www.bt-magazin.de 2.2010 Deutschland: € 14,90 Österreich: € 16,80 Schweiz: sFr 29,80 Luxemburg: € 16,90

Business Technology

Business Technology

Architektur & Management Magazin

Stark:
„Der Kritiker sorgt für die Qualität“

Expertenwissen für IT-Architekten, Projektleiter und Berater

AGILITÄT

Business Case für Agilität

Retrospektiven – wider die Macht der Verdrängung

Was uns die Geschichte lehrt
Vorteile von Storytelling in Projekten

Nur Wandel hat Bestand
Einführung agiler Methoden

Agil oder ingenieurmäßig
Agile Entwicklung im Vergleich mit anderen Branchen

4 191858 414906 02

Rechnet sich der Einsatz von agilen Methoden?

Business Case für Agilität

AUTOREN: ANDREAS EBBERT-KARROUM, MIRKO NOVAKOVIC

Agile Methoden wie Scrum [1] oder Kanban [2] etablieren sich zunehmend als Alternative zu klassischen Vorgehensmodellen in der Softwareentwicklung. Ist das alles nur ein großer Hype oder steckt mehr dahinter? Gibt es vielleicht sogar einen echten Business Case? Anhand einer idealisierten Geschichte soll verdeutlicht werden, welche Chancen die Agilität bietet und warum sich ein Umstieg rechnet.

Heute ist ein entscheidender Tag für Frau Schneider. In den letzten Wochen hat die Bereichsleiterin für Vertrieb bei einer großen deutschen Versicherungsgesellschaft viel Zeit investiert. Sie hat ihre Ideen von einer besseren Unterstützung der Agenturen und höheren Kundenbindung mit Vorstand und Kollegen diskutiert. Jetzt hat Sie endlich das benötigte Budget bekommen. Die Zeit drängt. Entscheidend wird auch sein, mit dieser innovativen Idee als erster im Markt zu starten. Jeder Außendienstmitarbeiter soll mit einem iPad ausgestattet werden. Auf diesem soll eine Vertriebssoftware laufen, die neben dem normalen Angebots- und Antragsgeschäft vor allem multimediale Elemente zur Vertriebsunterstützung enthält. Gerade junge Vertriebsmitarbeiter sollen dadurch eine bessere Servicequalität und höhere Abschlussraten erzielen können. Zudem lässt sich über eine dynamisch veränderbare

Software auch besser steuern, was und wie verkauft wird. Entscheidende Vorteile in einem hart umkämpften Markt. Außerdem soll es eine direkte Verbindung zwischen Agentursoftware und den Kunden geben. iPhone und das Web 2.0 mit seinen sozialen Netzwerken sollen genutzt werden, um einen Dialog mit dem Kunden herzustellen und Mehrwertdienste anzubieten. So soll die Kundenbindung erhöht und die Cross-Selling-Rate optimiert werden.

„LIVE" IN SECHS MONATEN

Nur eine Hürde liegt noch vor Frau Schneider. Aber eine Hürde, über die sie in den letzten Jahren sehr häufig gestolpert ist. Gleich hat sie ein Gespräch mit Herrn Müller, dem Bereichsleiter der Anwendungsentwicklung. Sie muss mit ihm über den Einführungstermin und die Anforderungen der Software sprechen. In sechs Monaten möchte Frau Schneider dem Außendienst eine erste Version der Software inklusive iPad bereitstellen. Parallel dazu soll das neue Kundenportal im Internet und für mobile Plattformen präsentiert werden. Gerade für das Kfz-Versicherungsgeschäft ist es wichtig, pünktlich vor dem Jahresendgeschäft ab Oktober zu veröffentlichen, da ansonsten das Geschäft für ein Jahr gelaufen ist.

Das aktuelle Außendienstsystem ist erst vor einem Jahr produktiv gegangen. Aus den ursprünglich geplanten drei Jahren Projektlaufzeit sind mehr als sechs Jahre geworden. Gesetzliche Änderungen wie die EU-Vermittlerrichtlinie und marktbedingte Produktänderungen hatten zu hohem Änderungsaufwand geführt. Die Mitarbeiter von Frau Schneider hatten zu Beginn des Projekts alle Anforderungen im Detail spezifiziert. Dabei wusste man selbst noch gar nicht genau, wie die neue Anwendung aussehen und funktionieren sollte. Aber die Methodik wollte es so, und die IT garantierte dafür, Termin und Budget zu halten. Am Ende war man dann im Termin drei Jahre nach hinten gerutscht – trotzdem war der Projektstatus immer „grün". Wie sollte man jetzt in sechs Monaten eine ganz neue Software bauen?

Herr Müller hatte bereits vorab die Informationen von Frau Schneider bekommen und im ersten Moment gedacht, sie hätte sich bei der Projektlaufzeit in der Zeiteinheit vertan. Sechs Monate? Doch der Vorstand unterstützte Frau Schneiders Idee, und der Druck auf Herrn Müller war groß. Er wusste, dass man es mit der etablierten Entwicklungsmethodik nicht schaffen konnte. Er dachte an andere Projekte unter ähnlichem Zeitdruck zurück. Beispielsweise die „Jahr-2000-Umstellung" oder die Einführung der „Riester-Rente". Damals hatte man auf die formalen Methoden verzichtet, die besten Leute zusammengezogen und Sie einfach machen lassen. „Feuerwehr" oder „Schnelle Eingreiftruppe" wurden diese Teams genannt. Funktioniert hatte es dann immer – Zeit und Budget wurden dabei auch gehalten. Warum also nicht das Vorgehen auch bei der normalen Entwicklung einsetzen? Herr Merten, ein junger Abteilungsleiter von Herrn Müller, hatte zudem bei sich eines dieser neuen „agilen" Vorgehensmodelle ausprobiert: Scrum und XP. Im Prinzip ein ähnlicher Ansatz wie bei den „Feuerwehr"-Teams – nur strukturierter und mit erprobten Praktiken. Funktioniert hatte das sehr gut. Sein Projekt hatte sehr schnell lauffähige Anwendungen ausgeliefert, und gerade die Fachbereiche waren begeistert von Flexibilität und Produktivität dieser „agilen Teams".

Herr Müller hatte sich mit Herrn Merten über das Projekt von Frau Schneider unterhalten und er war sich sicher, dass er mit dem richtigen Team die Anwendung in sechs Monaten „live" bringen könnte. Viele Alternativen hatte Herr Müller nicht, sodass er Herrn Merten sein Vertrauen aussprach und es mit Scrum versuchen wollte.

Frau Schneider war überrascht, als Sie von Herrn Müller hörte, dass es möglich sei, ihre Idee in sechs Monaten umzusetzen. Die Bedingungen waren für sie akzeptabel. Herr Merten hatte ihr deutlich gemacht, dass sie einen „Product Owner" stellen müsste, der einen Anforderungskatalog erstellen und priorisieren sollte. Zudem mussten Sie ein bis zwei Mitarbeiter abstellen, die mit der IT zusammen in einem Team arbeiten sollten, um direkt die Testfälle zu erstellen und bei den fachlichen Themenstellungen zu unterstützen. Man wollte mit den wichtigsten Funktionen für den Außendienst beginnen und dann alle vier Wochen eine lauffähige Anwendung zur Verfügung stellen, die dann auch schon von den Pilotagenturen getestet werden könnte. Für Frau Schneider war dies eine hervorragende Möglichkeit, schnell Feedback von den Anwendern zu bekommen, um so die erste Version so nah wie möglich an den Bedürfnissen der Vertriebsmitarbeiter ausrichten zu können. Zudem konnte man Unklarheiten bei den Anforderungen schrittweise mit Leben füllen und musste nicht alle Anforderungen im Voraus spezifizieren.

Nach einem Training für die Teammitglieder startete das Projekt zwei Wochen später mit der ersten Iteration: dem „1. Sprint". Man wollte damit die prinzipiellen technischen Schnittstellen und den Anwendungsrahmen der iPad-Anwendung erstellen. So sollte zum einen das technische Risiko minimiert und zum anderen schon zu Beginn das Feedback der Anwender über die Bedienerfreundlichkeit eingeholt werden. Diese würde für die Vertriebsmitarbeiter eine der entscheidenden Erfolgsfaktoren für das Projekt sein, denn die bessere Unterstützung des Vertriebs war eines der zentralen Ziele der neuen Anwendung. Sollte die Anwendung nicht auf 100 %-ige Akzeptanz stoßen, wäre das Projekt gescheitert. Unter den Piloten waren deshalb auch erfahrene und junge Agenturmitarbeiter, die durch die Mitarbeit und die Möglichkeit des frühen Feedbacks gewonnen werden sollten.

Der erste Sprint war ein voller Erfolg und man konnte bereits die Mainframe-Schnittstellen der Bestandssysteme anbinden sowie einen ersten funktionierenden Anwendungsrahmen mit einer Partnersuche anbieten. Bei der Vorführung der lauffähigen Anwendung vor allen Interessenten gab es viel konstruktives Feedback der Anwender,

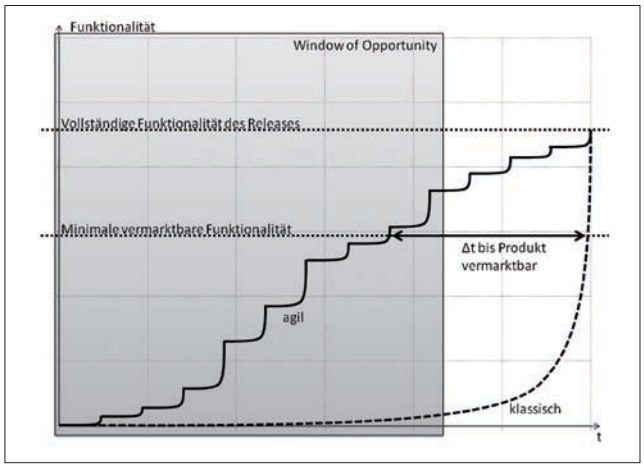

Abb. 1: Treffen des „Window of Opportunity" mit der minimalen vermarktbaren Funktionalität

und auch der Fachbereich konnte besser einschätzen, wie sich eine iPad-Anwendung verhält und wie man die Funktionen an die neue Technologie anpassen musste.

In den nächsten Sprints wurden ständig Anforderungen neu priorisiert und neue Anforderungen hinzugefügt. Parallel startete ein zweite Scrum-Team mit der Umsetzung des Kundenportals. Nach vier Monaten wurde man durch einen Konkurrenten überrascht, der einen neuen Kfz-Tarif vorgestellt hatte. Dieses neue Produkt ermöglichte es den Kunden, kleinere Lackschäden auf Basis von SmartRepair zu beheben. Der Vorteil für den Kunden ist dabei, dass der Selbstbehalt der Teilkasko nicht fällig wird. Der Fachbereich wollte zwei Monate vor Rollout noch nachziehen und zudem noch eine iPhone-Anwendung (App) zur Verfügung stellen, mithilfe derer man den nächsten autorisierten SmartRepair-Händler finden konnte. Im Gegenzug verzichtete man auf die Schadensmeldung im Internet für das erste Release. Durch die Umpriorisierung konnte im fünften Sprint direkt mit der Umsetzung des „SmartRepair"-Themas angefangen werden.

Nach sechs Monaten war ein Basissystem fertig, das genau auf die Bedürfnisse der Vertriebsmitarbeiter abgestimmt war. Die wichtigsten Elemente der Angebotserstellung inkl. neuer multimedialer Elemente waren vorhanden und auch das Portal konnte mit einer Vertragsauskunft und einer iPhone-App für die Lokalisierung von SmartRepair-Händlern ausgeliefert werden. Die Einführung war ein voller medialer und vertrieblicher Erfolg. Gerade junge Leute fühlten sich von der modernen Kommunikation im Vertrieb und den Mehrwerten über das Portal und die Apps angesprochen. Über eine offene Community in Facebook wurden zudem neue Anforderungen diskutiert und auch Kritik am System geäußert.

In den folgenden zwölf Monaten arbeiteten die Teams weiter an den Funktionen des Systems, erweiterten es schrittweise und nahmen das Feedback der Kunden und Vertriebsmitarbeiter auf, um es in monatlichen Releases umzusetzen. Die alte Vertriebsanwendung war nach 1,5 Jahren vollständig abgelöst – zu 20 % der ursprünglichen Kosten und deutlich höherer Zufriedenheit der Anwender und Kunden.

UND DIE MORAL VON DER GESCHICHTE?

Die Anforderungen an die IT nehmen immer mehr zu. Marketing und Vertrieb treiben uns in immer kürzere Entwicklungszyklen. Immer häufiger hört man selbst von hartgesottenen Verfechtern schwergewichtiger Methoden, dass man in bestimmten Fällen auf agile Verfahren zurückgreift: Nämlich dann, wenn man nicht genug Zeit und Budget für die Anwendung der Methode hat. Der Business Case für Agilität lässt sich dabei anhand von vier Kernthesen aufzeigen:

1. Schneller geschäftlicher Nutzen und Time to Market
2. Weniger Fehl- und Blindleistungen
3. Risikomanagement von komplexen Situationen
4. Hohe und nachhaltige Qualität

SCHNELLER GESCHÄFTLICHER NUTZEN UND TIME TO MARKET

Agile Methoden arbeiten mit kurzen Iterationen und liefern in jeder Iteration ein lauffähiges Inkrement („Working software over comprehensive documentation" [3]) aus. Die Iterationen dauern in der Regel zwei bis vier Wochen und beinhalten die geschäftlichen Funktionen mit der aktuell höchsten Priorität. Nach jeder Iteration kann der Funktionsumfang der nächsten Iteration neu festgelegt werden, d. h. es können Funktionen umpriorisiert oder auch völlig neue Funktionen hinzugefügt werden. Die Geschwindigkeit, mit der man heute am Markt agiert (Time to Market), kann entscheidend für den wirtschaftlichen Erfolg sein. Ein Produkt als Erster auf den Markt zu bringen (First Mover) bedeutet in der Regel, dass man höhere Preise erzielen kann, da es wenige oder keine Konkurrenten gibt.

Abbildung 1 zeigt, dass der iterative Ansatz von agilen Methoden es ermöglicht, schon früher mit einer minimal vermarktbaren Funktionalität zu starten, da prinzipiell jedes Inkrement lauffähig ist. Frau Schneider konnte beispielsweise schon nach sechs Monaten eine neue Vertriebssoftware ausliefern, obwohl der gesamte Funktionsumfang erst nach 18 Monaten fertiggestellt wurde. Entscheidend kann aber auch der Zeitpunkt der Markteinführung sein. Wenn Frau Schneider den neuen Kfz-Tarif nicht zum Oktober in das Vertriebssystem integriert hätte, wäre das Geschäft für ein Jahr ggf. stark eingeschränkt gewesen.

Ist die Konkurrenz besonders innovativ und bietet, wie in unserer Geschichte, den Kfz-Tarif mit Smart-Repair-Option ohne Selbstbehalt, ist es wichtig, schnell reagieren zu können und ggf. ein ähnliches Produkt anzubieten (Fast Follower). Als Second Mover hat man dann sogar die Chance, das Produkt der Konkurrenz zu übertreffen.

Agilität unterstützt schnelle Entwicklungszyklen und akzeptiert Veränderungen durch einen empirischen Entwicklungsprozess basierend auf iterativ entwickelten Inkrementen.

WENIGER FEHL- UND BLINDLEISTUNG

Studien gehen davon aus, dass fast zwei Drittel der Funktionen in einer Anwendung selten oder überhaupt nicht genutzt werden [4]. Im Umkehrschluss bedeutet das: Der Geschäftswert der Anwendung steckt hauptsächlich im verbleibenden Drittel. Die Problematik in Softwareprojekten ist, schon zu Beginn eines Projekts genau das Drittel zu identifizieren, das den größten Geschäftswert liefert. Frau Schneider und ihr Fachbereich hatten beispielsweise nur eine sehr vage Vorstellung davon, wie ihre Ideen in konkrete Funktionen umgesetzt werden sollen. Diese vage Vermutung über die benötigte Funktionalität konkretisiert sich aber, sobald man ein System benutzt hat [5]. In unserer fiktiven Geschichte war es wichtig, die Anwendung auf dem iPad schon sehr früh auszuprobieren, um ein Gefühl für die Bedienung zu bekommen.

Selbst unter der idealisierten Annahme, man kenne zu Projektbeginn schon alle Anforderungen im Detail und wisse um deren Geschäftswert, bleibt das Risiko, dass sich diese Anforderungen im Projektverlauf ändern, weil sich die Rahmenbedingungen geändert haben. Beispielsweise durch eine gesetzliche Änderung wie die Einführung der EU-Vermittlerrichtlinie. Bei agilen Methoden hat man nach jeder Iteration die Möglichkeit, den aktuellen Stand zu kontrollieren und ggf. nachzusteuern.

Durch das kontinuierliche Feedback der Kunden und Benutzer zu den bereits fertiggestellten Releases nähert man sich schrittweise dem wertschöpfenden Anforderungsdrittel im Projektverlauf. Zudem werden die tatsächlichen Anforderungen des Kunden durch die intensive Kommunikation besser verstanden, was in der Tendenz zu günstigeren Lösungen des eigentlichen Problems führt.

RISIKOMANAGEMENT VON KOMPLEXEN SITUATIONEN

Wie Friedrichsen in seinem Artikel in dieser Ausgabe [6] dargelegt hat, ist die Softwareentwicklung eine komplexe Aufgabenstellung, die sich durch die hohe Dynamik der Randbedingungen (Anforderungen, Technologien, Menschen) auszeichnet. Um einen komplexen Prozess steuern zu können, bedarf es eines empirischen Vorgehens. Eine Analogie zur Verdeutlichung: Auch Klimaanlage und Heizung können nicht für Stunden oder Tage im Voraus programmiert werden, um eine konstante Raumtemperatur zu erreichen. Die aktuelle Temperatur muss ständig gemessen werden, um dann die Regler nachsteuern zu können. Erst dadurch können Änderungen der Randbedingung wie Sonneneinstrahlung, Anzahl und Aktivität der im Raum befindlichen Personen ausgeglichen werden.

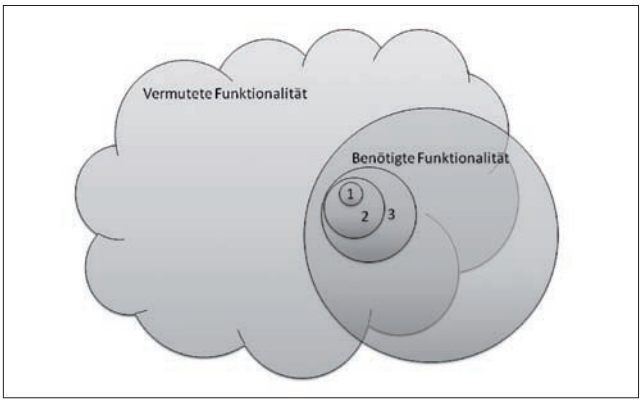

Abb. 2: Iteratives Herantasten an die tatsächlich benötigte Funktionalität

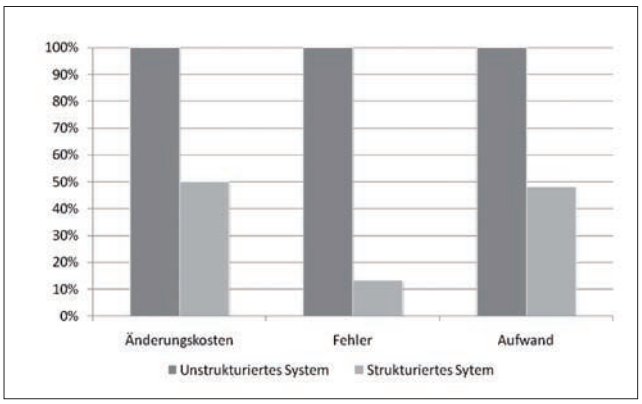

Abb. 3 : Vergleich von identischen Änderungen an einer Komponente vor und nach einer Strukturierung

Analog verhält es sich mit der Softwareentwicklung: Erst mit einem Prozess, der viele Messpunkte und Möglichkeiten zur Nachsteuerung bietet, kann man in komplexen Situationen das Risiko in den Griff bekommen. Gemessen wird z. B. in Scrum auf vielen Ebenen: Täglich beim Daily Scrum, um zu prüfen, ob das Sprint Commitment noch gehalten werden kann. Bei dem Sprint Review, um zu prüfen, wie viel Fortschritt innerhalb des Sprints gemacht wurde und welche Auswirkungen das auf das Release hat.

Das größte Risiko, das man dann eingeht, ist eine Iteration, die komplett am Geschäftsnutzen vorbeigeht: Man hat also keinerlei Kundennutzen aus der Arbeit der Iteration generieren können. Der iterative Ansatz deckelt das Risiko aber auch auf eine Iterationslänge. Im Gegensatz zu schwergewichtigen Prozessen, bei denen man sehr häufig erst gegen Projektende merkt, ob der „grüne“ Status eher auf Hoffnung oder lauffähiger Software basiert, die den Kundenanforderungen entspricht. Der Business Case von Agilität kann im Extremfall sogar darin liegen, das Projekt zu beenden – der finanzielle Schaden kann durch die frühzeitige Validierung von Risiken reduziert werden.

HOHE UND NACHHALTIGE QUALITÄT

Spaghetticode ist einfacher und schneller zu schreiben als eine gut strukturierte Komponente. Das jedenfalls be-

hauptet Josuttis, der sogar so weit geht zu behaupten, die Zukunft läge im „Crappy Code“. Man solle sich einfach damit abfinden, dass schlechter Code normal ist [7]. Marketing und Vertrieb würden einen solchen Kostendruck aufbauen, dass man einfach zu der billigsten Art der Programmierung greifen müsse, und das sei eben, einfach etwas zusammenzuhacken, was funktioniert. Der Denkfehler aus Sicht der Autoren: Schlechter Code ist nicht billiger, ganz im Gegenteil. Josuttis Ansatz greift zu kurz, weil er sich nur auf die einmaligen Erstellungskosten des Codes bezieht. Die wahren Kosten verstecken sich aber ganz woanders. 85 % der Kosten, Tendenz steigend, stecken in der Wartung [8]. Ja, Spaghetticode zu schreiben ist günstiger, aber eben auch nur das Schreiben. Dazu kommt, dass man sich bei einer agilen Vorgehensweise quasi in der zweiten Iteration schon in der Wartungsphase befindet. Nun, das könnte jetzt ein Argument sein, eben nicht iterativ zu entwickeln, um die Wartungsphase möglichst weit herauszuschieben, aber auch die Betrachtungsweise ist verkürzt. Denn auch wenn die Wartungsphase „offiziell“ erst nach dem Release anfängt, Software zu warten heißt, zu bestehender Funktionalität neue hinzuzufügen, und das geschieht tagtäglich in der Softwareentwicklung.

Eine Untersuchung des amerikanischen Verteidigungsministeriums hat den Einfluss von der Struktur eines Systems auf die Wartbarkeit gemessen. Danach wurden Kosten, Fehler und zeitlicher Aufwand einer Änderung an einem System gemessen, bevor und nachdem es von einem unstrukturierten in einen strukturierten Zustand überführt wurde. Die Unterschiede sind drastisch: Die Änderungen am strukturierten System dauerten nur halb so lange und verursachten die Hälfte der Kosten. Die Anzahl der Fehler ging noch drastischer auf fast ein Zehntel zurück [9]. All das sind Kostenvorteile, die man mitnimmt, wenn sich das System in einem qualitativ hochwertigen Zustand befindet.

Die Erfahrung zeigt, dass die Qualität des Codes häufig Zeit und Budget zum Opfer fällt. Martin Fowler nennt diese Qualitätsschulden, die ein Projekt über die Laufzeit aufbaut, auch Technical Debt [10]. Agile Methoden wie Scrum führen deshalb eine „Definition of Done“ ein, die die Qualitätskriterien für eine Anforderung festlegt. Nur wenn diese komplett erfüllt sind, wird eine Anforderung in einer Iteration ausgeliefert. Zudem schreiben agile Softwareentwicklungsmethodiken wie eXtreme Programming [11] Praktiken wie Test-driven Development, Continuous Integration und Pair Programming vor, um die Codequalität der Anwendung hochzuhalten.

ZUSAMMENFASSUNG

Natürlich kann kein allgemeingültiger Business Case aufgestellt werden, ohne die konkrete Situation mit einzubeziehen. Trotzdem rechnen sich agile Methoden in vieler Hinsicht und sind definierten („Wasserfall“-)Prozessen vorzuziehen.

Regelmäßige iterative Fertigstellung kleiner Inkremente kann die Time to Market von Produkten deutlich verkürzen. Dadurch können früher Umsätze erzielt und Gelegenheiten wahrgenommen werden, die man sonst als zu kurzfristig hätte verstreichen lassen müssen. Änderungen können nach jeder Iteration erfolgen und unterliegen keinem teuren Change Management, das versucht, dem Kunden die Änderungen auszureden. So kann schnell auf neue Rahmenbedingungen und die Konkurrenz reagiert werden. Die enge Kommunikation mit Kunden und Benutzern führt dazu, dass nur das entwickelt wird, was tatsächlich benötigt wird. Dadurch werden Fehl- und Blindleistung vermieden, was eine Kostenersparnis von rund zwei Dritteln der Entwicklungskosten ausmachen kann. Durch das Risikomanagement komplexer Problemstellungen in der Softwareentwicklung auf Basis empirischer Prozesse können Projektrisiken schon früh erkannt und gehandhabt werden. Der Nutzen des Systems steigt quasi durch die Vermeidung negativen Geschäftswerts. Die Forcierung hoher und nachhaltiger Qualität senkt die Kosten in der Wartung des Systems nachweislich bis auf die Hälfte der Kosten bei einem üblichen Vorgehen.

In Addition kann man durch den Einsatz von Agilität also Kosten sparen, Risiken minimieren und den Nutzen einer Anwendung steigern – auch wenn sicherlich gewisse Rahmenbedingungen erfüllt sein müssen.

Mirko Novakovic ist Mitgründer und Vorstand der codecentric AG. Er entwickelt selbst seit mehr als 14 Jahren geschäftskritische Anwendungen, davon die ersten zehn Jahre mit schwergewichtigen Methoden. Ein hochqualifiziertes Team, agile Methode und solide Technologien stehen für ihn und die Agile Software Factory im Mittelpunkt erfolgreicher und kosteneffizienter Entwicklung.

Andreas Ebbert-Karroum leitet das Competence Center Agilität der codecentric AG. Seit 14 Jahren entwickelt er mit Java[S|E]E und bringt seine Kompetenzen auch als Scrum Master und Product Owner ein. Aktuell arbeitet er aktiv am Scrum-Developer-Programm von scrum.org mit. Seine Leidenschaft ist die Auflösung der Engpässe zur Verbesserung der Agilen Software Factory.

Links & Literatur

[1] Scrum Guide: http://www.scrum.org/scrumguides/
[2] Software Kanban: http://agilemanagement.net/
[3] Agiles Manifest: http://agilemanifesto.org/
[4] Schwaber, Ken: „The Enterprise and Scrum“, Microsoft Press, ISBN-13: 978-0735623378, 2007
[5] Watts S. Humphrey: „A Discipline for Software Engineering“, SEI Series in Software Engineering. Addison-Wesley 1995
[6] Friedrichsen, Uwe: „Agil oder doch lieber ingenieursmäßig?“ Gleiche Ausgabe
[7] Josuttis, Nicolai M.: „Welcome Crappy Code“: http://www.josuttis.com/WelcomeCrappyCode.html
[8] Koskinen, J.: „Software Maintenance Cost“, 2003: http://www.cs.jyu.fi/~koskinen/smcosts.htm
[9] Department of Defense: Guidelines for Successful Acquisition and Management of Software Intensive Systems, Anhang F: http://www.stsc.hill.af.mil/resources/tech_docs/gsam3.html
[10] Technical Debt: http://martinfowler.com/bliki/TechnicalDebt.html
[11] eXtreme Programming: http://www.extremeprogramming.org

Agile Entwicklung im Vergleich mit anderen Branchen

Agil oder ingenieurmäßig?

AUTOR: UWE FRIEDRICHSEN

Eine häufig gestellte Frage ist, warum man agile Vorgehensweisen in anderen produzierenden Branchen nicht antrifft. Gibt es tatsächlich triftige Gründe dafür oder haben wir in der IT unsere Disziplin nur nicht im Griff und versuchen uns dann mit Dingen wie Agilität zu retten, anstatt uns wie die anderen Branchen auf erprobte Ingenieurstugenden zu besinnen? Oder gibt es Agilität sehr wohl in anderen Branchen, und suchen wir nur an den falschen Stellen?

Auch im Rahmen des Speaker Panels zum Abschluss des Agile Days auf der JAX 2010 [1] wurde von einem Teilnehmer aus dem Auditorium die zuvor beschriebene Frage an die versammelten Speaker des Tages gestellt. Das scheinbare Fehlen der Agilität in anderen Branchen wie etwa dem Gebäude- oder dem Autobau verursacht bei vielen Beobachtern und Interessenten von agilen Vorgehensmodellen (z. B. Scrum [2] oder XP [3]) häufig gewisse Vorbehalte gegenüber diesen Vorgehensweisen. Sie fragen sich, ob nicht einfach das Fehlen einer vernünftigen ingenieursmäßigen Behandlung der Softwareentwicklung – was schon seit vielen Jahren immer wieder bemängelt wird – die IT gezwungen hat, sich mit Themen wie Agilität auseinanderzusetzen. Sie fragen sich, ob es nicht sinnvoller wäre, die Softwareentwicklung vernünftig ingenieurmäßig aufzubereiten, um deterministisch eine Produktionsqualität, wie z. B. im Fahrzeug- oder im Hausbau, zu erzielen, anstatt sich auf Behelfslösungen wie agile Vorgehensmodelle zu verlassen. Es gibt eine Menge mehr oder minder guter Antworten auf diese Fragen. Auch die Speaker auf der JAX hatten einige (gute) Antworten parat. Verfolgt

man das Thema aber weiter, sind aus Sicht des Autors die folgenden zwei Kernaussagen von essenzieller Bedeutung:

- Softwareentwicklung ist in der Tat anders als andere Ingenieurwissenschaften, da sie einen relativ einmaligen Bauprozess hat.
- Ein ausschließlich ingenieurmäßiges Vorgehen funktioniert nicht in komplexen Umgebungen, mit denen wir bei der Softwareentwicklung häufig konfrontiert sind.

Was das im Detail bedeutet, wird im Folgenden genauer betrachtet.

SOFTWAREENTWICKLUNG IST ANDERS

Warum kann man z. B. ein Haus oder ein Brücke nicht agil bauen? Pavlo Baron, einer der Speaker des zuvor beschriebenen Panels auf der JAX 2010, hat dazu ein schönes Beispiel gebracht: „Stellen Sie sich vor, man würde eine Brücke agil bauen. Es werden zehn Meter gebaut, dann lässt man die Autos losfahren und dann ...“ (Pause, gefolgt von Gelächter aus dem Publikum) „... oder aber wir bauen die Brücke zwar auf der ganze Länge, aber nur zehn Zentimeter breit.“ Beides führt nicht zum gewünschten Mehrwert für die Anwender. Agiler Brückenbau macht offensichtlich wenig Sinn. Warum geht es dann in der Softwareentwicklung?

Das Hauptproblem liegt in einer falschen Wahrnehmung des Bauprozesses in der Softwareentwicklung. Zum besseren Verständnis betrachten wir noch einmal den Brückenbau. Der Brückenbau lässt sich in eine Entwurfs- und eine Bauphase unterteilen. In der Entwurfsphase werden die Anforderungen an die Brücke gesammelt, diese werden konsolidiert und abgestimmt, es wird eine Lösungsidee entworfen und immer weiter verfeinert, bis am Ende die detaillierten Baupläne fertig sind. Die Baupläne sind das finale Design des Brückenbaus. Danach geht es in die Bauphase. Wenn man nichts falsch gemacht hat und keine ungeplanten Überraschungen auftauchen, kann man auf Basis der Pläne die Brücke komplett bauen, ohne dass noch irgendeine weitere Klärung oder Verfeinerung notwendig wäre. Das ist die „Definition of Done“ der Entwurfsphase: Die Entwurfsphase ist abgeschlossen, wenn alle notwendigen Festlegungen getroffen sind, um das Produkt – in dem Fall die Brücke – vollständig bauen zu können.

Wie sieht das jetzt in der Softwareentwicklung aus? Beginnen wir mit einer einfachen Frage: Was ist das Produkt in der Softwareentwicklung? Das ist das ausführbare Programm, nicht der Sourcecode. Sourcecode interessiert die meisten Anwender nicht die Bohne und hat insbesondere auch keinen Mehrwert für sie, ein ausführbares Programm hingegen schon. So weit, so einfach.

Jetzt zur deutlich schwierigeren Frage: Wann ist in der Softwareentwicklung die Entwurfsphase zu Ende und wann beginnt die Bauphase? Und mit „Phase“ ist hier keine Phase im Sinne einer zeitlichen Abfolge wie beim Wasserfallmodell gemeint, sondern eine logisch in sich abgeschlossene Tätigkeit. Unter Berücksichtigung der zuvor beschriebenen Definition of Done hat Neal Ford [4] die folgende für viele überraschende Antwort geliefert: Entgegen der weit verbreiteten Ansicht ist die Entwurfsphase nicht am Ende der Designphase zu Ende, sondern erst am Ende der Implementierungsphase. Erst mit dem Schreiben der letzten Zeile Code ist der Entwurf abgeschlossen. Erst dann sind alle Festlegungen für das zu erstellende System getroffen.

Und wo bleibt dann die Bauphase? Nun, die ist in der Softwareentwicklung aufgrund des virtuellen Werkstoffs Software so billig, dass man sie häufig übersieht: Die Bauphase ist – wie der Name schon andeutet – der Build sowie das Deployment, das Übersetzen des Sourcecodes sowie das Zusammensetzen und Verteilen der fertigen Anwendung. Interessanterweise sind wir an dieser Stelle ingenieursmäßig heute schon so weit, dass wir das vielfach schon auf Knopfdruck können. Moderne Build- und Deploymentsysteme können das vollautomatisch. Wir verfügen an der Stelle über eine Effizienz und Effektivität, von der viele andere Ingenieurdisziplinen nur träumen können.

Woher kommt der häufig gemachte falsche Schnitt zwischen Entwurfs- und Bauphase? Dafür gibt es aus Sicht des Autors zwei Hauptgründe:

- Die Bauphase ist so billig, dass man sie einfach übersieht. Dadurch, dass das Produkt Software so ungemein günstig zu produzieren (nicht zu entwerfen) ist, gehen sowohl die Zeit als auch die Kosten für den reinen Bau der Software nahezu gegen Null. Das ist ein grundlegender Unterschied zu den meisten anderen Branchen, in denen der Bauprozess sehr aufwändig und teuer ist, wesentlich aufwändiger und teurer als der Entwurfsprozess. Ganz anders in der Softwareentwicklung: Hier können wir so häufig bauen wie wir wollen. Es kostet uns nur einen Mausklick. Die meisten Leute suchen aber nach einer aufwändigeren Phase und assoziieren so die Implementierungsphase fälschlicherweise mit der Bauphase anderer Ingenieurdisziplinen.
- Der Begriff „Designphase“ lässt vermuten, dass am Ende des Designs die Bauphase beginnt. Das ist grundsätzlich auch nicht falsch; nur wurden die bis heute üblichen Begriffe in der Softwareentwicklung zu einer Zeit geprägt, als Programmierung in der Tat noch Teil der Bauphase war. Zu der Zeit wurden Programme häufig während der Analyse und des Designs bis auf Statement-Ebene spezifiziert und die so genannten Anwendungsprogram-

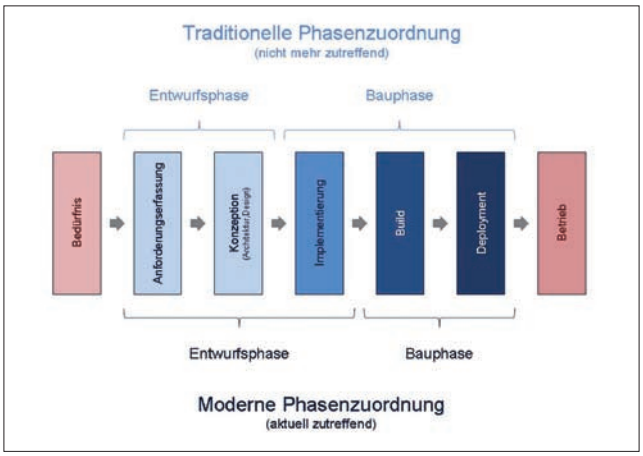

Abb. 1: Phasenzuordnung

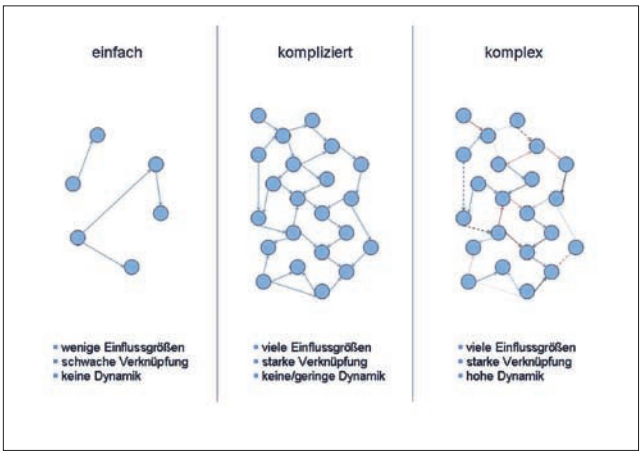

Abb. 2: Unterschiedliche Systeme

mierer hatten dann die Aufgabe, die fertige Programmbeschreibung relativ mechanisch in COBOL, PL/1, Assembler o. ä. zu übersetzen. Diese Art der Aufgabentrennung zwischen Design und Programmierung gibt es aber schon lange nicht mehr. Entwickler sind Designer und Programmierer in einer Person. Aufgrund des Zeit- und Kostendrucks erfolgen Designs nur noch auf einer wesentlich gröberen Ebene und werden im Rahmen der Implementierung dann weiter verfeinert. Das eigentliche Programmieren ist dabei nur noch Nebensache. Die alte Vorgehensweise kann sich heute niemand mehr leisten, weder zeitlich noch monetär. Was geblieben ist, sind die alten Bezeichnungen. **Abbildung 1** verdeutlicht die beiden Sichtweisen noch einmal.

Zusammenfassend bedeutet das, dass sowohl die extrem niedrigen Kosten in der Bauphase als auch eine nicht mehr zeitgemäße Begriffsverwendung zu den beschriebenen Verwirrungen beim Vergleich der Softwareentwicklung mit anderen Branchen geführt haben. Zusätzlich eröffnen die extrem niedrigen Baukosten in der Softwareentwicklung gegenüber anderen Branchen ganz neue Möglichkeiten. Softwareentwicklung ist also tatsächlich anders.

Oder können Sie sich folgendes Szenario im Brückenbau vorstellen: Erste Woche: „Ich will eine Brücke über diesen Fluss. Bauen Sie mal eine ganz normale Brücke bis nächste Woche." Zweite Woche: „Ach, ich glaube, dass wir die doch besser vierspurig bauen sollten." Dritte Woche: „... oder vielleicht doch besser als Hängebrücke?" Vierte Woche: „Ich habe gerade die Verkehrsprognose erhalten. Wir sollten sie doch auf sechs Spuren erweitern und als Tunnel bauen." ... und so weiter. Das klingt absolut absurd, oder? In der Softwareentwicklung ist das aber normal und funktioniert meistens sogar, weil die Bauphase praktisch nichts kostet.

SOFTWAREENTWICKLUNG IST KOMPLEX

Kommen wir zur zweiten Aussage: Ein ausschließlich ingenieurmäßiges Vorgehen funktioniert nur in einfachen und komplizierten Umgebungen, aber nicht in komplexen, mit denen wir bei der Softwareentwicklung häufig konfrontiert sind. Was soll das bedeuten? In der Systemtheorie und darauf aufsetzenden Konzepten findet man die Unterscheidung zwischen den folgenden Typen von Umgebungen bzw. Systemen (z. B. [5], [6], **Abb. 2**).

- *Einfache Systeme* zeichnen sich durch einfach nachvollziehbare Ursache-Wirkung-Zusammenhänge aus. Es gibt nur wenige Einflussgrößen, die nur schwach miteinander verknüpft sind und sich über die Zeit nicht verändern (keine oder nur sehr geringe Dynamik). Probleme in einfachen Systemen sind leicht zu verstehen und zu lösen. Der Lösungsweg lässt sich in klar strukturierten Prozessen und so genannten Best Practices für jedermann wiederholbar festhalten. Ein einfaches Problem ist z. B. das Reinigen eines Fahrzeugs oder das Austauschen der Tonerkartusche bei einem Drucker.
- *Komplizierte Systeme* unterscheiden sich von einfachen Systemen dadurch, dass es viele Einflussgrößen gibt, die stark miteinander verknüpft sind. Die Verknüpfungen sind aber ebenfalls über die Zeit konstant (geringe Dynamik). Im Gegensatz zu einfachen Systemen ist die Ursachen-Wirkung nicht mehr für jedermann offensichtlich. Deshalb benötigt man häufig Experten, um Probleme in komplizierten Systemen zu analysieren und zu lösen. Das System bleibt aber in sich deterministisch und es gibt zumindest eine richtige Lösung für ein Problem. Ein kompliziertes Problem ist z. B. das Reparieren eines defekten Fahrzeugs oder die Softwareverteilung in einem großen Unternehmen.
- *Komplexe Systeme* haben wie komplizierte Systeme viele Einflussgrößen, die stark miteinander verknüpft

sind. Allerdings sind diese Verknüpfungen einer hohen Dynamik unterworfen, d. h. sie verändern sich stark über die Zeit. Damit beginnt das System überraschende Eigenschaften zu entwickeln, die durch isolierte Betrachtung seiner Bestandteile nicht mehr erklärbar sind (Stichwort: Emergenz). Die Komplexität liegt nicht primär in den Bestandteilen des Systems, sondern in der dynamischen Interaktion zwischen den Teilen. Dennoch bilden sich im Gegensatz zu chaotischen Systemen (die hier nicht betrachtet werden sollen) Muster, die man erkennen muss, um Probleme in einer solchen Umgebung zu lösen. Man nähert sich der Lösung in (kurzen) Zyklen des Probierens, Bewertens und Korrigierens sukzessive an. Komplexe Systeme entstehen fast immer, wenn Menschen ins Spiel kommen. Ein Unternehmen ist z. B. ein komplexes System. Das Entwerfen eines neuen Autos (nicht nur die reine Konstruktion, sondern der gesamte Prozess) oder einer neuen Software sind ebenfalls (fast immer) komplexe Probleme.

Oft werden komplizierte und komplexe Probleme miteinander gleichgesetzt. Das kommt zum einen daher, dass die Begriffe im täglichen Sprachgebrauch häufig synonym verwendet werden. Zum anderen liegt es aber auch daran, dass komplizierte Probleme auf den Laien – und wir sind nun einmal Laien in allen Themen, die nicht zufällig unsere Fachgebiete sind – ähnlich wirken wie komplexe Probleme. Er erkennt die zugrunde liegenden Ursache-Wirkung-Prinzipien nicht und der Effekt ist, dass ihm das komplizierte Problem komplex erscheint.

Was haben einfache, komplizierte und komplexe Systeme denn mit der zuvor beschriebenen Aussage zu tun? Nun, ingenieursmäßiges Vorgehen basiert auf festen Ursache-Wirkung-Prinzipien und stabilen Teil-Ganzes-Beziehungen. Große Probleme werden nach dem Teile-und-Herrsche-Prinzip in kleine Probleme zerlegt, die in der Summe das große Problem ergeben. Komplizierte Lösungen werden nach klaren Regeln aus einfachen Lösungen zusammengesetzt. Damit ist ein ingenieursmäßiges Vorgehen dazu geeignet, einfache und auch komplizierte Probleme zu lösen. Es ist beeindruckend, welch hochkomplizierte Probleme die Ingenieurskunst gelöst hat, aber sie benötigt stabile Ursache-Wirkung-Zusammenhänge.

Komplexe Probleme hingegen mit ausgeprägter Dynamik, bei denen die Wechselwirkungen über die Zeit das Systemverhalten dominieren und nicht mehr die Eigenschaften der Einzelteile, kann man nicht ingenieurmäßig lösen. Komplexe Probleme kann man nur mit empirischen Vorgehensmodellen lösen, die auf kurzen Zyklen, schnellem Feedback, häufigen Reflektionen und kontinuierlicher Annäherung an die Lösung basieren.

Heutige Softwareentwicklung ist eine komplexe Aufgabenstellung. Genauer gesagt besteht sie aus einfachen, komplizierten und komplexen Anteilen. Die Komplexität ergibt sich alleine schon daraus, dass in der Regel viele Menschen in den Entwicklungsprozess involviert sind, vom Manager über Fachbereiche, Anwender, Projektleiter, Entwickler, Tester, Betrieb, Datenschutzbeauftragte, Betriebsrat usw., und sich dadurch ein komplexes System mit hoher, nicht vorhersagbarer Dynamik über die Zeit ergibt. Zusätzlich sind die genauen Anforderungen zu Beginn eines Projekts häufig noch unklar. Sie klären sich erst im Laufe des Projekts, und erste Lösungen haben dann wieder Rückwirkungen auf die Anforderungen. So ändern sich Anforderungen, weil man lernt, dass sie sich in der geplanten Form nicht umsetzen lassen, die ursprüngliche Anforderung nicht zum gewünschten Ergebnis führt, Konflikte zwischen verschiedenen Anforderungen aufgedeckt werden oder durch veränderte Rahmenparameter neue Anforderungen entstehen usw. Die Wechselwirkungen und Einflüsse sind hochdynamisch, kurzum: Softwareentwicklung ist heute in der Regel komplex.

KOMPLEXITÄT KANN MAN NICHT IGNORIEREN

Man kann versuchen, diese Komplexität zu verleugnen und so tun, als wären alle Probleme in der Softwareentwicklung höchstens kompliziert, getreu dem Motto „Es gibt kein Problem, das man per ‚Teile und Herrsche' nicht in den Griff bekommt". Die Fehlschlagstatistiken von IT-Projekten sprechen da aber eine andere Sprache. Massive Termin- und Budgetüberschreitungen, schlechte Qualität und unzufriedene Kunden sind das häufige Ergebnis, wenn man die Komplexität ignoriert und nur auf definierte Prozesse und ingenieursmäßiges Vorgehen setzt. Besser ist es, auf Vorgehensweisen zu setzen, die explizit mit der Komplexität umgehen. Die derzeit bekanntesten Vertreter dieser Kategorie sind die agilen Vorgehensweisen. Sie bündeln genau die Elemente, die man benötigt, um komplexe Probleme zu lösen: Kurze Zyklen, schnelles Feedback, häufige Reflektionen, kontinuierliche Annäherung an die Lösung, gepaart mit weiteren Erfolgsfaktoren für komplexe Umfelder wie viel Kommunikation, interdisziplinäre, sich selbst organisierende Teams, indirekte Führung, kontinuierliches Lernen usw.

Andere Branchen wie die Automobil- oder die Pharmaindustrie haben das übrigens schon lange verstanden: Komplexe Probleme wie das Entwickeln neuer Fahrzeuge oder neuer Medikamente werden dort schon lange mit agilen Methoden gelöst. Auch wenn es nicht immer explizit agil genannt wird, findet man an diese Stelle die ganzen zuvor beschriebenen Elemente der Agilität: kurze Zyklen, schnelles Feedback usw.

Tatsächlich stammen viele der Grundideen der heutigen Agilität nicht aus der Softwareentwicklung, sondern aus anderen Branchen wie Automobil-, Kopierer- oder Kamerabau, die Lösungen für die Komplexitätsprobleme in der Produktentwicklung gesucht haben [7]. Die agilen Grundideen lassen sich sogar bis in den militärischen Flugzeugbau der 50er Jahre zurückverfolgen [8], wo sie sehr erfolgreich angewendet worden sind. Vielleicht würde uns der in anderen Ingenieursbranchen schon lange selbstverständliche Umgang mit Agilität leichter fallen, wenn wir endlich akzeptieren, dass das Entwickeln und Schreiben des Sourcecodes nicht (mehr) zum Bauen, sondern zum Entwerfen der Lösung gehört. Und das ist wie in den anderen Branchen in der Regel eine komplexe Aufgabenstellung, für die man die angemessene Vorgehensweise – wie etwa Agilität – benötigt.

Ist damit ingenieursmäßiges Vorgehen in der Softwareentwicklung komplett passé? Nein, natürlich nicht. Wie bereits zuvor geschrieben, besteht Softwareentwicklung aus einfachen, komplizierten und komplexen Anteilen. Die einfachen und komplizierten Anteile sollten wir natürlich weiterhin mit ingenieursmäßigem Vorgehen lösen (bzw. ad hoc in ganz einfachen Fällen), denn da hilft uns Agilität nicht weiter. Das tun wir auch schon mit großem Erfolg, was aber nicht heißt, dass wir da nicht noch besser werden sollten. **Abbildung 3** verdeutlicht den Zusammenhang zwischen Aufgabenstellung und angemessener Methodik noch einmal.

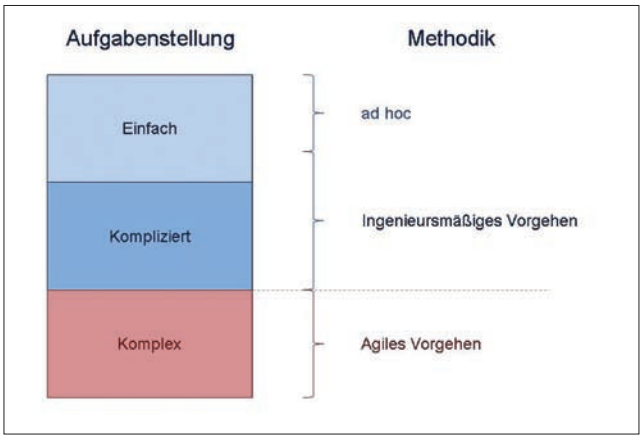

Abb. 3: Aufgabe und Methode

ZUSAMMENFASSUNG

Zusammenfassend lässt sich festhalten, dass das Implementieren von Software, d. h. das Entwickeln von Sourcecode, nicht Teil der Bauphase, sondern Teil der Entwurfsphase ist. Erst mit der letzten Zeile Sourcecode ist der Entwurf abgeschlossen. Die extrem niedrigen Kosten des eigentlichen Bauens von Software im Vergleich zu anderen Branchen führen häufig dazu, dass diese Phase komplett übersehen wird. Als Folge wird das Implementieren von Software mit der Bau- oder Produktionsphase in anderen Branchen verglichen, also sozusagen Äpfel mit Birnen. Dadurch hinken praktisch alle Vergleiche der Softwareentwicklung mit anderen Branchen sowohl in Bezug auf ingenieursmäßiges Vorgehen als auch in Bezug auf Agilität.

Weiterhin ist die heutige Softwareentwicklung eine komplexe Aufgabenstellung, die man mit klassischen ingenieurmäßigen Mitteln nicht in den Griff bekommen kann. Dafür benötigt man andere Mechanismen, wie sie z. B. die Agilität anbietet. Andere Branchen praktizieren das schon lange erfolgreich, indem sie z. B. die komplexe Produktentwicklung nach agilen Prinzipien organisieren, während sie den komplizierten Produktbau ingenieurmäßig optimieren. Agilität gibt es also sehr wohl in anderen Branchen, man muss nur an den richtigen Stellen suchen.

Damit stellt sich in der Softwareentwicklung nicht die Frage nach ingenieursmäßigem Vorgehen oder Agilität, sondern es ist klar, dass wir beides benötigen: Die einfachen und komplizierten Aufgabenstellungen sollten wir weiter ingenieurmäßig optimieren, während wir die komplexen Probleme mithilfe von Agilität lösen sollten. Grundsätzlich sind wir da aber schon auf einem guten Weg (wenn auch nicht immer aus den richtigen Gründen). Jetzt müssen wir nur noch mit den Glaubenskriegen aufhören und uns auf das Lösen unserer reichlich vorhandenen Herausforderungen und „Probleme“ mit den jeweils passenden Mitteln konzentrieren. Aber auch das werden wir noch schaffen...

Uwe Friedrichsen hat langjährige Erfahrungen als Architekt, Projektleiter und Berater. Aktuell verfolgt er als Leiter Competence Center Architektur bei der codecentric AG Software- und Unternehmensarchitekturen im Kontext agiler Konzepte.

Links & Literatur

[1] Agile Day auf der JAX 2010: http://it-republik.de/konferenzen/jax2010/session/?tid=1505
[2] Scrum: http://de.wikipedia.org/wiki/Scrum
[3] Extreme Programming: http://de.wikipedia.org/wiki/Extreme_Programming
[4] Ford, Neal: „Emergent Design & Evolutionary Architecture“, Vortrag bei der rheinjug: http://www.rheinjug.de/videos/gse.lectures.app/Talk.html#EmergingDesignRheinjug
[5] Snowden, D.J.; Boone, M.E.: „A Leaders's Framework for Decision making“, in Havard Business Review, November 2007
[6] Honegger, J.: „Vernetztes Denken und Handeln in der Praxis“, Versus Verlag Ag, Zürich 2008
[7] Takeuchi, H.; Nonaka, I.: „The new new Product Development Game“, in Havard Business Review, Januar-Februar 1986
[8] Bradshaw, Pete: „R&D Archaeology: The Lockheed Skunk Works“: http://valueshepherd.com/commentary/archaeology_skunk_works/archaeology_skunk_works.htm

codecentric AG
Merscheider Straße 1
42699 Solingen

Telefon: +49 (0) 212 - 233628 10
Telefax: +49 (0) 212 - 233628 79
E-Mail: info(at)codecentric.de