle fachlichen Problemstellungen geeignet ist.

Nimmt die Technologie in einem Projekt sehr viel Zeit in Anspruch, verliert das Team wertvolle Zeit, um sich mit einem guten fachlichen Modell zu beschäftigen. Häufig wird dann ein sehr plattes Modell implementiert, das zum Großteil aus Datensäcken besteht, die gespeichert und geladen werden können. Tatsächlich unterscheidet sich dann die Implementierung des fachlichen Kerns der Anwendung nicht von dem, was man früher in Cobol implementiert hat. Vielleicht ist das einer der Gründe dafür, dass die von der Objektorientierung versprochene bessere Wartbarkeit von Software im Großen und Ganzen nicht eingetreten ist. Ganz im Gegenteil werden viele Java-Systeme schon unwartbar, bevor sie überhaupt ausgeliefert werden.

Gemeinsame Sprache

Wenn den Entwicklern die fachlichen Konzepte fehlen, kann es auch keine gemeinsame Sprache zwischen Entwicklern und Anwendern/Kunden geben. Missverständnisse, Reibungsverluste und Enttäuschungen geben sich die Klinke in die Hand. Die Entwickler verstehen nicht, was die Anwender fordern. Die Software tut nicht das, was die Anwender benötigen. Viele unglaublich zähe Meetings zur Abstimmung von Anforderungen werden anberaumt. Und wenn das System mal läuft, kosten fachlich triviale Änderungen mitunter horrende Summen. Das liegt selten an der Profitgier der Entwickler, sondern viel häufiger daran, dass die Konzepte des Anwendungsbereichs unpassend in der Software umgesetzt wurden. Die Denkmodelle von Entwicklern und Anwendern/Kunden passen nicht zusammen.

Was ist zu tun?

Um die fachliche Sichtweise bei der technischen Gestaltung von Systemen stärker in den Mittelpunkt zu rücken, stellen wir die folgenden Forderungen auf:

- Die essenziellen fachlichen Anforderungen sollten früh im Projekt bekannt sein. Um die weniger wichtigen Anforderungen muss man sich später kümmern. Sie können die Architektur als Ganzes nicht in Frage stellen.
- Die Kommunikation zwischen Entwicklern und Kunden (Anwendern und Managern) funktioniert mit Abstand am besten über Prototypen. Je mehr diese sich schon wie echte Software anfühlen und erlauben, dass man seine zukünftigen Arbeitsweisen an ihnen erproben kann, umso wertvoller das darüber erhaltene Feedback. So können Entwickler überprüfen, ob sie die fachlichen Anforderungen wirklich verstanden haben.
- Was fachlich wichtig ist, muss einen wichtigen Ort im Softwareentwurf bekommen. Was weniger wichtig ist, darf auch nur eine Randposition im Softwareentwurf einnehmen.
- Was fachlich wichtig ist, muss im Rahmen einer gemeinsamen Sprache auf den Begriff gebracht werden.
- Was sich fachlich einfach ändern lässt, muss sich auch in der Software einfach ändern lassen. Zur Ausarbeitung einer gemeinsamen Sprache, eines gemeinsamen Verständnisses der Anwendungswelt und der Ziele des Projektes gehört es auch, dass trotz Dokumentation dieses Verständnis vor allem in den Köpfen der beteiligten Personen steckt. Es sollte deshalb in Projekten sowohl auf Entwickler- wie auch auf Anwenderseite für entsprechende personelle Kontinuität gesorgt werden.

Obwohl wir die Konzepte des Anwendungsbereichs für elementar wichtig erachten, sind die Fortschritte im Bereich der fachlichen Modellierung und deren Umsetzung in die Software eher bescheiden. Dabei können einige existierende Techniken bereits heute helfen:

- Die Objektorientierung hat die Möglichkeiten geschaffen, fachliche Konzepte besser in Software abzubilden, als dies vorher möglich war. Ein Patentrezept, wie man mit einfachen objektorientier-

ten Mitteln die oben genannten Ziele erreicht, existiert allerdings nicht. Einen wichtigen Schritt in diese Richtung geht allerdings Eric Evans in [2]. Dort finden sich viele nützliche Anleitungen und Beispiele, wie man das Design der Software an fachlichen Konzepten orientiert.

- Die ersten Ansätze im Bereich der Domain Specific Languages (DSL) versuchen, häufig mit Generierungstechniken kombiniert, zumindest Teile einer Anwendung in einer Sprache zu formulieren, die deutlich näher an der Fachlichkeit selbst ist als eine allgemeine Programmiersprache. Dadurch können die Konzepte der Anwendung stärker in den Mittelpunkt des Entwurfs rücken. Einen Schritt in diese Richtung gehen auch anwendungsorientierte Tests mit dem FIT-Framework [3]. Dort werden die Tests in einer möglichst anwendernahen Sprache formuliert. Allerdings sind DSLs auch wieder nur ein technisches Hilfsmittel, um der Probleme Herr zu werden. Sie nehmen den Entwicklern nicht die Aufgabe ab, die Fachlichkeit zu verstehen und geeignet in das softwaretechnische Design zu integrieren.
- Mit oder ohne GUI-Builder und mit ganz unterschiedlicher Technologie können heute schnell Prototypen gebaut werden. Sie machen das Verständnis der Entwickler über die anwendungsfachlichen Konzepte für die Anwender erfahrbar. In der Regel gilt aber der technologische Rat, dass man sie besser später nicht zur eigentlichen Software weiterentwickelt, weil sie auf Oberflächeneffekte hin und nicht nach einer sauberen Architektur entwickelt sind.

Diese Ansätze, deren Technologien uns hier wiederum nur als Mittel zum Zweck dienen, können uns helfen, möglichst schnell zu einem guten fachlichen Entwurf zu kommen. Vergessen wir hierbei jedoch nicht den organisatorischen Aspekt, der für einen guten fachlichen Entwurf eine entscheidende Rolle spielt: Um anwendungsfachliche Konzepte zu erkennen, zu vermitteln und letztlich geeignet in die Software umzusetzen, sind Fachexperten elementarer Bestandteil eines jeden Entwicklungsprojektes. Diese Fachexperten übernehmen eine maßgebliche Rolle: Sie entscheiden über fachliche Konzepte und beeinflussen so indirekt grundlegende Teile der Architektur des Systems. Diese Verantwortung sollte allen Beteiligten im Projekt klar sein.

Ergebnis

Eine konkrete Technologie ist für ein Projekt nur sehr selten kriegsentscheidend. Sie kann ausgewählt, beherrscht oder notfalls auch ausgetauscht werden. Im Gegensatz zur Realität heutiger Projekte sollte sie nur als Mittel zum Zweck gesehen werden. Viel entscheidender sind die Gestaltung des Systems nach den Konzepten des Anwendungsbereichs und deren Umsetzung in den konkreten technischen Entwurf. Neue Technologien sind aus unserer Sicht vor allem vor diesem Hintergrund zu betrachten: Helfen sie uns tatsächlich bei der möglichst einfachen Umsetzung fachlicher Konzepte?

Mal ehrlich: Ist es denn wirklich so spannend herauszufinden, welche Vorteile XML Schema gegenüber einer DTD hat? Wir beschäftigen uns jedenfalls viel lieber mit der Frage, wie die fachliche Architektur aussehen muss, damit man ein neues Versicherungsprodukt in wenigen Tagen implementieren und integrieren kann, wie man eine Bonus-Funktionalität ohne große Eingriffe in die Architektur hinzufügt, beziehungsweise ähnlichen Problemen. Wir wollen nützliche Software für Kunden und Anwender schreiben. Wenn uns das gelingt, befriedigt uns unser Job.

Wie bei Eclipse wird die Herausforderung für viele Projekte darin bestehen, eine anwendungsfachlich solide Plattform zu implementieren, auf deren Basis neue Anforderungen einfach und schnell implementiert werden können.

Martin Lippert, **Stefan Roock** und **Henning Wolf** sind als IT-Berater und Coaches bei it-agile tätig. Sie arbeiten dort als Experten für agile Softwareentwicklung und das Management agiler Projekte. Sie sind zu erreichen unter: {martin.lippert, stefan.roock, henning.wolf}@it-agile.de.

Links & Literatur

[1] Rod Johnson, Jürgen Höller: J2EE Development without EJB, Wiley, 2004

[2] Eric Evans: Domain Driven Design, Addison-Wesley, 2003

[3] Fit Framework for Integrated Test: fit.c2.com