Winter 2010

SIGS DATACOM

SONDERBEILAGE AGILITÄT

www.OBJEKTspektrum.de

OBJEKTspektrum

Die Zeitschrift für Software-Engineering und -Management

- Agile IT-Geschäftsmodelle
- Agil vs. klassisch – Welcome to Reality
- Agile Entwicklung in verteilten Teams
- Agile Infrastruktur für Entwicklung und Testen

Liebe Leserinnen und Leser,

Agilität wird von einem Teil der Entwicklergemeinde als Freibrief, Regeln zu brechen, interpretiert. Aus diesem Grund glauben viele, dass Agilität nur in kleinen „Spielzeug“-Projekten funktioniert. Genau diese Sichtweise wollen wir in der vorliegenden Sonderbeilage mit dem Schwerpunkt „Agilität“ widerlegen.
Wir wünschen Ihnen beim Lesen der Fachbeiträge viel Freude und hoffen, Ihnen damit eine neue Perspektive auf das Thema zu eröffnen!

Ihr Helmut Mestrovic
Teamleiter IBM Rational Technical Sales
(helmut.mestrovic@de.ibm.com)

Ihre Susanne Herl
Redaktionsleitung OBJEKTspektrum
(susanne.herl@sigs-datacom.de)

Damit Sie das Agilitätskonzept auch in Zukunft weiterhin vor Augen haben, liegt diesem Sonderheft ein Poster „IBM agility@scale“ bei.

PS.: Wenn wir Ihnen mit diesem Sonderheft nicht alle Ihre Fragen beantwortet haben sollten, dann lassen Sie es uns gerne wissen!

Unter http://www.ibm.com/software/de/rational/agile/ können Sie weitere Informationen abrufen.

Agilität – kurzer Prozess mit dem Prozess

Als ich vor einigen Jahren das erste Mal von agil im Zusammenhang mit XP hörte, musste ich schmunzeln. Unwillkürlich musste ich an eines meiner ersten Projekte nach dem Studium denken. Ein Projekt, das vor der Zeit von Standard-Java-Applikationsservern eine tragfähige Middleware (nach dem CORBA-Standard) für Großkunden zum Ziel hatte. Damals war ich geneigt zu sagen: „Wir haben keinen Entwicklungsprozess". Allerdings wurden die Freiräume „ohne Prozess" zu arbeiten, von uns durch Kommunikation und Engagement gefüllt. Es hat funktioniert, wir waren aufeinander eingespielt, ein Team. Heute würde ich sagen: Wir hatten einen Prozess, auch wenn er nirgends niedergeschrieben war. Hat der Prozess funktioniert? Wenn man nach dem Resultat geht: Ja, die Software war nahezu fehlerfrei und lief ausfallsicher auf mehreren tausend Rechnern.

Das waren Zeiten: Ich erinnere mich noch gut an den Anruf eines Kunden, dessen Ansinnen ich heute mit der Terminologie eines „Change Requests" benennen würde. Damals war der Begriff für mich ein Fremdwort. Er wollte eine andere Farbe (was für ein Klischee, aber wahr!) für die Ausgabe eines Statustextes, innerhalb einer Systemmanagement-Console. Ich erklärte ihm, dass er nicht lange auf die Änderung warten müsse – nicht länger als 20 Minuten. Ich würde ihn dann zurückrufen.

Umgehend ging ich in den Nachbarraum. Ich wusste, dass dort ein Entwickler saß, der einen zu dem Kunden passenden Workspace hatte (passendes Betriebssystem, Compiler und fast passend die Version der Dateien). Erfreut war er zwar nicht, seine Arbeit unterbrechen zu müssen, aber unsere gegenseitige Hilfsbereitschaft war durch ein Rotationssystem, bei dem jeder Entwickler von Zeit zu Zeit vor Ort beim Kunden die neuesten Versionen und Module installieren musste, nahezu grenzenlos. Wer will schon riskieren, dass er abends um 8 Uhr mit einem wütenden Kunden alleine gelassen wird. Nein, bei so etwas hielten wir zusammen!

Wir haben also schnell die Arbeitsdateien mit einem Check-in gesichert (kein Problem, da es so etwas wie Nightly-Builds bei uns nicht gab) und quer durchs Büro gerufen, dass doch jetzt bitte niemand die aktuelle Version auschecken solle. Dank offener Türen war jeder in der Firma informiert.

Ab jetzt war alles einfach: Ins makefile geschaut, welche Dateien man für die Management-Console brauchte, entsprechende Dateien ausgecheckt, per „grep" die richtige Stelle gefunden, mit vi editiert, makefile gestartet und das Ergebnis per ftp zum Kunden geschickt.

Haben wir wenigsten einen Smoke-Test zuvor durchgeführt? Wozu, war ja nur eine Zeile, solche Änderungen trauten wir uns auch ohne Test zu. Die zwei Rechner, die uns in der neuen Testumgebung zur Verfügung standen (früher hatten wir noch an unseren eigenen Rechnern getestet), waren eh gerade belegt.

Minuten später hatte ich den Kunden am Telefon: Alles in Ordnung, es läuft! Und dann fragte er mich noch, warum das so lange gedauert habe? Eine Woche später war ich beim Kunden vor Ort und wir konnten bei einem Bierchen die ftp-Übertragungsrate als Schuldigen für die späte Lieferung identifizieren.Wenn das nicht agil war! Erkennt man doch das ein oder andere agile Prinzip (zum Beispiel „kundennah") oder die agile Methode (wie zum Beispiel „Paarprogrammierung") wieder. Aber ist es so einfach? Reicht es denn „ein wenig agil" zu sein und den Rest der Methoden und Prinzipien zu ignorieren?

Wie könnte die Geschichte wohl heute ausgehen? Der Kunde quält sich durch den Support und erhält zwei Wochen und zwölf Emails später die Auskunft: „Ist geplant für das Release in 18 Monaten." Übersetzt heißt das etwa: „Wir planen auf zwölf Monate und da ist die Änderung definitiv nicht enthalten".

War es damals besser? Auf den ersten Blick scheint es so. Doch zwei Wochen später hat ein anderer Kunde angerufen, der gerade ein neues Release mit Änderung bekommen hatte. Leider hat das Telefon nicht bei mir geläutet, sondern er hat gleich bei der Geschäftsleitung angerufen. Der aufgebrachte Kunde konnte einige Meldungen auf seinem Monochrom-Bildschirm nicht mehr lesen...

Wie gewohnt hat die Änderung (ein weiteres #IFDEF "<Kunde1>") nicht lange gedauert und der Kunde war in rekordverdächtiger Zeit mit einem Patch versorgt.

Bleibt zu erwähnen, dass ich mir von meinem Chef so einiges anhören durfte. In der Zeit hätte ich einige weitere Patches produzieren und verteilen können.

Sie fragen sich nun sicherlich: Wie konnten wir für so viele (ausgelieferte) Software-Versionen überhaupt den Support leisten? Die ehrliche Antwort: Mit viel Optimismus und noch mehr Glück!

Bedeutet agil nun wirklich, dass man sämtliche Regeln aufgibt? Und wenn ja, wer übernimmt die Verantwortung für nicht bedachte Seiteneffekte (Monochrom-Bildschirm!). Der Kunde, die Firma oder gar der Entwickler? Und was bedeutet das für eine sicherheitskritische Anwendung? Funktioniert agil dann nicht?

Entwarnung! Agil bedeutet nicht die Aufgabe sämtlicher Regeln, daher überrascht es auch nicht, dass es für agile Prozesse tatsächlich formale Prozessbeschreibungen gibt.

Was bringt die agile Arbeitsweise Ihrem Unternehmen? Was bedeutet es in Bezug auf Organisationsstrukturen? Wie agil darf im Projekt gearbeitet werden? Wie formal muss das Team arbeiten? Mit vorliegendem Heft wollen wir Ihnen bei der Beantwortung dieser und weiterer Fragen rund um das Thema agil helfen. Denn eines ist sicher: Ins nächste Zimmer gehen zu müssen, um einen passenden Workspace zu finden oder in der Gegend rumzubrüllen, um Informationen mit dem Team auszutauschen, das ist nicht mehr zeitgemäß. Es gibt elegantere, effizientere und vor allem zielführendere Wege, den Projekterfolg zu gewährleisten.

Helmut Mestrovic

Teamleiter IBM Rational Technical Sales
(helmut.mestrovic@de.ibm.com)

Agilität als Schlüssel für Wachstum

Unternehmen erleben einen tiefgreifenden Wandel

Unternehmen sind dabei oder bereiten sich darauf vor, aus einer globalen Rezession herauszukommen und richten ihr Augenmerk wieder auf Wachstum. Die zunehmende Vernetzung von Volkswirtschaften, Unternehmen, Gesellschaften und Regierungen hat zwar zu immensem Chancenreichtum geführt, doch die Suche nach neuen Wachstumsfeldern ist nicht einfach in einem Umfeld, das von unzähligen einzelnen Märkten, immer mehr Produkt- und Servicekategorien und immer stärker individualisierten Kundensegmenten geprägt ist. Die weltweit über 1.500 in der IBM CEO Studie 2010 [IBMCEO10] befragten Führungskräfte sind sich einig (siehe Abb. 1): Die neue Wirtschaftswelt wird sehr viel dynamischer, ungewisser und komplexer als je zuvor.

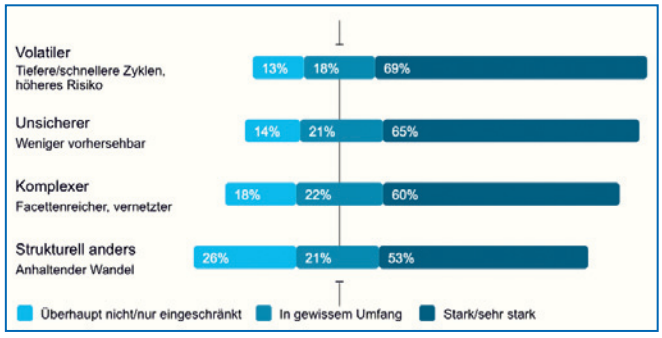


Abb.1 Das neue wirtschaftliche Umfeld wird ... (Quelle: IBM CEO Studie 2010)

Eines ist klar: Gefahren und Chancen kommen nicht nur schneller auf uns zu und sind weniger vorhersehbar, sondern gehen auch ineinander über und beeinflussen sich gegenseitig – und daraus entstehen völlig neue Situationen. Für die Unternehmen bedeutet dies, dass sie ihr Portfolio, ihr Geschäftsmodell, ihre Arbeitsweise und ihre lang gehegten Meinungen grundlegend überdenken und verändern müssen, um die weiter zunehmende Komplexität erfolgreich zu meistern. Dabei räumen 36 Prozent der befragten deutschen Führungskräfte ein, dass sie ihr Unternehmen nur unzureichend für die kommenden Herausforderungen gerüstet sehen. Um im internationalen Wettbewerb nicht ins Hintertreffen zu geraten, muss gehandelt werden.

Erfolgreiche Unternehmen zeigen den Weg zum Erfolg

Wie – das zeigen Unternehmen, die über Jahre solide Geschäftsergebnisse erzielt haben, selbst in der letzten Rezession. Laut Erkenntnissen der IBM CEO Studie 2010 konzentrieren sich diese herausragenden Unternehmen auf drei Bereiche.

1. *Kreativität zur zentralen Fähigkeit entwickeln:* Führungskräfte herausragender Unternehmen heißen revolutionäre Innovationen willkommen, ermutigen andere, ausgetretene Pfade zu verlassen, und gehen kalkulierte Risiken ein. Sie sind offen und einfallsreich, wenn es um die Ausweitung ihres Führungs- und Kommunikationsstils geht, vor allem im Austausch mit einer neuen Generation von Mitarbeitern, Partnern und Kunden.
2. *Eine neue Qualität der Kundenbeziehung erreichen:* Herausragende Unternehmen räumen der Kundennähe einen höheren Stellenwert ein als je zuvor und berücksichtigen stärker, was Kunden heute wichtig ist. Ständige Interaktion mit Kunden und deren Mitwirkung an der Entwicklung von Produkten und Services tragen zur Differenzierung gegenüber Mitbewerbern bei. Die „Informationsexplosion" wird nicht als Problem, sondern als große Chance genutzt, wenn es darum geht, profunde Erkenntnisse über Kunden zu gewinnen.
3. *Operative Agilität entwickeln:* Erfolgreiche Unternehmen setzen auf stärkere Veränderungsbereitschaft, radikale Vereinfachungen, variable Kostenstrukturen, flexible Sourcing-Modelle mit verstärkter Zusammenarbeit mit Partnern, sowie rasche Entscheidungs- und Umsetzungsprozesse. Dabei kommt es vor allem auf Schnelligkeit an. Es genügt nicht mehr, auf der Basis traditioneller Zeitrahmen oder strategischer Planungszyklen zu denken, zu führen oder zu delegieren. Sowohl neue Risiken als auch neue Chancen erfordern die Fähigkeit, Ergebnisse soweit wie möglich vorherzusagen, zu handeln, auch wenn ein Rest Ungewissheit bleibt, und dann noch erforderliche Kurskorrekturen vorzunehmen.

IT ist mitentscheidend

Die operative Agilität eines Unternehmens hängt entscheidend von den Fähigkeiten seiner IT ab. Flexible, schnell an geschäftliche Veränderungen und technologische Innovationen anpassbare IT-Lösungen für eine erhöhte Automation der Prozesse (Stichwort: „Business-Process-Management"), die zuverlässige Analyse und Interpretation von Daten für mehr Klarheit und Gewissheit bei der Entscheidungsfindung (Stichwort: „Business Analytics & Intelligence") und die optimierte Zusammenarbeit der Mitarbeiter untereinander und mit Kunden und Zulieferern (Stichwort: „Web 2.0, Social Computing") sind unentbehrlich für agile Geschäftsprozesse. Um dies optimal zu realisieren, wird für die IT selbst ein agiles Geschäftsmodell benötigt, gekennzeichnet durch die folgenden Merkmale (siehe Abb. 2).

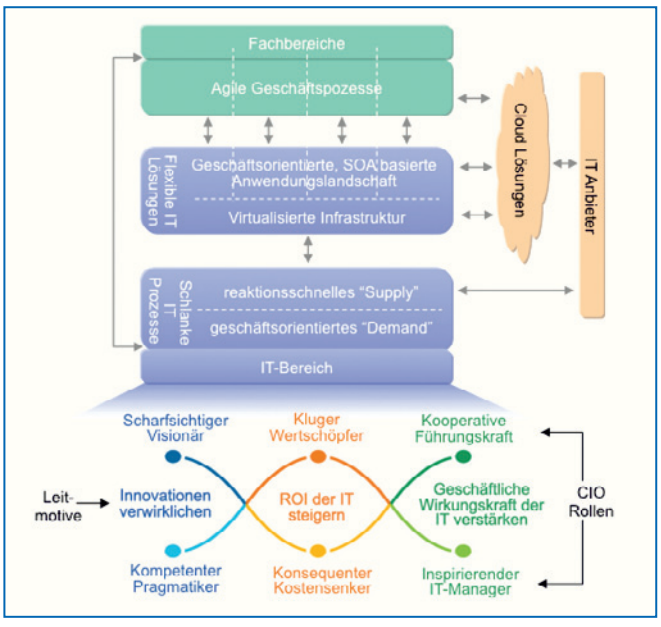


Abb.2 Merkmale eines agilen IT Geschäftsmodells

- Ausrichtung des IT-Bereichs auf die drei Leitmotive „Innovationen verwirklichen“, „ROI der IT steigern“ und „geschäftliche Wirkungskraft der IT verstärken“: Diese Themen spiegeln sich in drei Rollenpaaren wider, die der CIO und seine Organisation gleichzeitig vereinen muss. Der scharfsichtige Visionär erkundet für das Unternehmen, auf welche Weise Technologie Innovationen vorantreiben kann, während der kompetente Pragmatiker zukunftsweisende Pläne in die Tat umsetzt. Der kluge Wertschöpfer versteht die Bedürfnisse der Fachbereiche und entwirft daraufhin bessere Lösungen, während der konsequente Kostensenker stets darauf achtet, Ausgaben, wo möglich, zu reduzieren. Die kooperative Führungskraft hat ein umfassendes Verständnis für das Kerngeschäft des Unternehmens und bildet intern wie extern starke Partnerschaften. Der inspirierende IT-Manager zeichnet sich dagegen durch hohe persönliche IT-Kompetenz aus und fördert ausgeprägte Fachkenntnisse innerhalb des IT-Bereichs. Die IBM CIO Studie 2009 [IBMCIO09] belegt, dass diejenigen Unternehmen ein höheres wirtschaftliches Wachstum erreicht haben, die sich besser auf diese drei Leitmotive ausgerichtet und die scheinbar widersprüchlichen Rollen vereint haben, als andere.
- Schlanke, standardisierte IT-Prozesse nach einem „Demand & Supply“-Modell: Durch eine enge Verzahnung und ständige, vereinfachte Interaktionen zwischen IT- und Fachbereichen in den „Demand“-Prozessen werden die Kunden der IT besser in die Neu- und Weiterentwicklung von benutzerfreundlicheren IT-Lösungen integriert. Dabei geht es für die IT über das reine Erfassen und Beantworten von Anforderungen hinaus. Durch verstärktes betriebswirtschaftliches Verständnis werden Fachbereiche aktiv beraten und auf alternative Handlungsmöglichkeiten zur Erreichung von deren Zielen hingewiesen. Im Bereich der „Supply“-Prozesse (d.h. Implementierung, Betrieb und Support von Anwendungen und Infrastruktur) ist es für die Agilität der IT besonders wichtig, unnötige Komplexität zu vermeiden und schnell auf Veränderungsbedarf zu reagieren. Schnelles Handeln trotz gewisser Ungewissheiten ist hier die Leitlinie. Damit dies jedoch nicht im Chaos endet, sorgen iterative und inkrementelle Vorgehensmodelle, agile SW-Entwicklungsmethoden wie „Extreme Programming (XP)“, durchgängige Werkzeugketten für die Verfolgbarkeit von Ergebnissen und für generative Ansätze, aus einer Unternehmensarchitektur abgeleitete Architekturprinzipien, sowie eine klar nach Kompetenzbereichen aufgestellte IT-Organisation für die wesentlichen Eckpfeiler. Sowohl für „Demand“ wie für „Supply“ gilt es, durch die richtige Balance aus globaler und lokaler Ausrichtung und Aufstellung entsprechende Vorteile zu ziehen, nach dem Motto: Global, wo möglich und lokal, wo nötig.
- Klar am Geschäft ausgerichtete Anwendungslandschaft auf Basis einer service-orientierten Architektur (SOA) mit virtualisierter IT-Infrastruktur: Die individuelle Ausprägung der Geschäftsprozesse im Unternehmen (von „Commodity“ bis zu sehr differenzierend / hochspezialisiert, von stabil bis zu sehr flexibel /dynamisch) ist der wesentliche Treiber für die Gestaltung der Anwendungskomponenten. Hier gilt es, in der Balance mit Vorgaben aus der IT-Strategie zu den eingesetzten Technologien, die richtige Mischung aus fertigen Komponenten vom Markt, eingesetzt „as-is“ oder minimal konfiguriert, und individuell entwickelten Komponenten zu etablieren. Das richtige Maß für Modularisierung und Isolierung auf der einen Seite und Integration über Standardschnittstellen auf der anderen Seite sind wichtige Prinzipien, um die notwendige Agilität zu erzielen. Die Virtualisierung der Infrastruktur in der ganzen Bandbreite über Clients, Server und Storage sorgt für die Entkopplung von den Anwendungen und ermöglicht deutlich dynamischere Anpassungen bei Veränderungen agiler Geschäftsprozesse (zum Beispiel die Anpassung der Lastverteilung aufgrund sich schnell ändernder Benutzerzahlen oder von flexiblen geschäftlichen Abläufen).
- Variable, partnerschaftliche Sourcing-Modelle: Handeln im Alleingang ist nicht mehr adäquat. Die Kompetenz und Agilität der IT im Unternehmen wird deutlich gestärkt durch die Einbindung und enge Zusammenarbeit mit Partnern, die ausgewählte IT-Leistungen und Lösungen flexibler, reaktionsschneller und kosteneffizienter anbieten können. Neben den klassischen Möglichkeiten - Nutzung externer Dienstleistungen und Kauf von Anwendungs- und Technologiekomponenten - ermöglicht „Cloud Computing“ neue Perspektiven durch die direkte, rasche und variable Nutzung von bereits vorhandenen IT-Lösungen.

Operative Agilität zahlt sich aus

Operative Agilität kombiniert mit Kreativität und ausgezeichneten Kundenbeziehungen – im Geschäft und darauf abgestimmt in der IT – versetzen Unternehmen in die Lage, sich bei Bedarf schnell und gezielt anzupassen, um Wachstumschancen erfolgreich zu verfolgen. Laut der IBM CEO Studie zahlt sich das aus: Agiler agierende Unternehmen konnten in der Vergangenheit mehr Umsatz aus neuen Quellen generieren und erwarten dies auch für die Zukunft (siehe Abb. 3)

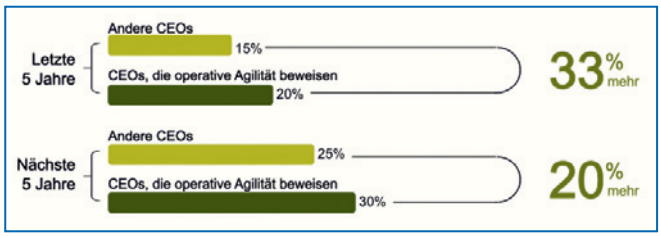


Abb.3 Umsatz aus neuen Quellen durch operative Agilität
(Quelle: IBM CEO Studie 2010)

Weiterführende Links:

[IBMCEO10] IBM CEO Studie 2010, siehe:
www.ibm.com/services/de/ceo/ceostudy2010/

[IBMCIO09] IBM CIO Studie 2009, siehe:
www.ibm.com/services/de/cio

Jürgen Henn

Partner und Leiter für IT Strategieberatung bei IBM Global Business Services
(juergen.henn@de.ibm.com)

Agile Softwareentwicklung im Kontext unternehmensweiter IT-Prozesse

Agile Vorgehensweisen im Softwareentwicklungslebenszyklus

Agile Ansätze beschränken sich typischerweise auf einen Ausschnitt des gesamten Softwareentwicklungslebenszyklus und beschreiben die Erstellung von Software primär aus der Perspektive der Entwickler. Sie berücksichtigen dabei auch die Erarbeitung der Anforderungen und die zugehörige Kommunikation zwischen Fachexperten und Entwicklern, um ein gemeinsames Verständnis der Lösung sicherzustellen.

Im Unterschied zu anderen, weniger agilen Vorgehensweisen wird hier der Nutzen direkter Kommunikation und früher Erstellung von funktional unvollständigen, aber bereits ausführbaren und wertbringenden Lösungsversionen hervorgehoben. Die agile Vorgehensweise endet in der Regel mit der Bereitstellung einer vom Auftraggeber für gut befundenen Software-Einheit zur Überführung in die Produktion. Unter den umfassenden IT-Prozessen eines Unternehmens konzentrieren sich die agilen Ansätze auf die direkte Transformation von fachlichen Anforderungen in auslieferbare Software. Ziel ist es, das Risiko der Erstellung einer am Bedarf der Benutzer vorbei entwickelten Software zu minimieren und den mit den verfügbaren Ressourcen erzielbaren geschäftlichen Nutzen zu maximieren.

Das Erarbeiten der (potenziell) auslieferbaren Software-Stände ist bei den agilen Vorgehensweisen teilweise strikt geregelt, wie man etwa bei Scrum in der Organisation der Sprints erkennt. Hierzu gehört auch die Verteilung einzelner Arbeitspakete an Teammitglieder sowie die Feststellung des erzielten Fortschritts. Dies lässt sich als „inneres Projektmanagement" bezeichnen. Im Gegensatz zum klassischen Projektmanagement wird dieses „innere Projektmanagement" vom Team und nicht von einer dedizierten Projektleiter-Rolle betrieben.

Schnittstellen zum Projektmanagement

Neben dem „inneren" Projektmanagement sehen die Controlling-Mechanismen größerer Unternehmen stets ein formales, weniger inhaltlich getriebenes „äußeres Projektmanagement" vor. Dort werden unter anderem verbrauchte und verbliebene Aufwände, der Finanzstatus und der Ampelstatus in Bezug auf die Plantreue des Projekts gemeldet und vor entsprechenden Gremien vertreten. Zu diesem äußeren Projektmanagement zählen auch die meisten Tätigkeiten der Projektinitialisierung, wie zum Beispiel das Einrichten von Buchungspositionen zur Erfassung geleisteter Aufwände.

Auch agile Projekte lassen sich sehr einfach in die Projektsteuerung des Unternehmens einbetten: Ein Projektleiter, der nicht Mitglied des agilen Teams ist, übernimmt den äußeren Teil des Projektmanagements. Er wird dabei regelmäßig mit ohnehin anfallenden Statusinformationen (zum Beispiel die Ergebnisse des Sprint Review Meetings, Burndown Charts) oder – wenn entsprechende Tools zum Einsatz kommen – durch automatisierte Reports versorgt. Dadurch kann er die notwendigen Berichte erstellen und den Fortschritt darstellen. Seine Sicht auf das agile Projekt ist der Projektplan mit den geplanten Meilensteinen, synchronisiert mit den Sprints des agilen Projekts. Die Verteilung der Aufgaben und die Gestaltung der Vorgehensweise innerhalb des Projekts verbleiben bei dem Team – der agile Kern bleibt unberührt.

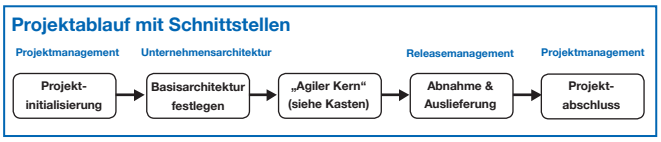


Abb. 1 Einbettung des agilen Kernprozesses in ein erweitertes Modell zur Projektdurchführung

Schnittstellen zum Architekturmanagement

Eine weitere Schnittstelle zu den unternehmensweiten Prozessen ergibt sich durch das übergreifende Architekturmanagement. Dort werden die für die IT geltenden Standards wie zum Beispiel Basistechnologien, Produkte oder Grobarchitekturen definiert und deren Einhaltung in Projekten und anderen IT-Maßnahmen geprüft. Häufig wird dieser Eingriff in die „technische Autonomie" von Projekten gerade von agilen Entwicklungsteams sehr kritisch gesehen, fühlen sie sich doch ganz der bestmöglichen Lösung aus Sicht ihres Kunden verpflichtet.

Dies muss nicht in jedem Fall mit der Technologieauswahl übereinstimmen, welche die Unternehmensarchitektur auf der Basis berechtigter anderer Kriterien trifft. Um hier mögliche Konflikte frühzeitig zu erkennen und zu lösen, ist die Festlegung auf eine Basisarchitektur zu Beginn des Projekts empfehlenswert, am besten unter direkter aktiver Beteiligung eines Unternehmensarchitekten. Beim Scrum-Vorgehen kann man eine kurze Phase mit wenigen, schnellen Explorationssprints („Architektursprints") vornehmen. Deren primäres Ergebnis ist zwar weniger direkt für den Benutzer des Systems nutzbar, legt aber die technische Basis der Lösung fest. Dazu können aus den initialen fachlichen Anforderungen technische Stories abgeleitet werden, deren Umsetzung in Form (evolutionärer) Prototypen die Auswahl und Tauglichkeit einer technischen Architektur nachweisen. Der beteiligte Unternehmensarchitekt soll dabei die Konformität zur Unternehmensarchitektur sicherstellen und gegebenenfalls die bestehenden Standards und Richtlinien erweitern. Eine gesonderte, nachgelagerte Abnahme der Basisarchitektur ist bei dieser Arbeitsweise nicht nötig. Gegebenenfalls erforderliche Änderungen der Basisarchitektur im Projektverlauf sollten immer direkt mit der Unternehmensarchitektur abgestimmt werden, dann ist die Bestätigung der Architekturkonformität des Projektergebnisses durch die Unternehmensarchitektur nur noch eine Formsache. Auf diese Weise lässt sich die Unternehmensarchitektur organisch in den agilen Entwicklungsprozess einbeziehen, ohne den agilen Charakter des Projekts zu verfremden.

Schnittstellen zum Change- und Release-Management

Die dritte Schnittstelle zu unternehmensweiten Prozessen, nämlich die zum übergreifenden Change- und Release-Management, ist unproblematisch - es gibt auch bei agilen Projekten einen mit dem Product-Owner abgestimmten Projekt-Release-Plan, der in den allgemeinen Release-Kalender einfließt. Der Projektleiter kann die notwendige Kommunikation zum Stand

des Release-Fortschritts im Projekt zum Verantwortlichen (Release Owner) des angestrebten Release als Teil der Kommunikation nach außen leisten. Der abschließende Release-Test, der über den Rollout des gesamten Release-Pakets inklusive des Projektergebnisses entscheidet, ist nicht mehr Aufgabe des Projekts, sondern wird vom Release Owner organisiert.

Organisationsschnittstellen

Mit den IT-Prozessen eines Unternehmens werden die beteiligten Rollen definiert, die zur Ausführung der Prozesse benötigt werden. Die Beschreibung dieser Rollen besteht aus drei Komponenten: Skill, Aufgaben und Arbeitsergebnisse. Rollen werden hauptsächlich aus zwei Gründen definiert: Zur Skill-Differenzierung und wegen der eindeutigen Verantwortlichkeit für Aufgaben und Ergebnisse. Die Unternehmen definieren im Zuge der Modellierung ihrer IT-Prozesse eine Reihe standardisierter Rollen vor, die dann bei der Durchführung der Prozesse konkret mit Mitarbeitern besetzt werden, die die entsprechenden Skill-Profile aufweisen. Häufig übernimmt ein Mitarbeiter bei der Durchführung eines Prozesses mehrere Rollen.

Agile Vorgehensweisen, insbesondere Scrum, stellen das (Entwicklungs-)Team und nicht einzelne Rollen in den Vordergrund. Diese Sicht wird auch durch das agile Grundprinzip des gemeinsamen Code-Besitzes (Collective Code Ownership) unterstützt. In der Beschreibung eines agilen Vorgehensmodells tauchen daher auch nicht die sonst üblichen Rollen wie zum Beispiel Anwendungsentwickler, Software-Konfigurationsmanager oder Lösungsarchitekt auf, sondern bei entsprechenden Aufgaben eben das Entwicklungsteam. Um hier dennoch eine weitgehende Kohärenz der Rollen auf Unternehmensebene zu gewährleisten, kann man das Teammitglied („Team Member“) als „Sammelrolle“ einführen. Bei der Beschreibung agiler Prozesse lässt diese sich wie eine herkömmliche Rolle einsetzen, während die Definition dieser Rolle auf die relevanten „klassischen“ Rollen verweist und im Wesentlichen verlangt, dass in konkret zu besetzenden Teams insgesamt der Skill dieser referenzierten Rollen in hinreichendem Umfang (das heißt ohne personelle Engpässe in der Projektdurchführung) vorhanden ist. Dadurch wird die Rollenbeschreibung agiler und traditioneller Ansätze durchgängig.

Dokumentation der Software

Das agile Manifest, das den Kodex agiler Softwareentwicklung darstellt, enthält den Passus

> *„Funktionierende Programme sind wichtiger als ausführliche Dokumentation.“*

Dieses eingängige Prinzip führt dazu, dass die Dokumentation zu agil entwickelter Software häufig nicht in der sonst üblichen Form vorliegt, wie zum Beispiel als Fach- und DV-Konzepte.

Trotzdem müssen auch in agilen Projekten zwei Aspekte berücksichtigt werden:

- Regulatorische Anforderungen können nur durch die hinreichende Dokumentation der Anforderungen und Systeme erfüllt werden.
- Falls das ursprüngliche Entwicklungsteam ganz oder in wesentlichen Teilen nicht mehr zur Verfügung stehen sollte, müssen bislang unbeteiligte Entwickler die Chance haben, ein rasches Verständnis des Systems zu erlangen – dazu ist eine alleinige Analyse des Source Codes zum Einstieg nicht immer das geeignete Instrument.

Eine angemessene Dokumentation kann durch Aufwertung und Aufbereitung vorhandener Artefakte des agilen Vorgehens erzielt werden:

Bei Scrum eignet sich das Product Backlog mit den enthaltenen User Stories als Darstellung eines Fachkonzepts. Voraussetzung hierfür: Die User-Stories müssen so abgelegt sein, dass sich daraus jederzeit ein Dokument (automatisiert) erstellen lässt. Die Inhalte der User Stories lassen sich durch entsprechend dokumentierte Unit Tests ergänzen.

Auf der technischen Seite werden bei agilem Vorgehen meist drei Aspekte dokumentiert: Der Source Code, die Tests sowie die Systemarchitektur nebst wesentlichen übergreifenden technischen Konzepten. Wenn entsprechende Vorgaben für diese Dokumentation definiert sind und eingesetzt werden, kann in Summe eine hinreichende Systemdokumentation vorliegen. Wichtig ist dabei, dass dieses Ziel dem Entwicklungsteam von Anfang an klar ist und sie entsprechende technische Mechanismen (zum Beispiel Javadoc bei Source-Code) einsetzen, die eine Extraktion der Inhalte in eine zentral verfügbare Dokumentation erlauben.

Zusammenfassung

Agile Vorgehensweisen lassen sich gut mit unternehmensweit gültigen IT-Prozessen in Einklang bringen, wenn die relevanten Schnittstellen berücksichtigt werden:

- Einbettung der agilen Vorgehensweise in ein erweitertes Vorgehensmodell mit früher Einbindung von Architekturmanagement, Release-Management und „äußerem“ Projektmanagement
- Widerspruchsfreies, durchgängiges Rollenkonzept durch Deklaration des Team Members als „Sammelrolle“
- Umsetzung von Dokumentationsstandards durch (werkzeugunterstützte) Aufbereitung agiler Artefakte. ■

Dr. Joachim Herzog

Senior Managing Consultant. Langjährige Erfahrung zur Prozessoptimierung in der Anwendungsentwicklung, Design und Einführung von Prozessen und Methoden, z. B. nach dem Software Process Engineering Metamodel (SPEM) und mit diversen Werkzeugen.
(joachim.herzog@de.ibm.com)

Günter Holzmüller

Senior Managing Consultant. Langjährige Erfahrung in der Durchführung von IT-Transformationsprogrammen. Schwerpunkte: Prozessmodellierung mit Rational Method Composer, Prozessmanagement, Kontinuierlicher Verbesserungsprozess, Prozessschnittstellen, Entwicklung und Einführung von Kennzahlensystemen.
(hlz@de.ibm.com)

Werner Schoepe

Field Technical Professional. Langjährige Erfahrung bei der Einführung von Rational Softwareentwicklungswerkzeugen. Aufgabenschwerpunkte: Application Lifecycle Management, Softwareentwicklungsprozesse, agile Softwareentwicklung
(werner.schoepe@de.ibm.com)

Agil(e) mit RUP Gegensatz oder ideale Verbindung?

Überblick RUP

Der Rational Unified Process (RUP) wurde erstmals 1999 von der Firma Rational veröffentlicht. Dabei war er keine komplett neue Erfindung, sondern vielmehr eine systematische Zusammenfassung vieler altbewährter Methoden und Best Practices, die schon seit mehr als zehn Jahren praktisch genutzt wurden. Der RUP wird bis heute wie ein Produkt von einem eigenen Entwicklerteam weiterentwickelt, zunächst bei der Firma Rational, später bei IBM, und ist auch dementsprechend lizenzpflichtig. Mittlerweile gibt es aber viele Varianten, unter anderem auch die Open-Source-Variante OpenUP, die im Eclipse Projekt zu finden ist ([Ecl]).

Dabei ist der RUP nicht als fest definierter Prozess zu verstehen, sondern vielmehr als umfangreiches und detailliertes Prozess-Framework, das an die konkreten Unternehmens- und Projektgegebenheiten, primär durch Kürzen, anzupassen ist. Dazu dient in erster Linie der IBM Rational Method Composer, eine Process-Authoring-Werkzeug, das zusammen mit den RUP-Inhalten ausgeliefert wird.

Die Inhalte des RUP enden an den Projektgrenzen, gehen also nicht auf unternehmensweite Prozesse ein.

Die Abbildung zeigt die zwei Dimensionen des RUP im Überblick: Disziplinen (Inhalte und Vorgehensweisen) und Phasen (zeitlicher Verlauf).

Als Kriterium, welche konkrete Prozessinstanz noch im Sinne des RUP definiert ist, können folgende RUP-Kennzeichen dienen (vgl. [Jac99]):

- *Use Case Driven:* Anforderungen, Design, Implementierung, Test und Dokumentation gehen von Use Cases aus.
- *Architecture Centric:* Eine früh stabilisierte Softwarearchitektur ist wichtige Voraussetzung für Projektmanagement, Design und Implementierung.
- *Iterative and Incremental:* Jede Phase wird in ein oder mehrere Iterationen unterteilt. (Fast) jede Iteration durchläuft alle Disziplinen und liefert ein Inkrement von ausführbarem und getestetem Code.

Gemeinsamkeiten mit agilen Methoden

Aufgrund seines umfangreichen und detaillierten Inhalts und dem Missverständnis, dass dieser genau so auszuführen ist, wie er im Framework beschrieben wurde, wird der RUP – zu Unrecht – häufig als schwerlastig bewertet. In Wirklichkeit hat er mehr Gemeinsamkeiten mit agilen Prozessen, als viele wissen. Ein Vergleich mit den Werten des agilen Manifests (vgl. [Agil]) zeigt:

- Individuen und Interaktionen haben Vorrang vor Prozessen und Tools: Dies ist in der Tat die Forderung, in der sich agile Methoden und RUP unterscheiden (aber nicht unbedingt ausschließen. Beim RUP wird über erarbeitete Inhalte, die in Werkzeugen hinterlegt sind, kommuniziert.
- Funktionierende Software hat Vorrang vor umfangreicher Dokumentation: Die wichtigste Bewertung für den Fortschritt im RUP ist die funktionsfähige, getestete Software am Ende jeder Iteration. Dokumentation soll hier auch nur so viel und so formal wie unbedingt nötig erzeugt werden.
- Zusammenarbeit mit dem Kunden hat Vorrang vor Vertragsverhandlungen: Grundidee des RUP ist ebenso dem Kunden so schnell und so oft wie möglich Ergebnisse vorzustellen, damit mit ihm immer wieder geklärt werden kann, ob es das ist, was er wirklich will. So kann man auf Abweichungen reagieren.
- Das Eingehen auf Änderungen hat Vorrang vor Planverfolgung: Im RUP ist das Änderungsmanagement während des laufenden Projekts ein wesentlicher Bestandteil. Eine Gesamtplanung soll ohnehin nur grob erfolgen und darf geändert werden. Lediglich die jeweils nächste Iteration wird detaillierter geplant.

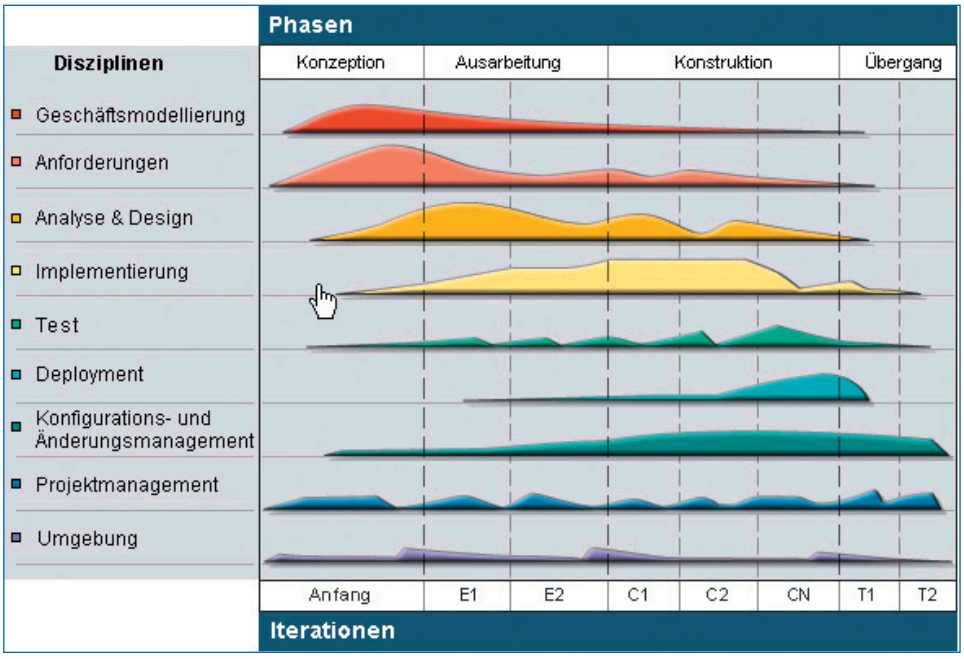


Abb. RUP Übersicht – Disziplinen und Phasen

Zusammengefasst (Kasten 1 listet die Abdeckung der agilen Prinzipien durch den RUP) erfüllt der RUP drei von vier Werten des Agile Manifesto. Dies ergibt sich primär durch die iterative Vorgehensweise, wie sie auch die meisten agilen Methoden befolgen. Lediglich der menschliche Faktor und die direkte Kommunikation zwischen Menschen finden im RUP keine große Erwähnung. Die Kommunikation erfolgt hier im Wesentlichen über Informationen (häufig graphische Modelle mit der Unified Modeling Language UML [UML]), die in Werk-

Abdeckung der agilen Prinzipien durch den RUP

+ = wird durch den RUP erfüllt
* = keine Festlegung durch den RUP
- = trifft für den RUP eher nicht zu

+ Die höchste Priorität besteht darin, den Kunden zufrieden zu stellen; dies durch frühe und regelmäßige Lieferung von Software, die einen Mehrwert bringt.
+ Änderungen an Anforderungen sind willkommen, auch spät in der Entwicklung. Agile Prozesse nutzen Änderungen, um Kunden einen Wettbewerbsvorteil zu verschaffen.
+ Regelmäßige Auslieferung lauffähiger Software in Intervallen von wenigen Wochen bis Monaten – bevorzugt wird eine kürzere Zeitspanne.
+ Lauffähige Software ist das primäre Maß für Fortschritt.
+ Fortlaufender Fokus auf technische Exzellenz und gutes Design steigern die Agilität.
+ Einfachheit – die Kunst den Teil der Arbeit zu maximieren, der nicht getan werden muss – ist wesentlich.
+ In regelmäßigen Abständen prüft das Team, wie es effektiver werden kann und passt dann sein Verhalten entsprechend an.
* Projekte bauen auf motivierte Individuen auf, es gilt der Grundsatz: Gib ihnen die Umgebung und die Unterstützung und vertraue darauf, dass sie die Arbeit richtig erledigen.
* Agile Prozesse fördern nachhaltige Entwicklung. Die Sponsoren, Entwickler und Anwender sollten eine gleichmäßige Arbeitsgeschwindigkeit beibehalten können.
* Die besten Architekturen, Anforderungen und Designs kommen aus Teams, die sich selbst organisieren.
- Fachleute und Entwickler müssen täglich zusammenarbeiten – über das gesamte Projekt hinweg.
- Die effizienteste und effektivste Methode, Informationen an und innerhalb des Entwicklungsteams zu verbreiten, ist das direkte Gesprächtätsregeln können komplexe Regeln auch auf Basis einer Rules Engine (iLog) definiert werden.

Kasten 1: Abdeckung der agilen Prinzipien durch den RUP

zeugen erarbeitet und abgelegt werden. Menschen werden im RUP lediglich über Rollen abgebildet, die am Prozess teilhaben.

Schwächen bekannter agiler Methoden

Wenn in diesem Abschnitt Schwächen agiler Methoden genannt werden, so soll das nicht darüber hinwegtäuschen, dass agile Methoden ihrerseits Antworten auf Schwächen anderer Vorgehensweisen (auch des RUP) haben, wie zum Beispiel mehr auf den Menschen im Projekt einzugehen.

Hier sollen nur exemplarisch einige wichtige Schwächen der zwei bekanntesten agilen Methoden angeführt werden. Größtenteils kommen die Lücken auch aus den Notwendigkeiten komplexerer Projekte und der Einbettung in die unternehmensweite IT (vgl. Artikel „Agile Softwareentwicklung im Kontext unternehmensweiter IT-Prozesse“ [Her10]).

Extreme Programming (XP)

Da es viele verschiedene Quellen zum XP gibt (z.B. [Wiki]), die teils versuchen unten genannte Lücken zu ergänzen (allerdings ohne offiziellen Charakter), wird hier als einzige gültige Quelle [XP] genutzt.

- *Projektmanagement:* Das Projektmanagement beschränkt sich im Wesentlichen auf die Release- und Iterationsplanung bezüglich Zeit und Umfang. Viele Bereiche des Projektmanagements, wie Risikomanagement und Finanzen (Changemanagement ist angedeutet enthalten), fehlen.
- *Anforderungen:* Das Anforderungsmanagement erscheint auf den ersten Blick zwar logisch und sinnvoll, weist aber bei genauerer Betrachtung Schwächen auf:
  - » Es gibt keinen systematischen Ansatz dafür, alle Stakeholder in die Anforderungserhebung und auch spätere Entscheidungen einzubinden. Vielmehr gibt es die vereinfachte Annahme, ein einziger fachlicher Vertreter kann alle Belange repräsentieren.
  - » Die Annahme, dass der fachliche Vertreter immer persönlich im Projekt verfügbar ist, ist in der Praxis nur selten realisierbar. Insbesondere bei verteilter Entwicklung, was immer mehr die Regel ist.
  - » Die Annahme, Use-Storys reichen als einziger Anforderungstyp widerspricht allen heutigen Erkenntnissen des Anforderungsmanagements.
  - » Es gibt keine Vision und keinen Business-Case, der eine geschäftliche Würdigung des Vorhabens enthält und die Ziele und den Nutzen des Vorhabens definiert.
- *Architektur:* Der architekturelle Ansatz von XP beschränkt sich auf „System-Metaphor“ und nachträgliches Refactoring. Das kann nicht reichen, um eine an den nichtfunktionalen Anforderungen orientierte, bewusst geplante Software-Architektur zu erzielen. Eine Aufteilung der Entwicklungsaufgaben auf Teams ist somit auch nur anhand der User-Storys möglich. Schnittstellen zwischen den entwickelten Modulen werden bestenfalls in ungeplanten mündlichen Gesprächen abgestimmt und kaum dokumentiert. In der Praxis wird beim Refactoring versucht, bestehenden Code nachträglich zu einer Architektur umzustrukturieren. Das erfolgt aber ohne Blick auf eine architekturelle Gesamtvision und bedeutet erheblichen nachträglichen Mehraufwand.
- *Qualitätssicherung und Test:* Der Fokus liegt stark auf Unit- und Entwickler-tests. Die Gefahr ist groß, dass die Integrations- und Performancetests zu kurz kommen. Auch die Testplanung beschränkt sich auf das Schreiben von Tests zu User-Storys.
- *Kommunikation:* Da die Kommunikation schon per Forderung des agilen Manifests [Agil] möglichst mündlich zwischen Personen erfolgen soll, ergibt sich die Schwierigkeit, die Information in großen oder gar verteilten Teams allen zugänglich zu machen. Auch für nachfolgende Releases und die Wartung wird die Dokumentation nicht aus der Methode heraus hinterlassen (vgl. [Her10]).

Scrum

Die gleichen Kritikpunkte wie für XP gelten auch für Scrum (vgl. [Scr]). Der größte Unterschied zwischen XP und Scrum besteht darin, dass Scrum mehr Hinweise zur iterativen Planung und Durchführung gibt, während XP mehr konkretes zur eigentlichen Implementierung enthält.

Im Wesentlichen wird es umso schwieriger, agile Methoden ohne Ergänzung einzusetzen, je größer und komplexer das Projekt ist und je mehr das Ergebnis von Anfang an – per Vertrag – festgelegt werden muss. Insbesondere Regeln, die durch Gesetzgeber, Normen und Unternehmen vorgegeben sind, müssen berücksichtigt werden.

Antworten des RUP auf die Schwächen agiler Methoden

Betrachtet man die Schwächen bzw. Punkte, zu denen die agilen Methoden keine Aussagen treffen, so findet man umfangreiche Antworten auf Projektebene im RUP (vgl. [Ess07]):

- *Projektmanagement:* Das Projektmanagement ist eine eigene, umfangreich beschriebene Disziplin des RUP. Ebenso das Change-Management, das in vielen Methoden zum Projektmanagement gehört. Lediglich zu den Finanzen gibt es auch im RUP wenig Aussagen.
- *Anforderungen:* Auch das Anforderungsmanagement ist eine Disziplin des RUP, die umfangreiche Beschreibung zu verschiedenen Anforderungsrollen, -Stufen und –Typen enthält. Mit „Us -Case Driven“ sind die Anforderungen sogar in den wichtigsten Kennzeichen des RUP verankert. Die Dokumentation der Vision gibt einen Überblick über die grundlegenden Ziele und den Nutzen. Darin enthalten ist auch ein Busines -Case.
- *Architektur:* Ein weiteres wesentliches Kennzeichen des RUP besteht darin, dass er „architecture-centric“iIst. Das heißt, ein wichtiger Teil ist das Überführen der nichtfunktionalen Anforderungen in den Iterationen der Ausarbeitungsphase in eine stabile tragfähige Softwarearchitektur. Dazu ist auch die Rolle des Architekten definiert.
- *Qualitätssicherung und Test:* In der eigenen Disziplin „Test“ werden jenseits der Entwicklertätigkeiten alle Belange modernen Testens beschrieben. Von der ersten Planung bis zum Auswerten der Testergebnisse der verschiedensten Testarten.
- *Kommunikation:* Die Kommunikation im RUP erfolgt im Wesentlichen über Informationen (häufig UML-Modelle), die in Werkzeugen hinterlegt sind. Das hat einerseits den Vorteil, dass der Entwickler keine „extra Dokumentation“ jenseits dessen, was er für die Entwicklung ohnehin erarbeitet, schreiben muss. Andererseits können alle Teammitglieder jederzeit über Server, die mit dem Internet verbunden sind, auf die aktuellsten Informationen zugreifen, und dies weltweit, auch in verteilten Teams. Die Information steht damit auch für nachfolgende Entwicklungszyklen und die Wartung zur Verfügung.

Wie kann der „richtige“ Prozess aussehen?

Ein konkreter Prozess sollte ohnehin immer exakt auf die Gegebenheiten eines Projekts abgestimmt sein. Handelt es sich um ein kleines Vorhaben und Team oder um ein komplexes Projekt mit verteilten Teams? Ist das Risiko bei Fehlern gering und gibt es dazu keine Vorgaben, oder geht es um Menschenleben und müssen entsprechende gesetzliche Bestimmungen erfüllt werden. Kann die Arbeit auf Zuruf für interne Nutzer organisiert werden, oder handelt es sich um eine Auftragsarbeit mit umfangreichen Verträgen?

Nur wenn der konkrete Prozess auf all diese unterschiedlichen Randbedingungen eines Projektes eingeht, kann er überhaupt richtig befolgt werden und zu effektiver Arbeit führen. In jedem Fall empfiehlt es sich das Beste aus den beiden Welten Agile und RUP zu vereinen.

Bezüglich vieler wertvoller Eigenschaften, wie vor allem der iterativen Entwicklung gibt es ohnehin keine Widersprüche. Das heißt, die iterative Entwicklung sollte im Allgemeinen gesetzt sein. Die vielen Ansätze und Hinweise zum menschlichen Faktor wie Respekt, Vertrauen und nachhaltiges Tempo sind sinnige Beiträge der agilen Methoden. Ebenso wie der Ansatz, von einem minimalen Prozess auszugehen (das heißt also erst mal von der agilen Methode) und diesen lieber nach Notwendigkeit zu ergänzen. Die im vorherigen Absatz genannten Beiträge des RUP zu Projektmanagement, Anforderungsmanagement, Architektur, Test und Kommunikation können aber genau zu dieser Ergänzung dienen.

Das Ergebnis darf dann ganz nach Geschmack als erweiterte agile Methode oder als gültige, angepasste RUP-Prozessinstanz bezeichnet werden.

Literatur & Links

[BQI10] BQI Research, Agile Softwareentwicklung, Ein Überblick der wichtigsten Methoden, Status Report 2010

[Ecl] Eclipse, EPF Projekt, siehe: www.eclipse.org/epf/

[Jac99] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999

[Agil] Manifesto for Agile Software Development, siehe: agilemanifesto.org/

[UML] Object Management Group, Unified Modeling Language, siehe: www.uml.org/

[Wiki] Wikipedia, Extreme Programming, siehe: de.wikipedia.org/wiki/Extreme_Programming

[XP] XP Extreme Programming, siehe: www.extremeprogramming.org/

[Scr] Scrum Alliance, What is Scrum, siehe: www.scrumalliance.org/pages/what_is_scrum

[Her10] Dr. J. Herzog, G. Holzmüller, W. Schöpe, Sonderbeilage Agility, ObjektSpektrum, 2010

[Ess07] A. Essigkrug, T. Mey, Rational Unified Process kompakt, Elsevier, 2007

Andreas Essigkrug

Teamleader IBM Rational Service für Prozess-, Produkt-, Projekt-, Portfoliomanagement und Enterprise Modernization. Koautor von „Rational Unified Process kompakt“. Aufgabenschwerpunkte in den Bereichen Softwareentwicklungsprozesse und Werkzeugunterstützung für den gesamten Softwareentwicklungszyklus. (andreas.essigkrug@de.ibm.com)

Wer suchet, der findet

Azlan sucht eigenständiges und kreatives Softwarehaus!

	Bitte hier ankreuzen JA	NEIN
Sie sind:		
• ein spezialisiertes **Softwarehaus** mit einer innovativen, plattformunabhängigen Lösung oder Applikation	●	●
• ein Anbieter von definierten **Service- oder Maintenance Dienstleistungen**	●	●
• **Profi in Ihrer Branche** oder in einem Marktsegment über deutsche Grenzen hinaus	●	●
• **Hostinganbieter** oder bereit Ihre Software zu vermieten	●	●
Sie suchen:		
• neue und profitable Wege, wobei der Fokus auf Ihrem Kerngeschäft liegt	●	●
• Ideen, um Ihre Lösung mit IBM Standardsoftwarekomponenten zu veredeln	●	●
• Komplettlösungen für Ihre Endkunden	●	●
• Zugänge zu neuen Marktsegmenten und langfristige Kundenbeziehungen	●	●
Wir bieten Ihnen:		
• das passende Businessmodell von „klassisch“ bis „modern“	●	●
• Bundlemöglichkeiten mit IBM Software- und Hardware Lösungen zu fairen Preisen	●	●
• einfache Bestell- und Abwicklungsprozesse	●	●
• individuelle Unterstützung im Vertrieb und Marketing	●	●

Azlan bietet mit dem IBM Business Modell (ASL) (auch bekannt als OEM) speziell Lösungs- sowie Applikationsherstellern neue Möglichkeiten und Chancen beim Vertrieb Ihrer Software. Ihr Kunde kann eine Gesamtlösung (Bundle) inklusive Ihrer Applikation, Dienstleistung und Serviceleistung beziehen - Sie bleiben nahe am Kunden und auf Wunsch sogar weltweit! Sie konzentrieren Ihre Energie auf Ihr Kerngeschäft, Ihre Stärken und gemäß dem Motto „Schuster bleib bei Deinen Leisten“ kümmern wir uns um den Rest.

Haben Sie sich wieder erkannt und möchten Sie mehr dazu erfahren?

Das IBM Azlan Team freut sich auf Sie!

Tech Data GmbH & Co. OHG
Geschäftsbereich Azlan
Birgit Brunner • Business Development Manager
Telefon: 089 4700-3039 • E-Mail: bbrunner@techdata.de

IBM Premier Business Partner

Azlan
A Trademark of Tech Data

Agile – Was nun?

Dass agile Entwicklungsmethoden den Erfolg eines Projektes fördern, steht für viele Softwareentwicklungsorganisationen außer Frage. Faktoren wie beispielsweise etablierte – teils agil-konträre – Entwicklungsmethoden und betriebliche Organisationsstrukturen stellen jedoch eine nicht zu vernachlässigende Hürde bei der gewünschten Adaptierung der agilen Methoden dar. Dieser Artikel geht auf die wesentlichen Aspekte der betriebsinternen Organisationsstrukturen und Prozesse ein, die es bei der Einführung agiler Methoden zu beachten gilt.
Das agile Manifest lautet: Individuen und Wechselwirkungen haben Vorrang vor Prozessen und Tools, die Arbeitssoftware hat Vorrang vor umfassender Dokumentation, Kundenzusammenarbeit hat Vorrang vor Vertragsaushandlung und die Reaktion auf Änderungen hat Vorrang vor dem Einhalten eines Plans. Die Vorzüge agiler Methoden liegen folglich in Zusammenarbeit, Flexibilität und Konzentration auf den Mehrwert der entwickelten Software. Prozesse, umfassende Dokumentation, Vertragsaushandlung und Einhaltung eines Plans sind im agilen Kontext weniger von Bedeutung. Was aber bedeutet dies nun im Zusammenhang mit vorhandenen Entwicklungsprozessen bzw. Organisationsstrukturen?

Anforderungen: Vorabplanung vs. iterativ inkrementell

Bei der Mehrheit der aktuell durchgeführten Softwareprojekte wird von Beginn an festgelegt, welche Anforderungen in welchem Zeitraum mit welchem Umfang an Ressourcen realisiert werden sollen. Die Anforderungen selbst werden meist sehr feingranular verfasst und während des Projektverlaufs gar nicht oder via eines etablierten Change-Management-Prozesses geändert. Hintergrund dieser Vorgehensweise ist die Einstellung der Kunden: Diese bevorzugen - verständlicherweise - eine vorab definierte und vertraglich festgehaltene Leistungsbeschreibung gegenüber einer auf Zeit und Material basierender evolutionären Entwicklung. Bei letzterem läuft der Kunde Gefahr, keine gute Kostenkontrolle zu haben.

Dies widerspricht dem agilen Ansatz. Bei einer agilen Entwicklungsmethode werden zu Beginn High-Level-Anforderungen erfasst und priorisiert. Die Anforderungen mit der höchsten Priorität werden anschließend bis zu einem solchen Detaillierungsgrad verfeinert, sodass der Aufwand für die zugehörigen Tasks und Aktivitäten gut geschätzt werden kann (entweder in Personentagen oder Story Points).

In einer ersten Iteration werden nun durch Abarbeitung dieser feingranularen Tasks die zugehörigen Anforderungen realisiert, getestet und dokumentiert. Am Ende der Iteration kann der Kunde durch Sichtung der erstellten Artefakte eine Vor-Abnahme durchführen oder aber Änderungswünsche anbringen. Mit den in dieser Iteration gewonnenen Erkenntnissen verifiziert das Team die verbleibenden Anforderungen, überarbeitet sie bei Bedarf und priorisiert sie neu. Es können neue Anforderungen hinzukommen oder alte gestrichen werden.

Jede nachfolgende Iteration erfolgt gemäß dem Vorgehen: Detaillierung der höchst priorisierten Anforderungen, Realisierung, Sichtung, Re-Priorisierung der verbleibenden Anforderungen. Dies geschieht bis alle Anforderungen realisiert wurden. Eine solche agile Vorgehensweise erhöht die Integration mit dem Kunden und erlaubt eine kontinuierliche Ausrichtung der Entwicklung basierend auf den bisher gewonnenen Erkenntnissen sowie etwaigen Änderungen im Anforderungsportfolio. Es ist Aufgabe des Produkt- und Projektmanagements bzw. des verantwortlichen Bereichsleiters, eine entsprechende Umgestaltung des Entwicklungsprozesses vorzunehmen.

Firmenkultur – Etablierte Entwicklungsprozesse

Die Firmenkultur beziehungsweise dort geltende Prozesse und Berichtsstrukturen beeinflussen die Etablierung von Softwareentwicklungsprozessen. Viele der börsennotierten Unternehmen beispielsweise haben eine quartalsgetriebene Kultur. Mittel- und langfristige Zielvereinbarungen bestimmen den Ablauf und das Reporting.

Existierende Entwicklungsprozesse – insbesondere dem Wasserfallmodell ähnliche Prozesse – können eine nicht zu unterschätzende Hürde bei der Einführung agiler Methoden darstellen, da diese oft konträre Ansätze verfolgen. Wie bereits im oberen Abschnitt „Anforderungen: Vorabplanung vs. iterativ Inkrementell“ beschrieben, müssen die Entwicklungsprozesse im Hinblick auf das Anforderungsmanagement und die Projektplanung geändert werden. Herkömmliche Entwicklungsprozesse weisen oft wasserfallartige Planungsmodelle und -zyklen auf. Meilensteine wie „Anforderungen definiert“, „Design abgeschlossen“ etc. müssen bei der Einführung agiler Entwicklungsmethoden durch iterative, inkrementelle Checkpoints ersetzt werden.

Darüber hinaus ändert sich das Verantwortungsfeld einiger, an einer Entwicklung beteiligter Rollen. Das Produktmanagement beispielsweise muss die Zusammenarbeit mit Kunden intensivieren und in Teilbereichen sogar ändern. Der Kunde muss idealerweise integraler Bestandteil des Entwicklungszyklus werden und die jeweils am Ende einer Iteration erstellten Software-Artefakte sichten und bewerten (➟ Stakeholder Demo). Zu diesem Zweck ist es Aufgabe des Produktmanagements, den Kunden davon zu überzeugen, den notwendigen Aufwand seinerseits zu tätigen. Es muss dem Kunden deutlich erkennbar sein, dass diese zu erbringenden Aufwände einen späteren Mehrwert der für ihn erstellten Software darstellen.

Projektleiter und leitende IT-Architekten müssen damit umgehen können, dass sich die Menge der Anforderungen während des Entwicklungszyklus dynamisch ändert. Sie müssen darauf achten, dass der Detaillierungsgrad der in der nächsten Iteration zu realisierenden Anforderungen granular genug ist, um die dahinter stehenden Aufwände zu schätzen. Darüber hinaus müssen sie die Kommunikation innerhalb des Teams mehr denn je forcieren, da ein Prozess agiler Entwicklung enge Teamintegration und häufige Kommunikation erfordert. Im Extremfall geht dies hin bis zu täglichen kurzen Teambesprechungen – den sogenannten Scrum-Meetings.

Organisationsstrukturen

Ein agiler Entwicklungsprozess erfordert eine enge Teamintegration und eine offene, häufige Kommunikation, gemeinsame Ziele sowie ein klares, gemeinsames Verständnis des Mehrwerts einer jeden Anforderung aus Sicht des Kunden. Je mehr Barrieren zwischen Projektleitern, Entwicklern, Kunden, Managern und anderen Beteiligten bestehen, desto schwerer ist es, diese notwendige Basis für agile Softwareentwicklung zu schaffen. Eine dieser Barrieren kann in der Organisationsstruktur des Entwicklungsbereiches liegen. Das Management ist hier gefordert, maximale Transparenz und Integration zwischen den Teams zu ermöglichen. Besteht ein Team aus Mitgliedern unterschiedlicher Organisationen, so muss der Leiter des Teams für die Dauer des Projektes bzw. der Iteration klar und deutlich die

Weisungsbefugnis für alle Teammitglieder erhalten. Diese übertragene Autorität gestattet es dem Teamleiter, interne Konflikte ohne unnötige Eskalation über externes Management und damit wesentlich effizienter zu lösen. Das Team muss als Einheit ohne unnötige, externe Abhängigkeiten agieren können.

Eine weitere wichtige Rolle bei der Zusammenstellung von agil arbeitenden Teams bildet der ausgewogene Mix an notwendigen Skills innerhalb des Teams. Agil arbeitende Teams sollen in der Lage sein, spezifische Anforderungen autark im Team zu realisieren. Dies umfasst Aktivitäten wie Architektur, Design, Implementierung, Test und Dokumentation. Damit das Team dazu in der Lage ist, sollte es idealerweise aus Mitgliedern mit jeweils unterschiedlichen Skill-Schwerpunkten zusammengesetzt sein. Dies ist erneut konträr zu den meisten bisher existierenden Team-Ansätzen, da Teams bisher eher auf Basis von Skill-Schwerpunkten definiert wurden. Zur Realisierung eines agilen Ansatzes ist demzufolge eine Team-Restrukturierung erforderlich. Auch hier ist das jeweilige Management gefordert, die neuen Teambildungsansätze durchzusetzen oder zumindest zu ermöglichen.

Ein weiterer wichtiger Punkt bei der Team-Zusammenstellung kann die Zuweisung eines sogenannten Agile Moderators sein. Gerade bei Entwicklungsorganisationen, die bisher keine Erfahrung im Bereich agiler Entwicklungsprozesse gesammelt haben, macht eine solche unterstützende Funktion als Bestandteil eines jeden Teams Sinn. Ein solcher Agile Moderator sollte um ausreichend praktische Erfahrung in dem Bereich agiler Entwicklungsprozesse verfügen. Seine Hauptaufgabe ist es, das Team während der verschiedenen Iterationen zu coachen und die Einhaltung des Agile Manifesto sicher zu stellen. Der Projektleiter sowie der leitende IT-Architekt sind hier gefordert, das Management davon zu überzeugen, den notwendigen Mehraufwand für den Einsatz eines solchen Moderators je Team zu gewähren. ■

Literatur und Links

[Cohn08] Mike Cohn, „Agile Estimating and Planning“, Prentice Hall, 2008

[Pich08] Roman Pichler, „Scrum – Agiles Projektmanagement erfolgreich einsetzen“, dpunkt.verlag, 2008

[Chin04] Gary Chin, „Agile Project Management – How to succeed in the face of Changing Project Requirements”, Mcgraw-Hill Professional, 2004

[Schwa04] Ken Schwaber, „Agile Project Management with Scrum”, Microsoft Press, 2004

Oliver Röhrsheim

Zertifizierter Senior IT Architect IBM Softwaregroup, PMP®. Aufgabenschwerpunkte in den Bereichen IT-Architektur, Software-Entwicklungsprozesse, Agiles Project Management.
(o.roehrsheim@de.ibm.com)

Was ist Scrum?

Scrum ist das wohl verbreitetste Vorgehen bei der agilen Software-Entwicklung. Das liegt an der klaren und überschaubaren Prozessbeschreibung: Scrum kennt lediglich drei Rollen und drei Artefakte.

Die Rollen umfassen:

- Mit dem Product-Owner, der dem Projekt ständig zur Verfügung steht, wird die agile Forderung nach Zusammenarbeit mit dem Kunden berücksichtigt. Er verwaltet den Product-Backlog, der alle Anforderungen (meist in Form von User-Stories) enthält und priorisiert diese ständig hinsichtlich ihres Geschäftswerts.
- Das Team ist in Scrum recht klein und besteht aus ca. 7 (± 2) Personen. Für die Teammitglieder werden bewusst keine weiteren Rollen definiert – sie sollen einen möglichst universellen Skill haben oder mit der Zeit entwickeln.
- Der Scrum-Master ist Teammitglied und Coach zugleich. Er beherrscht den Prozess und fungiert als Mentor für das Team bezüglich des Vorgehens. Er hat jedoch keine Enscheidungs- und Weisungsbefugnisse sondern das Team soll hier selbstständig agieren.

Die Scrum-spezifischen Artefakte sind:

- Im Produkt-Backlog werden die Requirements gesammelt und bewusst nur sehr knapp beschrieben. Sie werden erst verfeinert wenn sie sich der Realisierung nähern.
- Der Sprint-Backlog enthält alle Anforderungen die im Sprint (Iteration) zu realisieren sind. Diese Items werden aus dem Produkt-Backlog ausgewählt, weswegen er gelegentlich auch Selected-Backlog genannt wird.
- Die Sprint-Tasks sind die konkreten Aufgaben des Teams, die nötig sind, um die Anforderungen in Software umzusetzen.

Der iterative Prozess ist „time-boxed“, das heißt jeder Sprint hat konstante Dauer (meist 4 Wochen lang), und läuft folgendermaßen ab:

Der Produkt Owner sammelt und bewertet ständig die Anforderungen im Product Backlog.

Hieraus zieht er eine Kanditaten-Liste von Anforderungen, mit der er in das erste Planungsmeeting (Sprint Planning 1) geht und diese dem Team erläutert. Hier einigt man sich auf ein gemeinsames Ziel für den Sprint und wählt die zugehörigen Anforderungen aus. In einem zweiten, folgenden Meeting bespricht das Team die grobe Architektur und erarbeitet die konkreten Aufgaben, die die Anforderungen realisieren (Sprint Tasks). Die Teammitglieder wählen während des Sprints dann sukzessive Aufgaben zur Erledigung aus (Pull-Prinzip).

Ein innerer, täglicher Zyklus wird durch die allmorgendlichen „Daily Scrum Meetings“ gestartet: Das ganze Team trifft sich für max. 15 Minuten, um sich gegenseitig über den aktuellen Stand und die heutigen Aufgaben zu informieren. Andere Themen sollen in diesem Meeting nicht besprochen werden.

Die Ergebnisse jedes Sprints werden dem Product Owner und allen weiteren Stakeholdern in einem Sprint-Review Meeting präsentiert und von diesen abgenommen. Zur kontinuierlichen Verbesserung des Entwicklungsprozesses sieht Scrum nach jedem Sprint die Sprint Retrospective vor, in der sich das Team intern trifft und bespricht, was gut gelaufen ist und was es im nächsten Sprint besser machen möchte.

Anwendungsportfolios – strategisch und agil

Unternehmen sind einem ständigen Veränderungsprozess unterworfen. Die Frage, wie gut dieser Veränderungsprozess in seiner Gesamtheit gemeistert werden kann, entscheidet oft über Erfolg oder Misserfolg am Markt. Das Management von fachlichen Anforderungen und der Anwendungslandschaft spielt dabei eine große Rolle. Aus diesem Grund beleuchten wir hier einen agilen Ansatz zum Management von Anwendungsportfolios und Unternehmensarchitekturen.

Unternehmen stehen nicht still, sie sind dynamisch und entwickeln sich ständig weiter. Nur so können sie den sich ständig veränderten Anforderungen gerecht werden. Dazu zählen Veränderungen in der Gesellschaft, Marktchancen, aggressive Wettbewerber und alle rechtlichen Anforderungen, die Einfluss auf die Richtung eines Unternehmens haben. Die Geschwindigkeit, mit der eine Organisation in der Lage ist, auf diese Veränderungen zu reagieren, kennzeichnet die Agilität eines Unternehmens.

Wie kann diese Beweglichkeit verbessert werden? Eine Kombination aus Business (Geschäftsziele, Konzepte, Strategien), Technologie (Computersysteme, Infrastruktur) und Organisation (Menschen, Verfahren, Effizienz der Durchführung) ist notwendig, um Business Services zur Verfügung zu stellen. Die Ausrichtung dieser unterschiedlichen Elemente und ihre Fähigkeit zur Zusammenarbeit als Ganzes bestimmen die Effizienz, mit der die Organisation Dienstleistungen anbietet. Die IT-Systeme, die die Bereitstellung dieser Dienste automatisieren oder zumindest unterstützen, müssen daher mit den unterschiedlichen Elementen Business, Technologie und Organisation in Einklang gebracht werden. Anwendungsportfolio-Management hilft beim Messen und Ausrichten dieser Elemente, um die Leistungserbringung zu steigern.

Jede Anwendung wird dabei auf Basis vereinbarter Metriken bewertet, die die Bereiche Business, Technologie und Organisation umfassen. Damit wird ein ganzheitliches Bild des allgemeinen Zustands einer jeden Anwendung entworfen. Stärken, Schwächen, Trends und Muster zwischen unterschiedlichen Anwendungen innerhalb des Portfolios werden transparent gemacht.

Anwendungsportfolio-Management (APM) ist dabei keine einmalige Angelegenheit. Nachhaltigkeit, Kontinuität und die aktive Steuerung des Anwendungsportfolios sind wichtige Eigenschaften, die es zu berücksichtigen gilt. Dadurch wird sichergestellt, dass das Anwendungsportfolio die richtige Menge an Agilität und Qualität in Bezug auf Kosten und Anforderungen bereitstellt.

Aufgrund der ständigen Wiederholung und Optimierungen der Anwendungslandschaft ist APM ein idealer Kandidat, um durch einen kontinuierlichen Verbesserungsprozess angetrieben zu werden. Dabei muss ein formaler Prozess einschließlich einer Feedback-Schleife definiert und etabliert werden. Dieser kontinuierliche Verbesserungsprozess muss sicherstellen, dass die relevanten Akteure während des gesamten Prozesses beteiligt sind, dass die richtigen Metriken gesammelt und analysiert werden, und dass die Geschäftsleitung mit den richtigen Informationen versorgt wird. Ziel ist es dabei, strategische Entscheidungen treffen zu können, damit die Leistungsfähigkeit und Ausrichtung des Anwendungsportfolios noch besser an der Geschäftsstrategie und den Unternehmenszielen ausgerichtet werden kann.

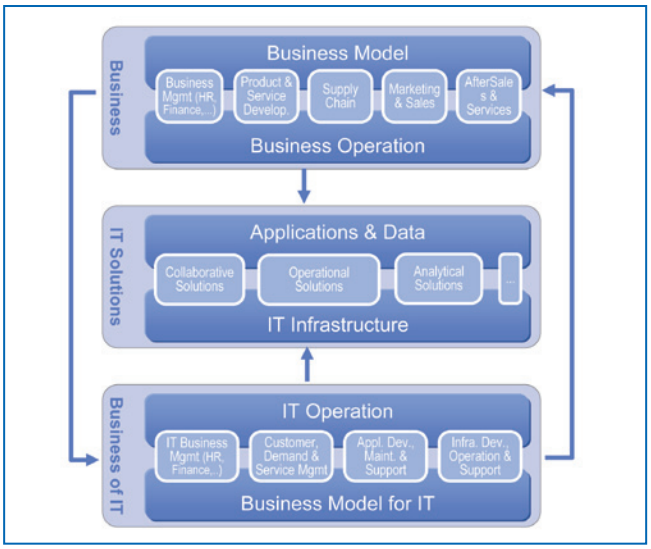


Abb. 1 Ganzheitliche Architekturbetrachtung eines Unternehmens

Aus den gewonnen Erkenntnissen und Empfehlungen müssen Aktionspläne entstehen. Dabei ist vor allem wichtig, eine ganzheitliche Sicht zu behalten. Anwendungsportfolio-Management dreht sich nicht nur um Anwendungen! Vielmehr geht es darum, wie gut diese Anwendungen an der Gesamtorganisation ausgerichtet sind. Beispielsweise kann die Wartung bestimmter Anwendungen ein bestimmtes Technologie-Know-how voraussetzen, was Unternehmen durch personelle Maßnahmen, wie zum Beispiel Einstellungs- und Ausbildungsprogramme, adressieren müssen.

Anwendungsportfolio-Management folgt im Wesentlichen einem Bottom-up-Ansatz. Es beginnt mit der Aufnahme aller Anwendungen und der Messung definierter Kriterien. Dabei lässt sich das Anwendungsportfolio-Management sehr gut mit dem methodischen Ansatz einer Enterprise-Architektur integrieren.

Enterprise-Architecture

Die Enterprise-Architektur hat zum Ziel, die strategischen Unternehmensziele, Visionen und die dafür notwendigen geschäftlichen Ziele, fachlich und IT-seitig zu unterstützen. Viele mehr oder weniger unabsehbare Aspekte wirken sich auf diese Herausforderung aus, wie etwa Marktveränderungen, Unternehmenszusammenschlüsse, Übernahmen, Technologieänderungen und anderes.

Unternehmen haben erkannt, dass eine funktionierende Enterprise-Architektur der Schlüssel für eine erfolgreiche (schnellere, bessere und günstigere) Unterstützung der Geschäftsbereiche zur Erreichung ihrer Ziele ist.

Oftmals erarbeitet ein Team bzw. eine Gruppe von Architekten die Enterprise-Architektur. Dabei entstehen Modelle (Unternehmensdaten-, Architektur-, Geschäftskomponentenmodelle etc.), Dokumente, Anweisungen, Definition von Standards und anderes. In einem Werkzeug wie zum Beispiel dem IBM Rational System Architect lassen sich diese möglichst zentral organisieren und anschließend im Intranet publizieren oder per E-Mail den Abteilungen (manchmal auch nur der IT-Abteilung) zur Verfügung stellen. Programme und Projekte werden meist IT-seitig aufgesetzt, um diese „neuen" von den Enterprise-Architekten erarbeiteten Richtlinien und Empfehlungen Rechnung zu tragen bzw. sie umzusetzen. Man definiert neue Rollen und Verantwortlichkeiten und etabliert sie im Unternehmen.

Kommunikation und Integration als Schlüssel für den Erfolg

Agile Software-Entwicklung ist aus der heutigen Anwendungsentwicklungslandschaft nicht mehr wegzudenken. Anstatt ein Projekt im Wasserfallmodell durchzuführen, wird in der Agilen Softwareentwicklung der Schwerpunkt auf möglichst rasche Ergebnisse gelegt. Dabei kommt es nicht darauf an, dass das Ergebnis gleich von Anfang an perfekt ist („niemand ist perfekt"), sondern im Projektverlauf wird das Ergebnis immer klarer und besser.

Bereits im Jahr 2001 haben einige Persönlichkeiten aus der Softwareentwicklung bei einem Treffen in Utah das „Agile Manifest" ins Leben gerufen. Es lautet:

> *„Wir suchen nach besseren Wegen, Produkte zu entwickeln, indem wir es selbst praktizieren und anderen dabei helfen, dies zu tun.*
>
> *Individuen und Interaktionen haben Vorrang vor Prozessen und Werkzeugen.*
>
> *Funktionsfähige Produkte haben Vorrang vor ausgedehnter Dokumentation.*
>
> *Zusammenarbeit mit dem Kunden hat Vorrang vor Vertragsverhandlungen.*
>
> *Das Eingehen auf Änderungen hat Vorrang vor strikter Planverfolgung.*
>
> *Wir erkennen dabei sehr wohl den Wert der Dinge auf der rechten Seite an, wertschätzen jedoch die auf der linken Seite noch mehr.*[1]*"*

Eine zentrale Bedeutung kommt bei der Agilen Softwareentwicklung der Kommunikation zu. Ein Team oder mehrere kommunizieren sehr viel miteinander, um möglichst früh Probleme diskutieren und beheben zu können. So soll verhindert werden, dass zu starre Prozesse und Standardisierungen die Entwicklung von Software lähmen.

Eine wichtige Sache wird dabei immer wieder übersehen bzw. nicht beachtet. Es sind Menschen, die auf fachlicher und IT-Seite miteinander zusammenarbeiten, um gemeinsam erfolgreich zu sein. Menschen müssen kommunizieren, um die für die Zielerreichung notwendige Information zu erhalten. Dies bezieht sich sowohl auf die fachlichen Anforderungen („user story ") als auch auf die technischen Herausforderungen.

Um nun für den fachlichen Auftraggeber (und damit für das Unternehmen) die besten Ergebnisse zu liefern, ist es notwendig, dass die erarbeiteten Richtlinien, Vorgaben etc. möglichst realistisch und bedarfsgerecht ausgelegt sind. Meistens sind Dokumente bereits aus Anforderungssicht veraltet, sobald sie erstellt sind, da zum Beispiel der Freigabeprozess zu lange dauert.

Neue Anforderungen müssen sich demzufolge möglichst unkompliziert und schnell in der Enterprise-Architektur abbilden lassen. Dies ist nur zu bewerkstelligen, wenn ein stetiger Informationsaustausch und eine entsprechende Flexibilität in der Enterprise-Architektur vorhanden sind. Ein zentrales Werkzeug vereinfacht dies.

Die „IBM Enterprise Architecture Method" verfolgt genau diesen Ansatz. Durch ihren iterativen und inkrementellen Aufbau unterstützt sie die Anforderungen und ermöglicht damit eine agile Enterprise-Architektur.

Eine zentrale Rolle bei der Methodik kommt den Anforderungen (sowohl fachlich als auch technisch) zu. Iterationen werden so oft wie nötig durchgeführt, um möglichst schnell Probleme zu erkennen und neue Anforderungen berücksichtigen zu können. Die Einführung einer Enterprise-Architektur sollte dabei in kleinen Schritten erfolgen und sich in vielen Iterationen immer weiter „ausdehnen".

Dabei hat sich bewährt, dass die Enterprise-Architekten nicht in einem „Elfenbeinturm" sitzen und Dokumente für die Regale erstellen, sondern aktiv in den Projekten vertreten sind, rege kommunizieren und die Informationen zentral und werkzeuggestützt organisieren. Diese Einbettung der Enterprise-Architekten in den Projekten (auch hier gilt: fachliche und technische Projekte) verspricht den größtmöglichen Erfolg für das Unternehmen.

Die Erfahrung zeigt, dass Enterprise-Architekten oftmals als „Polizei" betrachteten werden und einen eher „projektblockierenden" Eindruck in den Projekten hinterlassen. Durch eine aktive und integrierte Mitwirkung in den Projekten kann jedoch der Enterprise-Architekt frühzeitig an den Entscheidungen teilhaben, ohne dass für das Projekt endlose „Review"-Schleifen gedreht werden müssen.

Von zentraler Bedeutung hierbei ist es, die Kluft zwischen Fachabteilung und IT zu schließen. Schließlich soll die IT-Abteilung die Fachabteilung mit IT-Mitteln unterstützen und nicht nur als Kostenfaktor betrachtet werden. Hier steht der Mehrwert an erster Stelle, den die IT zum Unternehmenserfolg beiträgt. Ebenso wenig soll die IT die fachlichen Vorgaben des Unternehmens vorschreiben bzw. begrenzen, indem zum Beispiel durch eine unflexible Infrastruktur und Architektur fachliche Anforderungen gar nicht oder nicht zeitnah erfüllt werden.

Auch hier ist der Enterprise-Architekt gefragt und darf sich nicht hinter Dokumenten und Richtlinien „verstecken". Er muss durch seine Integration in die Fachabteilung, in das Projekt ein aktives Mitglied des gesamten Teams werden.

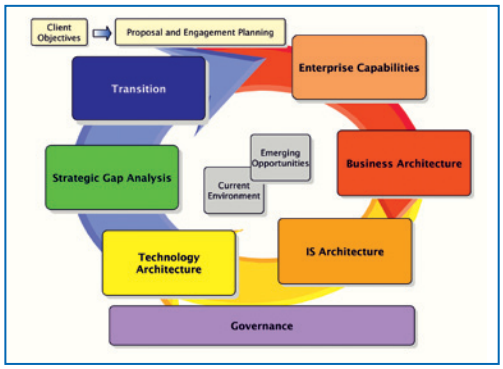


Abb. 2 Die IBM Enterprise Architecture Methode im Überblick

Wir haben festgestellt, dass die agile Vorgehensweise bei der Einführung und Durchführung einer Enterprise-Architektur am effizientesten ist und somit den größten Wertbeitrag für den Erfolg des Unternehmens leistet.

Weiterführende Links:

[1] Orig. in Englisch unter http://www.agilemanifesto.org, Übersetzung aus http://scrum-master.de

IBM Application Services: http://www.ibm.com/services/de/bcs/html/applicationservices.html

Markus Weyerhäuser

Berater im Bereich IBM Strategy & Transformation Management Consulting und Leiter des IBM GBS Kompetenzzentrums für Anwendungsmodernisierung. (markus.weyerhaeuser@de.ibm.com)

Günter Triep

Enterprise-Architekt im Bereich IBM Strategy & Transformation Management Consulting. (guenter.triep@de.ibm.com)

Welcome to Reality! Agile vs. Klassisch

Agile und klassische Vorgehensweisen müssen sich nicht widersprechen, im Gegenteil: Agile Ansätze können klassische Ansätze sinnvoll ergänzen. Um die Vorteile beider Welten nutzen zu können, kann eine gemischte Projektstruktur angewendet werden, die je Teilprojekt unterschiedliche Vorgehensweisen zulässt. Wie das in der Realität aussehen kann und welche Rolle die Projektleiter, die Teams und die Tools dabei spielen, davon handelt der folgende Artikel.

Die Vorteile agiler Ansätze

Agile Vorgehensweisen unterscheiden sich von klassischen Projektmanagement-Methoden zunächst in der Betonung der vier, im agilen Manifest beschriebenen, „agilen Werte“:

- Individuen und deren Interaktionen sind wichtiger als Prozesse und Tools
- Funktionierende Software ist wichtiger als umfangreiche Dokumentation
- Enge Zusammenarbeit mit dem Kunden steht über Verträgen
- Offenheit für Änderungen ist wichtiger als feste Pläne zu verfolgen

Aus diesen Werten leiten sich agile Prinzipien, Methoden und Vorgehensweisen ab. Beispiele für agile Vorgehensweisen sind Extreme Programming (XP), Scrum, Crystal und andere.

Wie das folgende Bild zeigt, bedeuten agile Vorgehensweisen nicht einen gänzlich unterschiedlichen Ansatz zum klassischen Vorgehen, sondern eher eine andere Strategie im Umgang mit den „Triple Constraints“: Anforderungen, Zeit und Ressourcen.

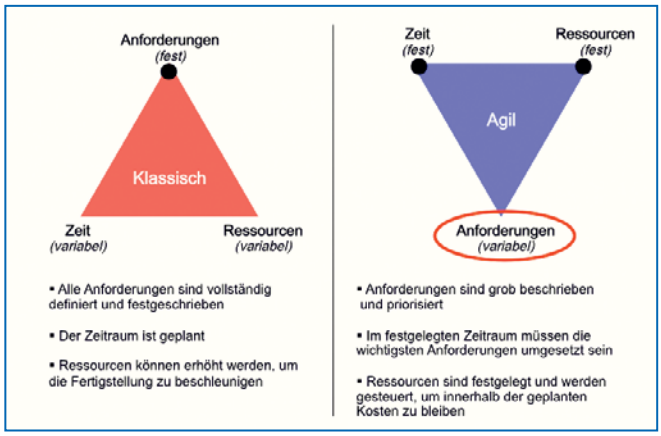


Abb. 1 Das Dreieck des Projektmanagements für klassische und agile Verfahren

Es gibt Notwendigkeiten, die über Projekten schweben

Neue Projekte werden nicht mit allen Freiheiten auf einer grünen Wiese gestartet, sondern im Kontext ihrer bestehenden klassischen Umgebung und Organisation. Daraus ergeben sich Restriktionen und Notwendigkeiten, an die sich alle Projekte anpassen müssen. Das können einerseits rechtliche Rahmenbedingungen, branchenübliche Standards oder Vorschriften sein, andererseits bewährte bestehende Systeme und Prozesse im Unternehmen. All das schränkt den Spielraum ein.

Nicht zuletzt ist es das Interesse des Kunden, Planungssicherheit hinsichtlich seiner Investition zu erhalten. Aber was genau wird bei agiler Vorgehensweise die Leistung sein, die er erhält? Wie erklärt er seinem Controller, dass er 1,5 Millionen Euro ausgeben wird für ein neues System, das passen und auch funktionieren wird … aber kein ausführliches, festgelegtes Pflichtenheft hat. Kriterien, wie Kundenzufriedenheit und Passgenauigkeit einer Lösung – häufige Stärken agiler Vorgehensweisen – lassen sich nur schwer in einem Geschäftsbericht präsentieren und nur mittelbar in einer Bilanz abbilden.

Ob Anforderungen an Projekte und deren Planungs- und Berichtswesen aus der unbeeinflussbaren rechtlichen Umwelt herrühren, oder von Interessensgruppen innerhalb der Organisation – der Bedarf der Organisation an Daten und bestimmten Vorgehensweisen aus Projekten wird nicht vor einem agilen Projekt Halt machen.

Agile und klassische Prozesse müssen sich ergänzen!

Agile Vorgehensweisen zur Entwicklung von Produkten bieten klare Vorteile, denen gegenüber sich kein Unternehmen verschließen sollte. Bei der Integration aber können Welten aufeinander prallen: Die Anhänger der klassischen Vorgehensweisen wollen klare Leistungsbeschreibungen und Termine, die der agilen wollen im Team bestimmen, was wann getan wird.

Aufgrund dieser Überlegung erscheint es sinnvoll, je nach Sachlage bzw. Projekteigenarten verschiedene Vorgehensweisen zuzulassen. Ob Agilität angewendet wird, hängt davon ab, ob es für das jeweilige (Teil-)Projekt Vorteile bringt und ob die Mannschaft damit umgehen kann und will. Abhängig von dem Leistungsgegenstand kann es etwa bei einem hoch innovativen Thema unumgänglich sein, mit regelmäßig iterativ neu bewerteten Zielen zu arbeiten. In einem stark reglementierten Umfeld dagegen gibt es vielleicht keine Alternative zum Wasserfall-Modell. Daher muss es ermöglicht werden, dass klassisch und agil durchgeführte Projekte im selben Unternehmen und sogar im selben Gesamtprojekt neben- und miteinander koexistieren können. Wie könnte das aussehen?

Ein Unternehmen wird ein oder mehrere Business-Management-Systeme (BMS) betreiben, in dem alle Daten konsolidiert werden, die für die Unternehmenssteuerung notwendig sind und deren Nutzung auch aufgrund gesetzlicher Bestimmungen obligatorisch sein kann. Hier werden u. a. von Projekten laufend Daten zu Aufwand (Arbeitsstunden, externe Kosten, Material) und entsprechende Forecasts dazu erfasst. Alle Projekte müssen die dafür benötigten Informationen liefern. Selbstverständlich auch die agilen.

Voraussetzung dafür ist ein Projektmanagement (PM)-System, das je nach Projekt anpassbar und skalierbar ist. Eine Möglichkeit ist es, dem agilen Projekt ein Standard PM-System zuzuordnen, das die notwendigen Informationen liefert, die das BMS benötigt. Die Daten dafür können entweder im Projekt oder an einer darüber stehenden Stelle erfasst und konsolidiert werden. Letzteres bedeutet einen manuellen Mehraufwand, den es zu vermeiden gilt. Hilfreich können in dieser Situation deswegen integrierende Tools sein, die unterschiedliche Datenquellen sinnvoll verbinden.

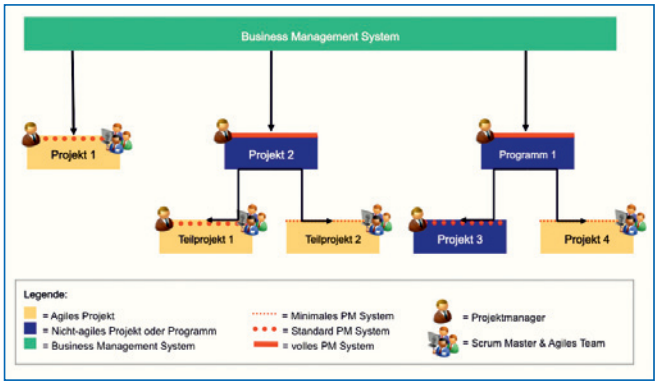


Abb. 2 Koexistenz verschiedener Projektvorgehensweisen

Für große Vorhaben, die nicht vollständig im agilen Kontext umgesetzt werden sollen oder können, ist die Lösung der Wahl eine gemischt klassisch/agile Projektstruktur. Die agilen Projekte sind dabei als Sub-Projekte des Gesamtprojekts oder -programms organisiert. Agil wird die eigentliche Entwicklung in den Expertenteams umgesetzt, während Dinge wie übergeordnete Zeitplanung, Kommunikation zum Management, Ressourcenmanagement, Infrastruktur- und Lieferantenmanagement sowie alles, was mit Finanzen zu tun hat, vom klassisch geführten Hauptprojekt abgewickelt wird.

Schnell wird klar, dass der Projektmanager mit seiner klassischen PMI- oder IPMA-Ausbildung in agilen Entwicklungsprojekten keineswegs obsolet ist. Ganz im Gegenteil: Die Schnittstelle zwischen Teilprojekten und die Kommunikation in Richtung Management und administrativen Einheiten eines Konzerns werden durch einen klassischen Projektmanager gebildet, während der Leiter des agilen Teams (zum Beispiel ein ScrumMaster) das Entwicklungsteam und die Kundenanforderungen im Fokus hat. Der klassische Projektmanager und der agile Teamleiter ergänzen sich so in ihrer Arbeit und haben keinen Widerspruch in ihrer Rolle.

Jedoch: Was immer im Bereich von Prozessen eines Unternehmens wünschenswert ist, wird erst praktisch umsetzbar durch eine ideal unterstützende Datenverarbeitung. Um auch in diesem Bereich nicht wieder getrennte Silos aufzubauen, muss darauf geachtet werden, dass die IT-Infrastruktur und die eingesetzten Werkzeuge eine vollständige Integration aller Informationen ermöglichen.

Was also wäre, wenn die Daten, die in einer Umgebung wie der Softwareentwicklung während des gesamten Lebenszyklus sowieso entstehen, einfach abgegriffen werden und für das Projektmanagement genutzt werden könnten? Wenn man sich überlegt, dass alle Arbeitsergebnisse eines (Software-) Entwicklungsteams digital sind und sogar fast der komplette Arbeitstag im virtuellen Raum stattfindet, dann ist die automatisierte Erhebung und Weiterverwendung von Informationen durchaus vorstellbar. Daten über offene Aufgaben, Bugs, Änderungsanforderungen, aktuelle Bearbeiter usw. entstehen doch originär nicht im Projektmanagementsystem, sondern tauchen praktisch zuerst in den Entwicklungstools auf.

Was bringt die Zukunft? Aktuell arbeiten die großen Hersteller genau daran, die Daten ihrer Tool-Familien so zu organisieren, dass sie über ein gemeinsames Interface zugreifbar und untereinander in Beziehung zu setzen sind. Eine der großen Initiativen auf diesem Gebiet ist die Jazz-Plattform, die als herstelleroffene Middleware für verschiedenste Tools zur Verfügung steht. Auf dieser gemeinsamen Basis können die Tools in einem Unternehmen barrierefrei Daten austauschen.

Zusammenfassung und Ausblick

Agile und klassische Vorgehensweisen bilden keinen Widerspruch zueinander. Im Gegenteil könne agile Ansätze genau dort Erfolg bringen, wo die klassischen Methoden bisher schwach waren: in der Entwicklung von Lösungen in einem dynamischen Umfeld mit unscharfen Kundenwünschen und Teams aus hochentwickelten Experten.

Um die Vorteile beider Welten auch in großen Unternehmen zu nutzen, kann eine agil/klassisch gemischte Projektstruktur angewendet werden, die je Teilprojekt unterschiedliche Vorgehensweisen zulässt. Dafür ist es notwendig, dass ein sinnvolles Maß gefunden wird, welche Daten für die Business-Management-Systeme und andere Systeme bereitgestellt werden müssen.

Leiter von agilen Teams konzentrieren sich auf den Entwicklungsprozess in den Teams und generieren so Qualität und Erfolg am Leistungsgegenstand der Projekte. Der klassische Projektmanager wird dabei immer die Aufgabe des Gesamtverantwortlichen und der zentralen Projektführungskraft behalten, er muss allerdings auch agile Methoden verstehen.

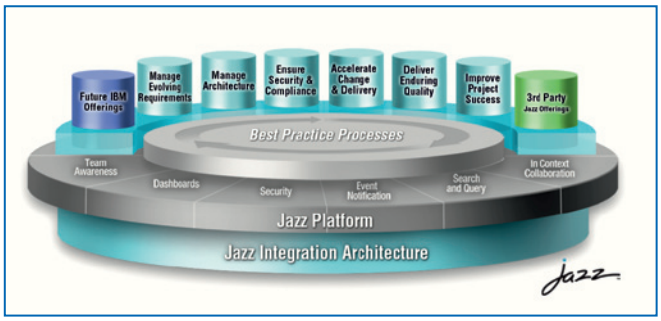


Abb. 3 schematische Darstellung der Jazz-Plattform - eine serviceorientierte Plattform zur Integration aller Projektaufgaben

Aktuell rückt die Effizienz der Tool-Landschaft von Projekten immer weiter in den Vordergrund. Eine intelligente und übergreifende Datenverarbeitung aller Tools im Entwicklungsprozess kann die Projektteams von Doppelerfassungen befreien und eine neue mehrdimensionale Transparenz für die Projektleiter schaffen. Neue Synergiepotentiale in der Zusammenarbeit werden genutzt.

Die hier vorgestellte Strategie bedeutet, so viel Agilität wie sinnvoll in einzelnen Projekten zu etablieren. Die Zukunft wird zeigen, ob ganze Organisationen (nicht zuletzt im Dienste des Kunden) agile Werte hinsichtlich ihrer Veränderungsbereitschaft, Flexibilität und Interaktion adaptieren können, so dass es heißen wird: „Welcome to agility.“

Thomas Müller

PMP® und Certified IBM Senior Projektmanager.
Über 18 Jahre Erfahrung als Projektleiter von internationalen Großprojekten und in vielen Bereichen der Hard- und Softwareentwicklung.

Aktuell bei IBM Rational als Berater und IT-Spezialist auf den Gebieten Prozess-, Portfolio- und Projektmanagement sowie Web-Anwendungssicherheit. Als „Community of Practice (CoP) Leader Project- and Portfolio-Management“ zudem mit dem aktiven Austausch zu diesem Thema innerhalb und außerhalb der IBM betraut.
(thomas.mueller@de.ibm.com)

MCI – Agile Verbesserung der Softwareentwicklung

Vorgehensweise zur Einführung agiler Methoden

Die Verbesserung der Softwareentwicklung bzw. von Softwareentwicklungsprozessen ist zwar bis heute keine Wissenschaft, aber schon über Jahrzehnte in der Praxis etabliert.

Das heißt, es gibt langjährige Erfahrungswerte, ja sogar Richtlinien und Modelle dazu, wie CMMI® (Capability Maturity Model® Integration [CMM06]) des Software Engineering Instituts. Gerade das CMMI-Modell gibt klare Bewertungskriterien zur Zielerreichung in der Qualität der Softwareentwicklung vor und wurde in den letzten Jahren in Deutschland weit verbreitet. Der Weg zum vorgegebenen Ziel ist dabei aber kaum beschrieben.

Wenn man diesen Weg zum Ziel wiederum als Prozess betrachtet, so gibt es bislang keine standardisierten und überhaupt kaum schriftlich definierte Vorgehensweisen.

IBM Rational hat nun seine eigenen langjährigen Erfahrungen aus der Verbesserung der Softwareentwicklung in unzähligen Unternehmen auf der ganzen Welt und in verschiedensten Industrien in eine definierte Vorgehensweise gepackt. Diese macht früher erarbeitete Vorgehensweisen, Artefakte und Erfolgsfaktoren wieder verwendbar.

Measured Capability Improvement – MCI

Unter dem Namen Measured Capability Improvement, kurz MCI, hat IBM Rational Vorgehensweisen, Services und Artefakte zur Verbesserung der Softwareentwicklung zu einem Paket zusammengefasst. Dabei stehen insbesondere wieder verwendbare Artefakte, wie Fragebögen, Spreadsheets und Folien zur Verfügung. IBM entwickelt das ganze Paket ständig weiter. So konnte das Unternehmen im Juni 2009 bereits das erste überarbeitete Release definieren (noch unter dem Namen MCIF - vgl. [Kro08]). Seit Anfang 2010 gibt es das neueste Release, das in MCI umbenannt wurde.

Diese Methode stellt keinen standardisierten Prozess dar, den IBM als Informationssammlung verkauft. Vielmehr ist es ein Baukasten, den IBM Servicemitarbeiter in Projekten zur Verbesserung der Softwareentwicklung ihrer Kunden nutzen.

Beim Design dieser Methode wurde besonderer Wert auf zwei Eigenschaften gelegt, die sich teils schon aus dem Namen ableiten lassen:

- *Measured*
  Jede Verbesserung, jeder Fortschritt soll durch vorher definierte Kennwerte gemessen werden. Dies kann auf Ebene der Geschäftsziele durch KPIs (Key Perfomance Indicators) erfolgen, aber auch auf der Detailebne einer einzelnen Praktik der Softwareentwicklung (zum Beispiel die Anzahl der Änderungen betreffend).
- *Capability Improvement*
  Verbesserung der Softwareentwicklung. Dabei wird besonderer Wert darauf gelegt, dass jede Verbesserung nachvollziehbar einem Geschäftsziel des Unternehmens dient und auch dementsprechend priorisiert wird.

Wichtig ist auch die Idee, Verbesserungen iterativ nach Priorisierung in kleineren Schritten durchzuführen und nicht einen komplett neuen Prozess in einem großen Wurf einzuführen.

Praktiken in der Softwareentwicklung

Mit der Idee, die Softwareentwicklung iterativ zu verbessern, anstatt einen komplett neuen Prozess einzuführen, musste auch die Beschreibung der Methodik, die bislang als vollständiger Prozess in Form des IBM Rational Unified Process (vgl. [Ess07]) vorlag, geändert werden. Für die iterative Vorgehensweise beschreibt man die Methodik in möglichst voneinander unabhängigen Praktiken, die sich in der Softwareentwicklung bewährt haben. Derzeit sind insgesamt 22 Praktiken beschrieben, die aus unterschiedlichsten Prozessen und Methodiken kommen. Insbesondere entsprechen viele Praktiken den agilen Methoden (zum Beispiel „Whole Team“).

Abbildung 1 gibt einen Überblick über die verfügbaren Praktiken.

Dadurch existiert eine Art „Baukasten“ für den agilen Entwicklungsprozess, aus dem man sich beliebig bedienen kann.

Die Praktiken sind umfangreich mit Hilfe des IBM Rational Method Composer beschrieben. Das heißt, es liegen Quellbibliotheken für die Beschreibung der Methodik vor, die man selbst nach Bedarf ergänzen, anpassen, kombinieren und als HTML-Seiten veröffentlichen kann.

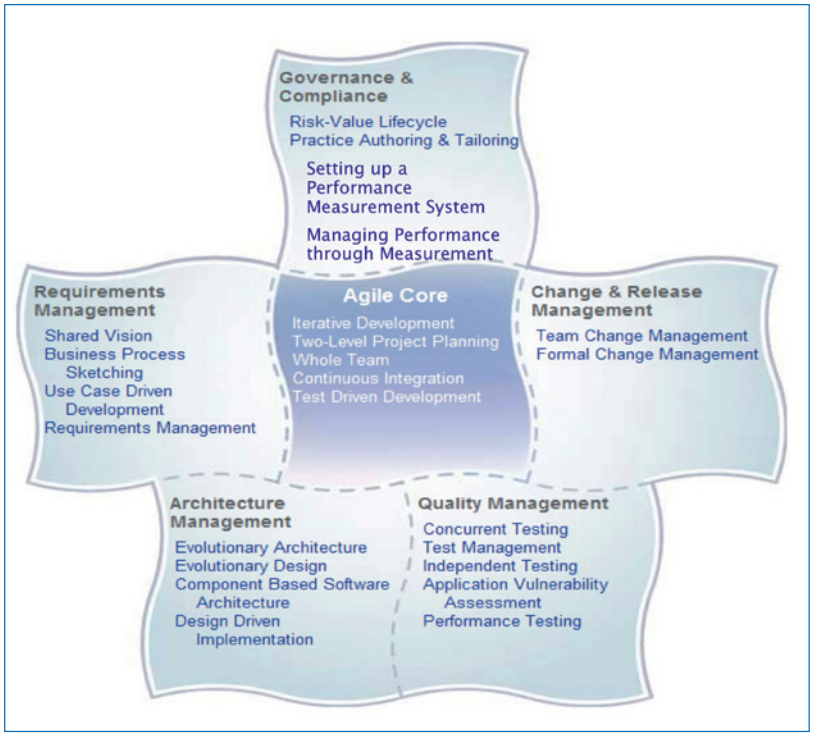


Abb. 1 Überblick Praktiken

Die fertige Beschreibung enthält neben der eigentlichen Methodik wichtige Hinweise:

- Wann eine Praktik sinnvoll ist (welche operationalen Ziele unterstützt sie?).
- Wie die Praktik eingeführt werden kann.
- Welche Kennwerte zur Messung des Fortschritts bei der Verbesserung denkbar sind.

Überblick über MCI

Abbildung 2 gibt einen Gesamtüberblick über die Vorgehensweise des MCI:

Abb. 2 MCI Überblick

Phase 1: Festlegen der Geschäftsziele und der operationalen Ziele.
In einem Workshop (ca. 4 Stunden) werden die Geschäftsziele des Unternehmens ermittelt und daraus operationale Ziele abgeleitet. Dazu dienen auch fertige Graphen, die aufzeigen, welche operationalen Ziele unter bestimmten Geschäftszielen in Frage kommen.

Phase 2: Priorisieren der Praktiken und des Fahrplans.
Die gleichen Graphen, die eine Verbindung von operationalen Zielen zu den Geschäftszielen aufzeigen, geben auch Hinweise auf die Frage, welche Praktiken in der Softwareentwicklung gegebene operationale Ziele unterstützen können.

Um die Tiefe und Qualität zu beurteilen, wie die ermittelten Praktiken derzeit in Projekten oder im Unternehmen umgesetzt werden, wird eine genauere Ist-Analyse der Softwareentwicklung des Unternehmens vorbereitet und durchgeführt. Dabei kann die Analyse von einzelnen Projekten bis hin zum gesamten Unternehmen über weltweit verteilte Standorte variieren. Der Aufwand hierfür liegt zwischen zwei bis sechs Wochen mit ein bis fünf Mitarbeitern. In folgenden Iterationen wird primär neu priorisiert, daher wird der Aufwand dann nur noch ein bis zwei Tage mit einem unterschiedlich starken Team betragen.

Natürlich sollten sich die Beteiligten dabei gezielt auf Praktiken fokussieren, die die gegebenen operationalen Ziele beeinflussen. Als Ergebnis wird der Ist-Zustand klar aufgezeigt und ein Fahrplan erarbeitet, um diesen Ist-Zustand iterativ zu verbessern. Das heißt, man konzentriert sich auf die wichtigsten Praktiken zuerst – etwa eben auf diejenigen, die den schnellsten Return on Investment (ROI) bringen. Dabei ist es auch ganz wichtig, Kennwerte festzulegen, die es erlauben, die Verbesserung später zu beurteilen.

Phase 3: Einführung mit Tools und Services begleiten.
Nachdem alle Beteiligten den Fahrplan abgestimmt haben, können die beschlossenen Verbesserungsmaßnahmen umgesetzt werden. Dabei werden bestimmte Praktiken in einer de-

finierten Art und Weise in ein oder mehreren Projekten eingeführt oder verbessert. Um die Iterationen der Einführung nicht zu groß werden zu lassen, sollte diese Phase zwischen zwei und maximal sechs Monaten dauern.

Dabei ist es in der gesamten Einführungsphase wichtig, mit Hilfe der in Phase 2 bestimmten Kennwerte die Verbesserung fortlaufend zu hinterfragen. Das kann neben Zahlen, die automatisch aus den Entwicklungswerkzeugen bezogen werden, auch eine Beurteilung des Projektteams in Form eines Fragebogens sein.

Neben Serviceleistungen in Form von Konfiguration, Beratung, Coaching und Schulungen gehören insbesondere in diese Phase Tools zum MCI. Wichtiger Bestandteil davon ist das neue Werkzeug IBM Rational Insight. Dieses basiert auf Cognos-Business-Intelligence- und Datawarehousing-Technologie. Es kann unterschiedlichste Datenquellen (Entwicklungswerkzeuge) nutzen, um diese Daten wiederum beliebig als Kennwerte zur Messung des -Fortschritts in der Softwareentwicklung aufzubereiten. Ebenso gibt es einen SelfCheck-Fragebogen, mit dem alle Teammitglieder den Fortschritt in einzelnen Punkten beurteilen und diese Beurteilung wiederum graphisch aufbereiten können. Und auch der IBM Rational Method Composer gehört dazu, mit dem die neu definierte oder verbesserte Praktik beschrieben und als HTML-Seite veröffentlicht werden kann.

Phase 4: Berichten, analysieren und anpassen.
Zum Ende jeder Einführungsiteration muss der erreichte Fortschritt möglichst in Form der definierten Kennwerte festgestellt und berichtet werden. Eine Analyse der erreichten Verbesserung und der „Lessons learned" der letzten Iteration und eine neue Bewertung der Ziele führen zur Anpassung des Fahrplans und einem detaillierten Plan für die nächste Iteration. Hierfür sollte man noch mal ein Aufwand von ein bis fünf Tagen mit wechselnd starkem Team veranschlagen.

Definierte Servicekomponenten des MCI

Die bislang beschriebene gesamte Vorgehensweise des MCI ist in definierte Servicekomponenten unterteilt, die einzeln oder in beliebiger Kombination eingesetzt und angepasst werden können.

Executive-Business-Value-Workshop: ein halber Tag Workshop zum Ermitteln der Geschäftsziele und der dazu gehörigen operationalen Ziele, die die Softwareentwicklung in den nächsten drei Jahren betreffen werden. Ziel ist es, ein gemeinsames Verständnis dieser Ziele und deren Prioritäten zu entwickeln.

Health-Assessment: Ermittlung und Ist-Analyse der Stärken und Schwächen der momentanen Softwareentwicklung. Dazu werden Fragebögen zur Selbsteinschätzung, Interviews und Analysen von Unterlagen genutzt. Aufgrund der ermittelten Ergebnisse erarbeitet man einen Vorschlag zur Optimierung der Softwareentwicklung, also welche Praktiken wie und mit welcher Priorität zu verbessern sind. Ebenso werden die Hindernisse und Hürden auf dem Weg zu dieser Optimierung aufgezeigt und Kennzahlen zum Messen der Verbesserung festgelegt. Daraus ergibt sich ein Fahrplan für das weitere Vorgehen.

Rapid-Deployment-Package: Nach einer in IBM Rational Method Composer beschriebenen und angepassten Vorgehensweise zur Einführung werden die ermittelten Praktiken iterativ eingeführt bzw. verbessert. Dazu werden Entwicklungswerkzeuge eingeführt und/oder angepasst, die konkrete Methodik der Praktiken angepasst und beschrieben, die Mitarbeiter geschult und in den ersten Schritten begleitet sowie der Fortschritt fortlaufend anhand der definierten Kennzahlen gemessen.

Self-Check: Einführung und Coaching in der Nutzung eines Werkzeugs zur einfachen Selbsteinschätzung der Projekte. Per Fragebogen ermittelt das Projektteam, wie gut bestimmte Praktiken im Projekt umgesetzt werden. Damit können die Teammitglieder in der Verbesserungsphase laufend bestimmen, wo das Projekt nach Einschätzung aller bereits steht.

IBM Rational Insight Quick Start: Mit Hilfe eines Performance-Measurement-Workshop bestimmt man die zu ermittelnden Metriken für die Entwicklungsverbesserung. Das Werkzeug IBM Rational Insight zum automatischen Reporting der Metriken wird installiert und konfiguriert. Schließlich können vorhandene Reports angepasst oder neue Reports erstellt werden.

Vorteile des MCI

Mit dem MCI wurde eine ganzheitliche Vorgehensweise zur Verbesserung der Softwareentwicklung – insbesondere zu mehr Agilität – wieder verwendbar gemacht, die es erlaubt:

- alle Verbesserungen in der Softwareentwicklung an Geschäftszielen zu orientieren – und das mit jederzeit nachvollziehbaren Abhängigkeiten,
- die Verbesserung entsprechend der Situation im Unternehmen in einzelnen Praktiken iterativ durchzuführen – im Gegensatz zu einer großen Entwicklungsprozesseinführung,
- die Verbesserung laufend objektiv zu kontrollieren und die Verbesserungsmaßnahmen entsprechend zu korrigieren,
- über Reports mit definierten und automatisch ermittelten Metriken den Fortschritt nachzuweisen,
- auf umfangreiche praktische Erfahrungen in Form von fertigen Artefakten zurückgreifen zu können

Abgrenzung zu CMMI: Um die im CMMI definierten Qualitätsziele (vgl. [CMM06]) in der Softwareentwicklung zu erreichen, kann MCI einen effizienten Weg aufzeigen, das heißt CMMI und MCI ergänzen sich eher, als dass sie in Konkurrenz stehen. Die im MCI beschriebenen Praktiken decken sich allerdings nicht 100% mit den CMMI Zielen. ■

Literatur & Links

[CMM06] Carnegie Mellon Software Engineering Institute, CMMI for Development, Version 1.2, August 2006

[Kro08] Per Kroll, IBM Whitepaper, The Measured Capability Improvement Framework: a systematic approach to software delivery excellence, November 2008

[Ess07] A. Essigkrug, T. Mey,
Rational Unified Process kompakt, Elsevier, 2007

Andreas Essigkrug

Teamleader IBM Rational Service für Prozess-, Produkt-, Projekt-, Portfoliomanagement und Enterprise Modernization. Koautor von „Rational Unified Process kompakt". Aufgabenschwerpunkte in den Bereichen Softwareentwicklungsprozesse und Werkzeugunterstützung für den gesamten Softwareentwicklungszyklus.
(andreas.essigkrug@de.ibm.com)

Agile Entwicklung in verteilten Teams durch Kollaboration und Kommunikation

Viele heutige Softwareentwicklungsorganisationen haben die Notwendigkeit zur Globalisierung erkannt und bereits mehr oder weniger in ihrem Unternehmen umgesetzt. Oft sind mehrere Entwicklungslokationen weltweit untereinander vernetzt. Für die zugehörigen Entwicklerteams bedeutet dies, dass sie global zusammenarbeiten müssen – meist über mehrere Zeitzonen hinweg.

Teilweise organisieren sich solche globalen Teams auf Basis mehr oder weniger klar abgrenzbarer Aufgabenbereiche. Beispielsweise kann ein Team in Lokation A mit der Gesamtintegration gewisser Komponenten betraut sein, während weitere Teams in den Lokationen B und C für die Entwicklung und Bereitstellung dieser weitgehend unabhängigen Komponenten verantwortlich sind. In dieser Art der Organisation der Entwicklerteams kommunizieren die verschiedenen Teams untereinander zumeist über den jeweiligen Manager, Projektleiter oder leitenden IT-Architekten. Die Zusammenarbeit der Teams erfolgt in einem solchen Fall eher intern und nur vereinzelt – insbesondere bei Integrationspunkten der Applikationen – über Lokationsgrenzen hinaus. Verteilte Teams mit Mitgliedern in mehreren unterschiedlichen Lokationen sind aufgrund der Herausforderungen in Bezug auf Teamintegration, globale Zusammenarbeit und Kommunikation eher noch die Ausnahme. Der Trend geht jedoch mehr und mehr hin zu global integrierten Teams.

Agile Entwicklungsprozesse sehen einen ausgewogenen Mix an Skills innerhalb des Teams vor (siehe Artikel „Agile – Was nun?, Seite 12. Darüber hinaus propagiert das agile Manifest, dass Individuen und Kommunikation/Kollaboration wichtiger sind als die zugrunde liegenden Prozesse. Dieser Artikel beschreibt die Möglichkeiten und notwendigen Maßnahmen zur Etablierung von global integrierten Teams auf der Grundlage agiler Entwicklungsprozesse. Wichtige Rahmenbedingungen für Teambildungsprozesse werden erläutert.

Teambildung und kulturelle Einflüsse

Kulturelle Einflüsse und Rahmenbedingungen sind wesentliche Aspekte, die man vor der Bildung von verteilten Teams unbedingt beachten sollte. Dies gilt nicht nur bei agilen Entwicklungsprozessen. Je nach Projektlaufzeit und -umfang bietet es sich an, einen entsprechenden Berater für die Phase der Teambildungsprozesse hinzuzuziehen.

Solche Berater haben sich auf die jeweiligen landestypischen kulturellen Rahmenbedingungen spezialisiert. Sie kennen kulturelle Besonderheiten und können mit Hilfe dieses Wissens die unterschiedlichen Teammitglieder bezüglich kulturspezifischer Verhaltensweisen unterrichten. Missverständnisse, gegenseitiges Unbehagen und andere teaminterne Konflikte, die ihren Ursprung in kulturellen Unterschieden haben, können dadurch vorab vermieden oder während des Projektes gelöst werden. Bei der Bildung global integrierter Teams müssen sich die Manager der unterschiedlichen Entwicklungsbereiche der Sprach-, Kultur- und Zeitzonenbarrieren bewusst sein und Wege finden, um diese Hindernisse zu überwinden. Die Globalisierung führt in professionellen Umgebungen langsam, aber beständig einerseits zu einer Sensibilisierung in Bezug auf spezifische kulturelle Unterschiede und lässt Teammitglieder andererseits die eigenen kulturellen Eigenarten weitgehend ablegen

Zeitzonenproblematik

Eine der Grundprinzipien agil arbeitender Teams ist die offene und häufige Kommunikation zwecks optimaler Zusammenarbeit. Um dies bei global integrierten, über unterschiedliche Zeitzonen verteilten Teams zu ermöglichen, sind einige wesentliche Aspekte bei der Zusammenstellung solcher Teams zu beachten. Die nachfolgende Liste enthält Best Practices für die Teamzusammenstellung, die sich in der Praxis bewährt haben. Räumlich zusammensitzende Teams sind generell verteilten Teams vorzuziehen. Da dies aus beschriebenen Gründen jedoch meist nicht möglich ist, wird dieser Punkt nicht weiter diskutiert.

Abb. 1 Verteilte Teams – geografische Betrachtung

Bewährte Best Practices bei der Zusammenstellung agiler, verteilter Teams

- Minimierung der Anzahl der Zeitzonen pro Team
- Mitglieder aus möglichst nahe beieinander liegenden Zeitzonen
- Annähernde Gleichverteilung der Teammitglieder pro Lokation

» „Minimierung der Anzahl der Zeitzonen pro Team“ bedeutet hier, dass die verschiedenen Mitglieder eines Teams aus so wenig wie möglich unterschiedlichen Zeitzonen stammen sollten. Idealerweise besteht ein Team aus Mitgliedern ein und derselben Zeitzone. Erfahrungen aus der Praxis haben gezeigt, dass sich die Mitglieder eines Teams aus nicht mehr als drei Zeitzonen zusammensetzen sollten. Hintergrund dieses Punktes ist, dass nur dann eine ausreichende Zusammenarbeit im Team erfolgen kann, wenn genug Möglichkeiten zur Kommunikation innerhalb des Teams existieren. Es muss genügend Freiraum vorhanden sein, um – spontan oder geplant – täglich miteinander im Team zu kommunizieren. So muss das Team etwa technische oder organisatorische Fragen oder den generellen Projektfortschritt diskutieren.

» Der Punkt „Mitglieder aus möglichst nahe beieinander liegenden Zeitzonen“ hat den gleichen Hintergrund. Je

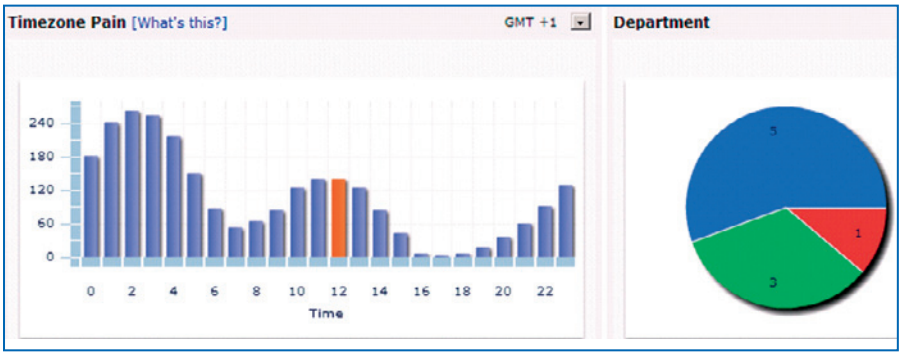


Abb. 2 Verteilte Teams – Zeitzonenbetrachtung

mehr Stunden pro Tag sich die Arbeitszeiten der einzelnen Teammitglieder überlappen, desto höher ist das Maß an möglicher Kommunikation im Team. Praktische Erfahrungen haben gezeigt, dass eine Zeitverschiebung zwischen Teammitgliedern von bereits neun Stunden die Möglichkeiten der Kommunikation negativ beeinträchtigt. Idealerweise sollten sich die Arbeitszeiten der Mitglieder eines Teams zu mindestens 30% überschneiden. Bei einem Achtstundentag entspricht das in etwa zwei bis drei Stunden. Das kommt einer maximalen Zeitverschiebung von ca. sechs Stunden gleich. Bei größeren Abweichungen sollte die Projektleitung eine Verschiebung der Arbeitszeiten der Teammitglieder einer Lokation in Erwägung ziehen.

» Der Punkt „Annähernde Gleichverteilung der Teammitglieder pro Lokation“ betrifft den Teambildungsprozess. Erfahrungen aus der Praxis zeigen, dass ein global integriertes Team aus in etwa der gleichen Anzahl von Teammitgliedern pro Lokation bestehen sollte. Dieser Aspekt wirkt positiv auf den Teambildungsprozess und bewirkt eine Art Gleichwertigkeit der verschiedenen Lokationen und deren Teammitgliedern. Teams mit dem Schwerpunkt in einer Lokation und lediglich ein bis zwei Mitgliedern aus anderen Lokationen führen in der Regel dazu, dass die Mitglieder der schwächer vertretenen Lokation oft eine weniger wichtige Rolle im Team einnehmen können und/oder sich nicht voll integriert fühlen.

Die Herausforderungen zur Bildung global verteilter, agil arbeitender Teams sind groß. Um negative Erfahrungen zu vermeiden, ist es Aufgabe des Managements bzw. der gehobenen Projektleitung, die oben angeführten Punkte bei der Bildung agiler Entwicklungsteams zu beachten.

Tools für die Zusammenarbeit

Um die Kommunikation in global vernetzten, agilen Teams zu unterstützen, ja sogar zu ermöglichen, bedarf es der notwendigen Infrastruktur sowie einiger wesentlicher, in die Geschäftsprozesse des jeweiligen Unternehmens integrierte Tools. Wichtige Infrastrukturkomponenten sind breitbandige Internetzugänge, um die vielfältigen Kommunikationsansprüche der Tools zu ermöglichen, sowie kostengünstige Telefoniermöglichkeiten, wie etwa Firmennetz- oder VoIP-Telefonie. Wichtige Komponenten der Tool-Landschaft sind Instant Messenger, E-Mail-Clients und eine teamübergreifende Kalenderfunktionalität. Diese ist in der Regel in den E-Mail-Client integriert und bietet unter anderem Möglichkeiten wie Verfügbarkeitsüberprüfung bei Erstellung von Meeting-Einladungen, integrierte Zeitzonenberücksichtigung und organisatorische Teamübersicht. Bei einer zeitzonenüberschreitenden Teamzusammensetzung ermitteln diese Tools die für alle Beteiligten beste Zeit für Besprechungen (Abbildung 2).

Weitere wichtige Tools für die Zusammenarbeit sind Web 2.0-basierte Intranet-/Internet-Portale wie Wikis und Blogs. Gerade in agilen Teams ist es wichtig, dass alle Teammitglieder sowohl alle für sie bestimmten Informationen lesen als auch anderen Teammitgliedern strukturiert zur Verfügung stellen können. Heutige Konzepte erlauben es, sich zu bestimmten Themen zu registrieren. Sobald eine Änderung auf der entsprechenden Seite erfolgt, erhält man eine Benachrichtigung per E-Mail. Derartige Wikis sind sehr effektive Hilfsmittel zum Verteilen von Informationen im Team.

Agil arbeitende Teams sind noch mehr als andere Teams von der möglichst durchgängigen Integration der Tools miteinander angewiesen. Individuen und Wechselwirkungen besitzen laut agilem Manifest Vorrang vor Prozessen und Tools. Die Effizienz der Tool-Landschaft von Projekten rückt dadurch immer weiter in den Vordergrund. Durch intelligente und übergreifende Datenverarbeitung aller Tools im Entwicklungsprozess müssen die Projektteams Daten nicht mehrfach erfassen. Es entsteht eine neue, mehrdimensionale Transparenz für die Projektleiter. Neue Synergiepotenziale in der Zusammenarbeit werden frei. Der Umgang mit dieser neuen Transparenz und Qualität der Tools im Projektmanagement wird ein spannendes Lernfeld für Projektmanager und Unternehmen in den nächsten Jahren sein.

Last but not least bildet ein gemeinsames Source-Code-Verwaltungssystem den Kern der Kollaborationsplattform. Der Einsatz solcher Systeme ist heutzutage bereits De-facto-Standard. Neu auf diesem Gebiet ist die durchgängige Integration der verschiedenen Phasen des Softwareentwicklungszyklus auf einer gemeinsamen Datenbasis. Anforderungsdefinition, Architektur, Design, Implementierung und Test arbeiten – ohne unnötige mehrfache Datenerfassung – auf einem gemeinsamen Repository. Das offene Jazz Framework bietet einen interessanten Ansatz zur Tool-Integration miteinander (http://jazz.net)

Literatur und Links

[Cohn08] Mike Cohn, „Agile Estimating and Planning“, Prentice Hall, 2008

[Pich08] Roman Pichler, „Scrum – Agiles Projektmanagement erfolgreich einsetzen“, dpunkt.verlag, 2008

[Chin04] Gary Chin, „Agile Project Management – How to succeed in the face of Changing Project Requirements“, Mcgraw-Hill Professional, 2004

[Schwa04] Ken Schwaber, „Agile Project Management with Scrum“, Microsoft Press, 2004

Oliver Röhrsheim

Zertifizierter Senior IT Architect IBM Softwaregroup, PMP®. Aufgabenschwerpunkte in den Bereichen IT-Architektur, Software-Entwicklungsprozesse, Agiles Project Management.
(o.roehrsheim@de.ibm.com)

Kontinuierliche Integration für kontinuierliche Softwarequalität

In der agilen Softwareentwicklung wird eine hohe, kontrollierte Dynamik in Projekte eingeführt, welche es ermöglicht in kurzen Abständen fertige Softwareversionen zu liefern. Je größer ein System dabei wird, desto mehr Aufwand muss für die Integration der einzelnen Module aufgewendet werden. Schnell wird mehr Zeit in die Anwendungsintegration, als in neue Funktionen investiert. Eine automatische und kontinuierliche Integration der Software schafft hier Abhilfe.

Durch die kontinuierliche Integration wird in einem Realisierungszyklus sichergestellt, dass den Entwicklern und Testern jederzeit eine lauffähige Anwendung zur Verfügung steht. Es ist dabei unerheblich, nach welchen Methodiken ein Projekt Software entwickelt. Am Ende wird immer ein fertiges Produkt gebaut und integriert. Je früher und regelmäßiger dieses Produkt erzeugt wird, desto eher können Probleme erkannt und die Anwendungsqualität erhöht werden.

Versionsverwaltung der Software als Basis

Die Grundlage für die kontinuierliche Integration bildet ein Konfigurationsmanagementsystem. Dabei können sowohl zentralisierte Systeme, wie Subversion, oder auch dezentrale Systeme, wie Git zum Einsatz kommen. Wichtig ist, dass jeder Entwickler in regelmäßigen Abständen seinen Arbeitsstand versioniert und anderen Entwicklern und zentralen Build-Systemen zur Verfügung stellt. Alles was nicht im Repository vorhanden ist, wird als nicht-existent betrachtet („single source of truth“). Immer wenn sich die Software in der Versionsverwaltung ändert, wird automatisch das Bauen und Integrieren der Anwendung angestoßen. In größeren Systemen, welche langlaufende Builds benötigen, kann es dabei sinnvoll sein, Änderungen zu sammeln und diese nur in fest definierten Zeitabständen zu bauen. Vertreter für Build-Systeme sind dabei BuildForge, Rational Team Concert, Hudson oder CruiseControl. Die verschiedenen Build-Systeme unterstützen eine Vielzahl an Versionsverwaltungen und Build-Varianten (Ant, Maven Shellskripte, etc.).

Die kontinuierliche Integration alleine führt jedoch noch nicht zwangsläufig zu einer besseren Softwarequalität. In den Projektteams ist es notwendig ein Verständnis zu schaffen, dass ein gebrochener Build (etwa durch Fehler beim Kompilieren oder bei der Testausführung) erst gefixed werden muss, bevor man sich mit der Umsetzung neuer Aufgaben beschäftigt. Niemandem ist geholfen, wenn das Build-System regelmäßig über Fehler informiert und keiner darauf reagiert. Hier haben Systeme einen Vorteil, welche sich direkt in die Entwicklungsumgebung integrieren. Mit Rational Team Concert zum Beispiel werden die Entwickler sofort informiert (Pop-up, E-Mail, RSS Feed etc.), wenn ein Fehler im Build aufgetreten ist und man kann sich Details zum Build direkt in der Entwicklungsumgebung anschauen. In kleinen, nicht verteilten Teams kann es zusätzlich hilfreich sein eine Infowand zu installieren, welche den aktuellen Stand des Builds für jeden sichtbar darstellt. Nur wenn ein fehlerfreier Build und eine fehlerfreie Integration möglich sind, kann auch eine fehlerfreie Software ausgeliefert werden.

Kontinuierliches automatisches Testen

Ein weiterer Schritt zur Qualitätserhöhung ist das automatisierte Testen. Im Java-Umfeld haben sich dabei Unit-Tests mit JUnit etabliert. Zu jeder geschriebenen Java-Klasse wird immer auch ein Unit-Test erzeugt. Über den Unit-Test stellt der Entwickler sicher, dass seine Software fehlerfrei funktioniert und auch auf Fehlersituation angemessen reagiert. Diese automatischen Tests werden ebenso in die Versionsverwaltung eingestellt und direkt nach dem Build der Anwendung ausgeführt. Über die kontinuierliche Integration stellt so das Projektteam die Kompilierfähigkeit und auch die Lauffähigkeit der Anwendung auf Unit-Ebene sicher. Um die Erstellung der Unit-Tests stärker zu forcieren, wird das testgetriebene Entwickeln verwendet. In diesem Fall wird immer zuerst der Test implementiert („was soll mein Stück Software leisten?“), bevor mit der eigentlichen Umsetzung der Funktionalität begonnen wird. Die Software wird so direkt gegen den Test entwickelt. Erfolgreiche kontinuierliche Integration erfordert immer auch ein kontinuierliches Testen.

Reporting zur Qualitätssteigerung

Wie bereits angedeutet, ist ein weiterer Schritt zur Verbesserung der Softwarequalität die Verteilung von Informatio-

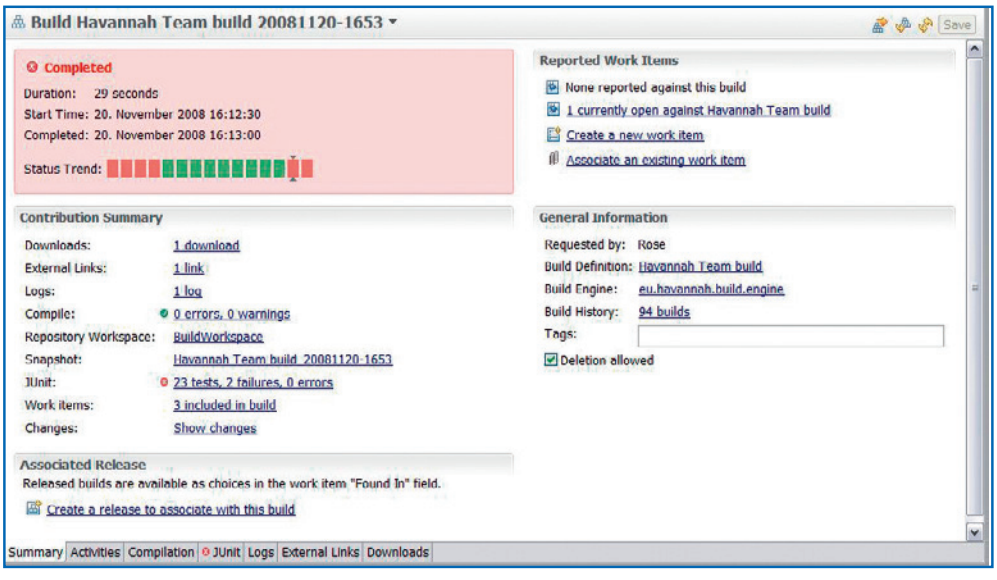


Abb. Buildintegration in Rational Team Concert

nen über den aktuellen Zustand der Software. Eine effektive kontinuierliche Integration ermöglicht es, dass jederzeit Auswertungen zur Software verfügbar sind. Neben den Kompilier- und Testinformationen können auch Softwaremetriken bereitgestellt werden. Durch den Vergleich der Daten mit vergangen Ständen können der Entwickler und der Teamleiter sehen, wie sich die Softwarequalität verbessert bzw. verschlechtert hat. Nehmen die fehlgeschlagenen Builds/Testfälle zu oder zeigen Qualitätsmetriken eine Verschlechterung, so sollte man einen Schritt zurückgehen und die Ursachen analysieren. Wenn nicht in regelmäßigen Abständen eine Integration und ein Test der Software durchgeführt werden, hat man keine Möglichkeit frühzeitig auf anstehende Probleme zu reagieren. Eine kontinuierliche Integration wird dann richtig erfolgreich, wenn Sie in die Arbeitsabläufe und auch in das Entwickler- bzw. Teamleiter-Tooling integriert ist (siehe Abbildung).

In der realen Welt lässt sich das Idealbild der kontinuierlichen Integration jedoch selten erreichen. Oft müssen Entwickler Arbeitsstände sichern, die noch nicht in die Gesamtanwendung integriert werden dürfen. Manchmal möchte man auch die Lauffähigkeit der letzten Änderungen integrativ testen, ohne dass das Gesamtergebnis einer Iteration oder reines Sprints beeinträchtigt wird. In solchen Fällen ist es hilfreich, wenn die Versionsverwaltung und auch das Build-Tooling private Softwarestände und private Builds unterstützt. Mit Rational Team Concert kann jeder Entwickler seine eigenen Versionen zentral pflegen und entscheidet selbst, welche Anpassungen allen Entwicklern und damit der Gesamtanwendung zur Verfügung gestellt werden. Zusätzlich kann der Entwickler eine private Integration der Anwendung über das Build-System anstoßen. Da der Build remote ausgeführt wird und der Rational Team Concert Server den Build auf verschiedene Build-Clients auslagern kann, wird der Entwickler in seiner Arbeit nicht behindert. Schon bei Softwaresystemen mittlerer Größe kann der Build inklusive Unit-Tests und Sammlung von Metriken viele Minuten bei voller Prozessorlast in Anspruch nehmen.

Fazit

Die kontinuierliche Integration ist ein zentraler Bestandteil der agilen Softwareentwicklung. Da jederzeit ein aktuelles Bild der Software verfügbar ist, kann man einfacher sicherstellen, dass nach einem Iterationszyklus auch eine lauffähige Software zur Verfügung steht, die den gesetzten Qualitätsanforderungen genügt. Die Basis der erfolgreichen kontinuierlichen Integration ist jedoch die Akzeptanz des Systems im Entwicklerteam. Das kontinuierliche Bauen ist keine Überwachung der Mitarbeiter, sondern eine Möglichkeit qualitativ hochwertige Software zu liefern und Fehler frühzeitig zu eliminieren.

Karsten Voigt

IT Architekt und IT Consultant seit vielen Jahren in verschiedenen Kundenentwicklungsprojekten.
Setzt sich Entwicklungsmethodiken und dem effektiven Einsatz von Entwicklungstools auseinander. Sprecher auf Konferenzen und Autor mehrerer Fachartikel.
(karsten.voigt@de.ibm.com)

impressum

Kontaktadresse für die Beiträge S. 3 bis 39:
IBM Deutschland GmbH
IBM-Allee 1
71139 Ehningen
URL: www.ibm.de

Herausgeber
SIGS DATACOM GmbH

Verlag
SIGS DATACOM GmbH,
Lindlaustr. 2c, D-53842 Troisdorf
Tel.: +49 (0) 22 41/23 41-1 00,
Fax: +49 (0) 22 41/23 41-1 99
www.sigs-datacom.de
E-Mail: info@sigs-datacom.de

Verlagsleitung
Günter Fuhrmeister

Redaktions- und Herstellungsleitung Zeitschriften
Susanne Herl, Tel.: +49 (0) 22 41/23 41-5 50,
E-Mail: Susanne.Herl@sigs-datacom.de

Schlussredaktion:
Heike Weidner

Bildmaterial Cover:
Gettyimages.de

Druck
Erdl Druck Medienhaus GmbH,
Gabelsbergerstr. 4–6, 83308 Trostberg

Abonnenten-Service
IPS Datenservice GmbH, Postfach 13 31,
D-53335 Meckenheim,
Tel.: +49 (0) 22 25/70 85-3 74
Fax: +49 (0) 22 25/70 85-3 76,
Patrick König, Markus Preis
E-Mail: aboservice@sigs-datacom.de

Erscheinungsweise OBJEKTspektrum
zweimonatlich

Bezugspreis Sonderheft
Deutschland € 3,80, Europa € 4,50

Bezugspreise OBJEKTspektrum
Einzelverkaufspreis: D: € 8,50, A: € 9,50, CH: sfr 15,60
Jahresabonnement Deutschland: € 48,00 inkl. Versand
Jahresabonnement Europa: € 55,20 inkl. Versand
Studentenabo: € 43,20 inkl. Versandkosten

Lieferung an Handel
Verlagsunion KG, Postfach 57 07,
65047 Wiesbaden, Tel.: +49 (0) 61 23/6 20-0

Get me approved, please! Lizenzkompatibilität von Open-Source Komponenten

Trotz ihres Siegeszuges in den letzten Jahren wird die Umsetzbarkeit der agilen Softwareentwicklung „in der Praxis" teilweise noch angezweifelt. Insbesondere Unternehmen mit traditionell recht formellen, „dokumentenorientierten" Prozessabläufen – etwa im Pharma-, System- oder Public-Sektor – sind nach Erfahrung der Autoren nur schwer von diesem „neuen" Ansatz zu überzeugen.
Teilweise beruht diese Einschätzung auf einem ungenauen Verständnis darüber, was „agile" überhaupt bedeutet – oder der Angst, dafür die komplette Organisation umkrempeln zu müssen.
In Diskussionen mit Prozessverantwortlichen wird ein „wasserfallartiger" Prozess oft als notwendiges Übel dargestellt, um alle internen und externen Prozessanforderungen erfüllen zu können. Iterativ-inkrementelles Vorgehen wird bereits als ein hohes Ziel angesehen – und „agile" als in der Praxis nicht wirklich realisierbar.

Interessanterweise zeigt die Umfrage unter [Amb09], dass agile Teams meist in einem reglementierten Prozessumfeld arbeiten müssen. Die zentrale Frage lautet also: Welche Bedingungen müssen grundsätzlich erfüllt sein, damit komplexe IT-Organisationen Agilität zumindest in Teilbereichen erfolgreich einführen und auch „leben" können.

Ein wichtiger Punkt ist nach Meinung der Autoren hierbei die effiziente Integration der unterschiedlichen Governance-Prozesse (siehe Kasten) mit dem eigentlichen Softwareentwicklungsprozess.

„License-Compliance" beim Einsatz von Open-Source-Software

Agile Prozesse wie XP oder Scrum „instanzieren" die agilen Werte und Praktiken selektiv für bestimmte Teilbereiche der „Entwicklungs"-Phase entlang der Wertschöpfungskette eines Produkts. Hierbei wird der Softwarelebenszyklus als mehr oder weniger isolierter Geschäftsprozess innerhalb der IT-Organisation betrachtet.

Unsere Erfahrung zeigt aber, dass oftmals auch sehr effiziente, agile Entwicklungsteams an den „Prozessschnittstellen" ausgebremst werden, etwa durch schwergewichtige Approval- oder Auditprozesse. Diese sind oft nicht kontinuierlich in einen iterativen Entwicklungsprozess integriert, sondern vor- oder nachgelagert oder laufen sogar neben dem eigentlichen Entwicklungsprozess. Dies erhöht das Projektrisiko enorm, da solche Compliance-Prozesse eine unmittelbare und weitreichende Auswirkung auf die Verfügbarkeit eines Software-Produkts haben können.

Im Folgenden stellen wir dies am konkreten Fall der „Lizenz-Konformität" beim Einsatz von Open-Source-Software (OSS) in kommerziellen Entwicklungsprojekten dar. Dieses Beispiel ist nach unserer Erfahrung in der Praxis ein sehr häufiger Anwendungsfall. Gleichzeitig ist dies oft ein vernachlässigtes und unterschätzes Thema.

Der Einsatz von Open-Source-Komponenten, die entgegen der vorliegenden Lizenzvereinbarungen eingesetzt werden, wird in der Praxis meist nicht entdeckt. Unter Umständen kann eine derartige vertragswidrige Verwendung aber hohen finanziellen Schaden anrichten – etwa durch Verlust der Reputation, Unterlassungsklagen oder Kosten, Softwarekomponenten nachträglich zu entfernen und neu zu entwickeln. Auch das komplette Zurückziehen eines bereits am Markt verfügbaren Produkts ist hierbei denkbar.

Ein Beispiel für eine solche rechtliche Non-Compliance ist der Einsatz einer modifizierten bzw. erweiterten Open-Source-Bibliothek, die unter der GPL-Lizenz steht [GNU] und in einem kommerziellen Produkt integriert wurde, ohne den erforderlichen Quellcode offenzulegen. Zudem ist davon auszugehen, dass in diesem Fall der „virale" Effekt der GPL-Lizenz [COPY] grundsätzlich Auswirkungen auf alle Codeanteile der betroffenen Produktentwicklung haben wird.

Deshalb benötigen Unternehmen einen entsprechenden Compliance-Prozess, in welchem die Rechtsabteilung die Lizenzvereinbarungen von Open-Source-Komponenten überprüft, bevor diese für die Produktentwicklung eingesetzt werden dürfen.

Innerhalb dieses Prozesses werden die fraglichen Lizenzbedingungen auf Eignung für den geplanten Einsatz überprüft. Hierbei müssen auch Faktoren wie etwa Projekttyp (zum Beispiel Middleware mit eingebetteten OSS-Bibliotheken versus Appliance mit eingebettetem Linux-Betriebssystem) und generelle Unternehmensrichtlinien berücksichtigt werden.

Nach unserer Erfahrung unterstützen drei Maßnahmen derartige Compliance-Anforderungen:

- Kontinuierliche Verfügbarkeit von Compliance-Expertise im Projektteam
- Durchgängige Prozessunterstützung zwischen Entwicklungs- und Compliance-Prozess mittels integrierter Entwicklungswerkzeugen
- Eine Kombination aus den zwei zuvor genannten Varianten

Agile Entwicklungsteams sollten zunächst die kontinuierliche Mitarbeit entsprechender Compliance-Experten im Projektteam sicherstellen bzw. von der Organisation zuverlässig

IT-Governance und Compliance

In diesem Artikel wird etwas vereinfacht zwischen IT-Governance (als Prozess) und Compliance (als Eigenschaft) unterschieden. Beispiele sind die Konformität zu Prozessmaturitätsstandards wie CMMI, SPICE, ISO 9002 oder generell selbstauferlegten Unternehmensrichtlinien.

Auferlegte Anforderungen sind etwa gesetzliche oder regulatorische Vorgaben wie Basel II, SOX oder die Bestimmungen der FDA. Der Einfluss solcher Anforderungen auf IT-Projekte variiert stark und hängt vom Kosten/Nutzen-Verhältnis des zu betreibenden Aufwands ab. Die möglichen Folgen von „Non-Compliance" liegen auf einem breiten Spektrum zwischen höheren internen Betriebskosten und massiven rechtlichen Sanktionen, etwa dem Verlust der Börsenzulassung des Unternehmens.

Zugriff auf entsprechende Expertise bekommen. Dies bedeutet etwa die Bennennung und Ausbildung eines „Open-Source-Spezialisten“ – als integriertes Teammitglied und als Schnittstelle zum Projekt- und Produktmanagement bzw. der Rechtsabteilung.

Das Team braucht effiziente Möglichkeiten, um „sichere“ Bibliotheken zu identifizieren und Informationen, welche zusätzlichen Anforderungen sich aus deren Einsatz ergeben (etwa das Bereitstellen der Lizenzvereinbarung im Endprodukt). Zudem muss die Schnittstelle zum Compliance-Prozess, etwa zum Bewilligen von neuen Komponenten oder Versionen, klar definiert sein und sich ohne großen bürokratischen Aufwand in den Entwicklungsprozess einfügen.

Um in diesem Spannungsfeld die Reibungsverluste zwischen IT-Prozess und Compliance-Prozess zu minimieren und das Projektrisiko der Non-Compliance möglichst früh (und kontinuierlich) im Lebenszyklus zu adressieren, ist nach unserer Erfahrung eine werkzeuggestützte Lösung mit durchgängiger Prozessunterstützung extrem hilfreich.

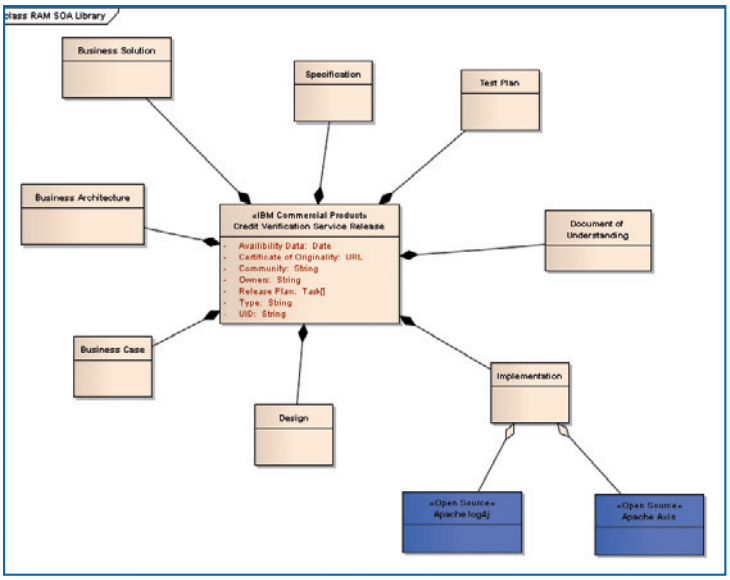


Abb. 1 Stark vereinfachte Asset-Struktur einer IBM Produkt Wertschöpfungskette

Im Folgenden stellen wir daher eine konkrete Werkzeugkette aus dem IBM Entwicklungslabor in Böblingen vor, welche die oben genannten Anforderungen adressiert.

Open-Source Asset Management innerhalb des Softwarelebenszyklus

Produktentwicklungsprojekte in der IBM obliegen einem Projektmanagement-Rahmenwerk ähnlich dem Inhalt des PMBOKs [PMBOK]. In allen Projektphasen ist das Thema Compliance in diesem Zusammenhang relevant. Hierbei geht es um Themen wie spezifische gesetzliche Vorschriften einzelner Länder, Exportkontrolle (zum Beispiel die Bereitstellung von kryptographischem Material), das Implementieren von publizierten Standards, aber auch die Nutzung von Software von Drittanbietern. Hier wiederum sind Nutzung und Integration von Open-Source-Software in IBM Produkten von herausgestellter Bedeutung.

Sämtliche während des Software-Lebenszyklus erstellten und wiederverwendeten Artefakte werden in der IBM Produktentwicklung erfasst. Auf Basis dieser Arbeitsprodukte wird ein sogenanntes „Certificate of Originality“ (COO) erstellt. Dieses Dokument ist der zu erbringenden Nachweis, dass ein Produkt ordnungsgemäß nach sämtlichen Urheberrechtsrichtlinien entwickelt wurde. Letztlich stellen ein IP-Rechtsanwalt und das Management diese „Compliance-Konformität“ anhand der im COO enthaltenen Daten fest. Erst nach der Freigabe des COOs ist ein Produkt reif für die „General Availability“, d. h. für die Verfügbarkeit des Produkts am Markt.

Sämtliche Artefakte müssen vorliegen, um nachweisen zu können, dass ein IBM Produkt entsprechend den Regeln entwickelt wurde. In der Praxis umfasst die vorschriftsgemäße Dokumentation der Wertschöpfungkette eines IBM Produkts mindestens zwischen 40 – 50 Artefakte. Open-Source-Material (siehe Abbildung 1, blau hervorgehoben) kann dabei in mehrfacher Ausprägung vorkommen.

Bei der Verwendung und Integration von Open-Source-Software in einem kommerziellen Produkt besteht eine Herausforderung darin, zu analysieren, woraus genau der Bedarf für deren Einsatz besteht. Erst danach kann dann die Auswahl für eine bestimmte Komponente getroffen werden. Bei einem solchen Auswahlprozess sind neben der reinen Funktionalität auch rechtliche Fragen zu stellen, wie etwa diejenige nach dem Lizenztyp und dessen „Verträglichkeit“ mit dem eigenen Produkt. Um diese Auswahl zuverlässig und nachhaltig treffen zu können, bedarf es an Erfahrung im technischen Team und eben auch eines fundierten Prozess-Rahmenwerks, in welchem Governance- und Entwicklungsteam zusammenarbeiten können.

Bei IBM existiert ein interner Software-Katalog (Asset Library), der genaue Klassifikationen der verschiedenen Open-Source-Komponenten je nach Lizenz-Typ enthält. Vereinfacht gesagt, wird anhand verschiedener Kriterien zwischen „bedenkenlos einsetzbar“, „eher bedenklich“ und „nicht zulässig“ unterschieden. Gegebenenfalls zieht dann der Einsatz der jeweiligen Komponente weitere Genehmigungsschritte nach sich. Komponenten, die in diesem Katalog nicht gelistet sind, bedürfen einer besonderen Behandlung und folglich auch einer entsprechenden Vorlaufzeit.

Der Auswahlprozess für Open-Source-Softwarekomponenten kann sehr schnell sehr kompliziert und unübersichtlich werden. Im Alltagsgeschäft fällt es schwer, den Überblick darüber zu behalten, an welcher Stelle des Produkts von wem welche

IBM Rational Asset Manager

IBM Rational Asset Manager (RAM) ist ein Software-Library-Management-System zur Katalogisierung von Assets jeglicher Art, die im Zuge der Wertschöpfungskette eines Produkts entstehen bzw. benötigt werden. Durch einen prozessgetriebenen Ansatz wird dabei über entsprechende Approval-Ketten die „Compliance“ des jeweiligen Assets sichergestellt. Eine OSLC-Fähigkeit ermöglicht es letztlich, Assets mit Artefakten in Rational Team Concert, wie beispielsweise Tasks, zu verknüpfen. Zudem können direkt aus einem Jazz Build-Engine-Prozess heraus Artefakte in einem RAM Asset Repository publiziert werden. Die genannten Beispiele basieren unmittelbar auf im RAM enthaltene Beispielprojekte.

Komponente integriert wurden und ob die obligatorische Genehmigung dafür erteilt wurde. Im schlechtesten Fall landet eine Open-Source-Softwarekomponente dadurch unbemerkt und versehentlich im Endprodukt und wird somit im Lizenztext des Produkts nicht explizit ausgewiesen. Dies ist in fast allen Fällen ein Verstoß gegen die Lizenzbedingungen.

Unser Ansatz zur durchgängigen Prozessunterstützung zwischen Entwicklungs- und Governance-Prozess mittels integrierter Entwicklungswerkzeuge haben wir mit dem IBM Rational Asset Manager [RAM] umgesetzt. Dieser wurde in die Werkzeugkette des Entwicklungsprozesses integriert (Abbildung 5).

In einem ersten Schritt wurde die Asset-Struktur (siehe Abb. 1) als Repository-Library modelliert, um darin entsprechende Open-Source-Assets zu verwalten. Die Lebenszyklen der jeweiligen Asset-Typen wurden entsprechend modelliert, um den Freigabeprozess für ein bestimmtes Asset abzubilden.

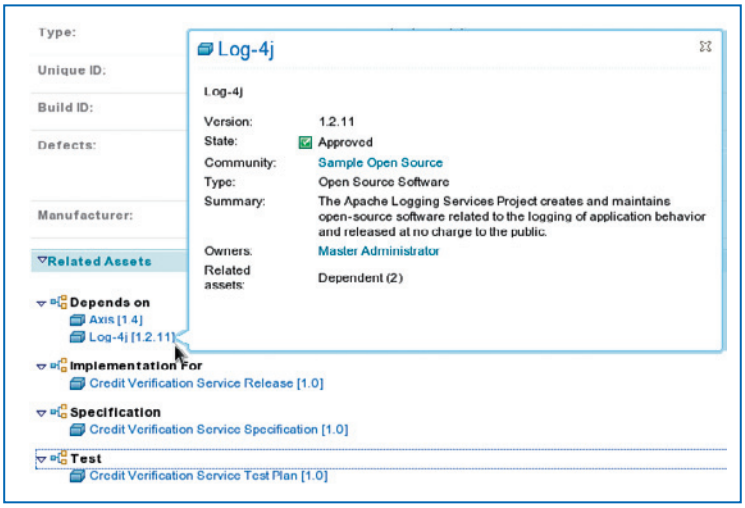


Abb. 4 Ein Implementation-Build mit Auflistung aller inkludierten Abhängigkeiten

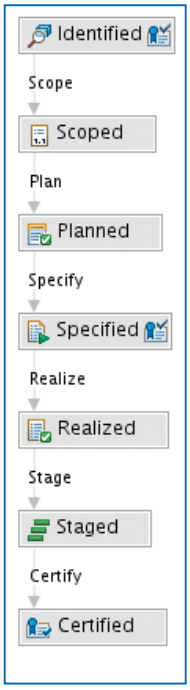


Abb. 2 Lifecycle Diagramm des Credit Verification Service Release Assets

Abbildung 2 veranschaulicht den Lebenszyklus für ein „Release"-Asset, welches eine abgeschlossene und „konsumierbare" Softwarekomponente inklusive ihrer Abhängigkeiten repräsentiert.

Das Team pflegt nun mit Hilfe des Open-Source-Compiance-Experten kontinuierlich und iterativ während des Projekts sämtliche nicht bereits genehmigten Open-Source-Artefakte in das Asset Repository ein (Zustand: Submitted). Hierbei sind gegebenenfalls sogar unterschiedliche Versionen derselben Komponente zu verwalten.

Das Projektmanagement hat wiederum die Aufgabe, die entsprechenden Assets in direkter Zusammenarbeit mit der zuständigen Rechtsabteilung genehmigen zu lassen (Zustand „Approved"). Hierbei ist der Informationsaustausch über die RAM-Plattform möglich, etwa um Unklarheiten zeitnah beseitigen und den Genehmigungsprozess revisionssicher dokumentieren zu können.

Eine Herausforderung ist es, die Entwicklungsteams mit dem Prozess und dem Repository vertraut zu machen und die Abhängigkeiten zu Open-Source-Artefakten zukünftig nicht mehr selbst zu „organisieren". Statt wie bisher über das Versionskontrollsystem müssen die Komponenten nun über das Asset-Repository verwaltet und bezogen werden.

So wird der Einsatz von Open-Source-Komponenten sehr früh transparent gemacht. Wichtig ist hierbei auch die direkte Integration des Rational Asset Managers in die IDE der Entwickler (Abbildung 3).

Produktinkremente („Builds") können über eine entsprechende Schnittstelle im Repository publiziert und verwaltet werden, was die vollständigen Komponentenabhängigkeiten transparent macht (Abbildung 4). Dies ermöglicht eine Impact-Anlayse für den Fall, dass eine bereits integrierte Komponente nachträglich zu aktualisieren oder zu entfernen ist.

Zudem werden z. B. auch Verweise auf zugeordnete Testpläne und Implementierungsspezifikationen (im Beispiel ein WSDL-Dokument) verwaltet.

Ausblick

Aktuell arbeitet unser Team an einer weiteren Integration der eingesetzten Konfigurations-, Anforderungs- und Testmanagementwerkzeuge, um die Rückverfolgbarkeit über alle Disziplinen automatisch sicherzustellen (siehe Abbildung 5). Damit dies insgesamt als integrierter, agiler Prozess möglich ist, ist eine durchgängige Prozessunterstützung über Werkzeuggrenzen hinweg absolut notwendig. Für die Integration der unterschiedlichen Werkzeuge existiert mit OSLC (siehe Kasten) bereits eine entsprechende technische Grundlage.

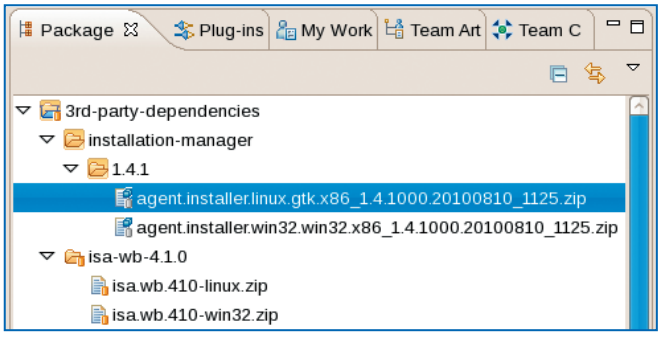


Abb. 3 Abruf von Assets im Rational Asset Manager durch eine entsprechende IDE-Unterstützung

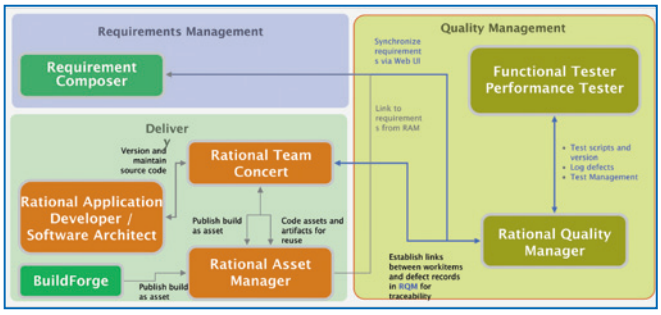


Abb. 5 Vollständig intergrierte Werkzeugkette (Zieldefinition)

Open Services for Lifecycle Collaboration

„Open Services for Lifecycle Collaboration“ (OSLC) bezeichnet eine 2008 von IBM Rational ins Leben gerufene Industrieinitative, Ziel der Initiative ist es, herstellerübergreifend offene Standards für die Dienste, Schnittstellen und Datenformate von Entwicklungswerkzeugen zu definieren. Die Arbeit ist in unterschiedliche Domänen unterteilt (Change Management, Requirements Management etc.) und wird derzeit aktiv von einer Community von über 25 Unternehmen, Partnern und Open-Source-Projekten entwickelt.

OSLC will die Integrations- und Wartungskosten zwischen unterschiedlichen Werkzeugen dramatisch zu reduzieren. Weitere Infos unter http://open-services.net

Fazit

Es hat sich als empfehlenswert herausgestellt, Compliance-Konformität in der Softwareentwicklung mittels team-interner Compliance-Experten und mit einer durchgängigen Tool-Unterstützung zu adressieren. Der im Artikel beschriebene Prozess hat sich bei der Produktentwicklung im IBM Entwicklungslabor Böblingen als hilfreich und sinnvoll erwiesen. Der IBM Rational Asset Manager und dessen Integrationsmöglichkeit in die Tool-Landschaft mittels OSLC bietet hier einen effizienten Ansatz.

Literatur & Links

[Amb09] Agile software development and regulatory compliance http://www.ambysoft.com/surveys/practices2009.html#Figure7

[Amb10] 2010 Agile Project Success Rates Survey Results http://www.ambysoft.com/surveys/agileSuccess2010.html

[GNU] GNU general Public License http://en.wikipedia.org/wiki/GNU_General_Public_License

[COPY] Erläuterungen zum Thema Urheberrecht http://de.wikipedia.org/wiki/Copyleft

[PMBOK] Project Management Body of Knowledge http://www.pmi.org/

[RAM] Rational Asset Manager http://jazz.net/projects/rational-asset-manager/

Daniel S. Haischt

Softwareentwickler im IBM Deutschland Research & Development Labor Boeblingen. Setzt sich für agile Softwareentwicklungsmodelle basierend auf der Rational Jazz-Plattform ein. Arbeitet in seiner Freizeit als Committer der Apache Software Foundation an zahlreichen Open Source Software Projekten. (daniel.haischt@de.ibm.com)

Florian Georg

Kundenberatung bei der Umsetzung und Verbesserung Ihrer Softwareentwicklungsprozesse durch die Kombination von Best Practices in den Bereichen Prozesse, Methodik und Werkzeuge. Schwerpunkte sind agiles Lifecycle Management und modellgetriebene Softwareentwicklung. (fgeo@ch.ibm.com)

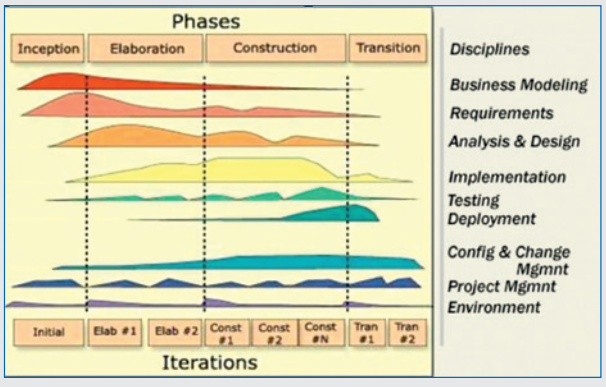
Agile Lizensierung für agile Organisationen

Ein Token repräsentiert eine Werteinheit, die bei einem Lizenzserver wiederholt gegen eine Kombination von Rational Softwarelizenzen eingetauscht werden kann. Auf diese Weise steht den Spezialisten zu jedem Zeitpunkt im Entwicklungszyklus das passende Werkzeug für die benötigte Anzahl von Arbeitsplätzen zur Verfügung. Für jedes Rational Produkt wird für die Dauer der Nutzung eine spezifische Anzahl von Tokens temporär allokiert. Nach Ende der Nutzung, beispielsweise mit Übergang in die nächste Projektphase, werden die Tokens in den „Pool“ zurückgegeben, wo sie für den Bedarf eines anderen Anwenders zur Verfügung stehen – sei es für dieselbe oder auch eine andere Rational Lösung (siehe Abb.)

Traditionell haben Lizenzen und Wartung der Softwarewerkzeuge einen signifikanten Einfluss auf die Flexibilität und laufenden Kosten einer IT-Infrastruktur. Kauf, Management und Nutzung der Softwarelizenzen können teuer und zeitaufwendig sein. Hinzu kommt, dass Lizenzen in der Regel nicht so einfach zwischen Produkten oder Nutzern getauscht werden können. Dies hat zur Konsequenz, dass die Produktivität leidet, weil für gerade im Projekt benötigte Werkzeuge nicht genug Lizenzen da sind, während gleichzeitig Lizenzen für andere Tools ungenutzt brachliegen.

Eine flexible Lizensierungsstrategie ermöglicht die optimale Nutzung der Rational Lösungen über den ganzen Entwicklungszyklus hinweg. Es ist nicht erforderlich, vor der Investitionsentscheidung exakt festzulegen, wie viele und welche Art von Lizenzen für welches Werkzeug benötigt werden. Mit Hilfe eines so genannten Token-Rechners wird der Bedarf an Tokens für den aktuellen und den erwarteten zukünftigen Bedarf geschätzt. Da man die Tokens jeweils temporär für die Nutzung der gerade benötigten Werkzeuge einsetzen kann, besteht kein Risiko, das in die Lösungen von Rational investierte Kapital nicht verwenden zu können.

Mit dem Token Lizensierungsmodell gewinnen die Anwender ein Maximum an Flexibilität bei der Verwendung der Rational Produkte. Die Investition wird geschützt; das eingesetzte Kapital kann produktiv genutzt werden, anstatt in Form von ‚Shelfware‘ dem Wertschöpfungsprozess entzogen zu werden. IT und Einkauf gehen daher kein Risiko bei der Kaufentscheidung ein und der Einsatz der Lizenzen für Rational Produkte wird optimiert.



Der IBM Rational Ansatz Agility@Scale zur Skalierung agiler Entwicklungsprozesse

Mit dem Erfolg der agilen Softwareentwicklung wurde in den letzten Jahren zunehmend nach Ansätzen gesucht, diese erfolgreichen Prinzipien auch in komplexen IT-Organisationen einsetzen zu können. Im Gegensatz zu kleineren Softwareherstellern oder „isolierten" Softwareprojekten gibt es hier jedoch Rahmenbedingungen – wie etwa weltweit verteilte Teams – die sich nicht vollständig mit den traditionellen agilen Methoden adressieren lassen. Der Artikel gibt einen kurzen Überblick über den Ansatz „Agility@Scale", der von IBM Rational entwickelt wurde und weltweit bei Beratungsprojekten zur Einführung agiler Prozesse verwendet wird.

IBM Rational hat den Ansatz „Agility@Scale" auf Basis der langjährigen Beratungserfahrung im Bereich Software-Methodik und -Prozesse entwickelt. Die Implementierung eines skalierbaren, agilen Entwicklungsprozesses ist immer kundenspezifisch und wird durch spezielle Workshops, Assessments, Schulungen und gegebenenfalls Werkzeuglösungen unterstützt. Es ist jedoch wichtig, dass der hier vorgestellte Ansatz grundsätzlich ein methodisches Rahmenwerk bietet und somit unabhängig von Softwarewerkzeugen gleich welcher Hersteller umgesetzt werden kann.

Das IBM Rational Skalierungsmodell für agile Entwicklung ist dreistufig (siehe Abb. 1):

- Agile Core
- Disciplined Agile Delivery
- Agility@Scale

Ausgehend von den grundlegenden agilen Arbeitstechniken (zum Beispiel iterative Entwicklung, Continuous Integration) wird zunächst der Betrachtungswinkel auf den vollständigen Softwarelebenszyklus und die IT-Governance ausgeweitet („Disciplined Agile Delivery"). Im nächsten Schritt erfolgt dann eine Identifizierung der im konkreten Kontext anzutreffenden Skalierungsfaktoren. Diese Faktoren erschweren generell zunächst einmal die Umsetzung agiler Prozesse und müssen daher mit zusätzlichen Arbeitstechniken [PRAC] adressiert werden.

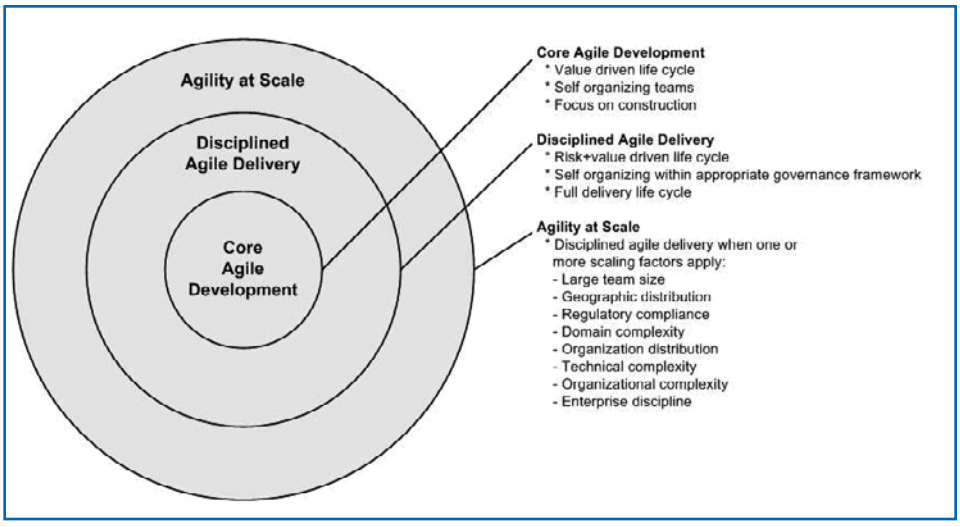


Abb.1 IBM Rational Skalierungsmodell

... breiter

Der erste Schritt in der Skalierung agiler Prozesse wird als „Disciplined Agile Delivery" bezeichnet und umfasst die folgenden Punkte:

- Abdeckung des vollständigen Softwarelebenszyklus
- Risiko- und wertgetriebener Ansatz
- Selbstorganisation innerhalb der vorgegebenen IT-Governance-Strukturen

Die bekannten agilen Ansätze wie Scrum oder XP konzentrieren sich hauptsächlich auf die Softwarekonstruktion. Aus diesem Grund muss zunächst sichergestellt werden, dass der komplette Lebenszyklus durch den agilen Entwicklungsprozess abgedeckt werden kann – von der Projektinitiierung über die Umsetzung bis hin zu Betrieb, Support und irgendwann auch die Systemablösung (Retirement).

Agile Ansätze konzentrieren sich auf die kontinuierliche Demonstration von „nützlichen" Softwareinkrementen („value driven"). Die Entwicklung unter komplexeren Rahmenbedingungen erfordert jedoch zusätzlich einen erhöhten Fokus auf technische und organisatorische Risiken, die frühestmöglich im Lebenszyklus durch entsprechende Techniken adressiert werden müssen.

Zu guter Letzt muss versucht werden agile, selbstorganisierte Teams effizient innerhalb der kompletten IT-Organisation zu verankern. Hierfür muss definiert werden, wie der Geschäftsprozess „Softwareentwicklung" innerhalb der IT-Governance-Richtlinien und -Standards effizient ausgeführt werden kann.

... und höher

Aufbauend auf dem „Disciplined Agile Delivery"-Lebenszyklus gilt es nun über entsprechende Assessments und Business Workshops herauszufinden, welche weiteren Skalierungsfaktoren im konkreten Fall vorliegen.

Die erfolgreiche Einführung von agilen Entwicklungsprozessen bedarf immer der genauen Analyse des Unternehmenskontexts. Der Agility@Scale-Ansatz definiert acht Skalierungsfaktoren, die in unterschiedlich starker Ausprägung auftreten können. Das Vorhandensein jedes dieser Faktoren definiert, welche zusätzlichen unterstützenden Arbeitstechniken empfehlenswert sind, um einen agilen Prozess erfolgreich umsetzen zu können.

Im Einzelnen können die folgenden Faktoren in IT-Organisationen beobachtet werden:

- **Teamgröße:** Von „unter zehn“ bis hin zu tausenden Entwicklern
- **Geographische Verteilung:** „Gemeinsames Büro“ bis hin zu weltweiter Verteilung (Off-Shoring)
- **Regulatorische Vorgaben:** Risikolos bis hin zu strenger Auditierung
- **Fachliche Komplexität:** Einfach bzw. gut verstanden bis hin zu fachlich komplex bzw. Innovationsprojekten
- **Organisatorische Verteilung:** Übergreifende Zusammenarbeit oder starke Vertragsgebundenheit (Contracting, Outsourcing)
- **Technische Komplexität:** Homogene, bekannte Technologie oder inhomogene, Legacy bzw. Emerging
- **Unternehmensorientierung:** Reine Projektorientierung oder Entwicklung innerhalb globaler Unternehmensarchitektur, Projekt-Portfoliomanagement und strategische Wiederverwendung.

Nach der Identifikation und Gewichtung der Skalierungsfaktoren gilt es, den Bedarf an entsprechenden Practices (siehe Kasten) zu analysieren. Je nach identifizierten Stärken und Schwächen werden nun sukzessive die entsprechenden Arbeitstechniken in den Entwicklungsprozess eingeführt. Beispielsweise kann der Skalierungsfaktor „Organisatorische Verteilung“ durch die Practice „Formal Change Management“ adressiert werden.

Fazit

Der aus der langjährigen Erfahrung von IBM Rational im Bereich Prozess- und Methodikberatung entwickelte Ansatz „Agility@Scale“ bietet ein strukturiertes Rahmenwerk zur Definition und Umsetzung agiler Entwicklungsprozesse in komplexen IT-Organisationen.

Ausgehend von der Unterstützung des kompletten Softwarelebenszyklus und der IT-Governance werden zunächst die spezifischen Skalierungsfaktoren identifiziert. Diese können dann mittels Fokusierung auf dezidierte Arbeitstechniken (Practices) unterstützt werden. Auf diese Weise kann ein individueller, agiler Entwicklungsprozess definiert und schrittweise eingeführt werden.

Literatur & Links

[PRAC] IBM Rational Practice Library
http://www-01.ibm.com/software/awdtools/rmc/library/#Practices

Florian Georg

Kundenberatung bei der Umsetzung und Verbesserung Ihrer Softwareentwicklungsprozesse durch die Kombination von Best Practices in den Bereichen Prozesse, Methodik und Werkzeuge. Schwerpunkte sind agiles Lifecycle Management und modellgetriebene Softwareentwicklung.
(fgeo@ch.ibm.com)

Abb.2 IBM Practices

Practices

Eine Practice (auf deutsch in etwa „Arbeitstechnik“) stellt einen Ansatz zur Lösung wiederkehrender Herausforderungen innerhalb eines Entwicklungsprozesses dar.

Im Gegensatz zu traditionellen Prozessrahmenwerken sind Practices in sich abgeschlossene „Methoden-Bausteine“. Diese Bausteine ermöglichen es flexibel und inkrementell eine spezifische Gesamtmethodik zu entwicken und im Unternehmen schrittweise zu institutionalisieren. Die IBM Rational Practice Library unterscheidet zwischen Business-, Management- und Technical-Practices. Teil jeder Practice ist eine Beschreibung dazu, wie der Implementierungsgrad in einem Unternehmen gemessen werden kann, und wie sich dies auf die adressierten Geschäftsprobleme auswirkt.

Agile Modeling

Agil entwickeln und dabei „trotzdem" modellieren? Geht das? Agilität und Modellierung werden auf den ersten Blick oft als Gegensätze begriffen. Dennoch gibt es Ansätze zur Modellierung in agilen Projekten und die Praxis zeigt: Agilisten modellieren! Dieser Artikel stellt die Grundkonzepte vor und geht insbesondere auf den Ansatz des „Agile Modeling" von Scott Ambler ein.

Ideen der agilen Softwareentwicklung

Es gibt wohl kaum eine Diskussion über agile Softwareentwicklung, ohne dass die Teilnehmer nicht mindestens einmal auf das Agile Manifesto [AGIL] und die dort beschriebenen vier Grundwerte verweisen. Aus diesen wiederum leiten sich agile Prinzipien ab, auf denen letztendlich agile Methoden bzw. Practices basieren.

Insgesamt kann man die Agile-Idee als Gegenbewegung zu den schwergewichtigen, klassischen Prozessen ansehen, die aufgrund ihrer festgelegten Phasen und Abläufe sowie ihrer Fokussierung auf umfangreiche Dokumente als starr und unflexibel wahrgenommen werden. Regalfüllende Wälzer an Anforderungs- oder Design-Spezifikationen, die dem Entwickler gleichsam „über den Zaun" zugeworfen werden, gehören genauso zum Schreckensbild des Agilisten, wie starre Werkzeuge, die eher reglementieren als helfen. Stattdessen ist das Ziel der agilen Softwareentwicklung, möglichst schnell zu ausführbarem Code zu gelangen, den die Entwickler dann in kurzen Iterationen überarbeiten bzw. erweitern können. Dabei spielt die direkte Interaktion der Entwickler untereinander und mit dem Kunden eine entscheidende Rolle. Sie ersetzt eine langwierige initiale Entwurfsphase.

Klassische Software-Modellierungskonzepte

Was bedeutet es eigentlich Software zu modellieren?

In erster Annäherung lässt sich antworten: Abstrahieren! Abstraktion aber bedeutet Vereinfachung und Vereinfachung bedeutet Verstehen. Ein Softwaresystem, das zu komplex ist, um es in seiner Gesamtheit im Detail zu verstehen, wird durch die Abstraktion vereinfacht und damit erst verständlich. Wenn dieses Verständnis dann auch noch auf Projektbeteiligte übertragen werden kann, erhalten wir den zweiten Nutzen: die Kommunikation wird einfacher.

Grady Booch, IBM Fellow und einer der Väter der Unified Modeling Language (UML), drückt dies in vier Zielen der Software-Modellierung aus (vgl. [BOO99]):

- *Visualisieren*, um zu beschreiben, was gemeint ist – also Verständnis schaffen.
- *Spezifizieren* als Vorgabe von Struktur und Verhalten eines Systems, das erstellt werden soll.
- *Konstruieren* des Codes auf Basis der Vorgabe in Modellform.
- *Dokumentieren*, um Designentscheidungen zu erklären und festzuhalten. So kann man später darauf aufbauen.

Aus der Erkenntnis, dass Menschen Systeme auf abstrakter Ebene besser behandeln können, folgt natürlich der Wunsch, Vorgaben für ein zu erstellendes System generell auf abstrakter, möglichst sogar auf rein fachlicher Ebene zu erstellen und dann die Information top-down und wenigstens teilautomatisiert in eine Implementierung zu transformieren. Der Oberbegriff Model Driven Design (MDD) ist für solche Ansätze etabliert.

Aktuelle Methoden wie zum Beispiel Model Driven Architecture (MDA) sehen einen mehrstufigen Übergang vom rein fachlichen Modell zur technischen Realisierung vor. Dabei sollen zwar auf jeder Abstraktionsebene manuell Informationen eingeführt werden, die Transformationen von einer abstrakten auf eine detailliertere Ebene sollen aber möglichst automatisiert vonstatten gehen. Automation ist ein wesentlicher Bestandteil des Konzepts, von ihr verspricht man sich eine Steigerung der Effizienz und Produktivität.

Agil und Modellierung – ein Widerspruch?

Passen Agilität und Modellierung zusammen? Bergen die jeweiligen Prinzipien nicht einen Widerspruch? Auf der einen Seite die Ablehnung von überbordender, vorab erzeugter Dokumentation in agilen Prozessen – auf der anderen Seite Modelle, die der Spezifikation und Dokumentation dienen und vor der Codierung erstellt werden. Auf der einen Seite das Streben nach sehr kurzen Entwicklungszyklen und der möglichst schnellen Erzeugung lauffähiger Software – auf der anderen die statisch anmutende Ableitung von Informationen über die Abstraktionsstufen hinweg von Modell zu Modell bis zum Code, wie zum Beispiel bei der Erstellung eines fachlichen Modells, aus dem das systemorientierte Anforderungsmodell abgeleitet wird, aus dem wiederum das Designmodell abgeleitet wird, woraus schließlich der Code abgeleitet wird.

Außerdem: Wie verhält sich die direkte Interaktion bei Bedarf, eine der Grundvoraussetzungen der agilen Entwicklung, zu dem Formalismus, der in der Modellierung als Grundlage für die Automation benötigt wird und diese schnell wie ein kompliziertes, schwergewichtiges und wenig flexibel handhabbares Mittel erscheinen lässt?

Bei genauerem Hinsehen kann man demgegenüber jedoch auch Ähnlichkeiten und Überschneidungen in den Motiven und Zielen entdecken:

- *Kommunikation:* Modellierung soll die Kommunikation im Projekt zwischen allen Beteiligten unterstützen und verbessern. Kommunikation und Interaktion sind aber auch Schlüsselelemente agiler Entwicklung.
- *Produktivitätssteigerung durch Automation:* Diese Kernidee der MDD-Ansätze findet sich auch in vielen agilen Methoden wieder, wie zum Beispiel bei der Continuous Integration und dem Test Driven Development.
- *Vereinfachung:* Modellierung liefert mit der Abstraktion ein Mittel zur Vereinfachung komplexer Systeme. Diese Fokussierung auf das Wesentliche kann das agile Prinzip Simplicity, also das Ziel nur das Wesentliche zu erstellen, ideal unterstützen.
- *Gemeinsames Verständnis:* Modelle als Kommunikationsmittel sollen ein gutes gemeinsames Verständnis innerhalb des Projektteams und mit den „Stakeholdern" erzeugen – so wie es auch das agile Prinzip Whole Team fordert.
- *Gute(s) Software-Design / -Architektur:* Wie in klassischen Prozessen wird auch in den agilen Prinzipien gefordert, das Augenmerk auf ein gutes Design zu legen. Womit könnte man das besser erreichen, als über die Modellierung?

Darüber hinaus lösen sich auch einige der genannten Konflikte auf, wenn man eine geeignete Handhabung annimmt. So schließen die agilen Werte eine Dokumentation keinesfalls aus. Dort, wo es beim Verstehen und Realisieren hilft, wird sie explizit als nützlich betrachtet. Es wird lediglich gefordert, die Dokumentation nicht um ihrer selbst Willen zu erstellen und keine überflüssige Arbeit hineinzustecken. Im Bereich der Modellierung gibt es jedoch kein Gesetz, das vorschreibt, man solle mehr modellieren als für die erfolgreiche Erstellung der Software nötig ist. Der Widerspruch löst sich also bei einer geeigneten Handhabung der Modellierung auf.

Eigentlich hätte dieser Widerspruch nie bestanden, wenn die Praxis klassischer Prozesse heute nicht oftmals leider anders aussehen würde. Der Umfang, die Vorgehensweise, der Grad der Formalität, die Mächtigkeit des eingesetzten Modellierungstools – all diese Parameter können auf die Situation adaptiert werden. Modellierung steht damit nicht grundsätzlich im Konflikt mit agilen Werten und Prinzipien. Betrachtet man ein Whiteboard als eine Art Modellierungstool und bezieht das Skizzieren von Software-Strukturen und -Abläufen am Whiteboard in den Oberbegriff Modellierung mit ein, so gibt es sogar Hinweise darauf, dass Modellierung im Bereich der agilen Prozesse verbreiteter ist als in traditionellen (nicht-iterativen) Prozessen. Scott Amblers Umfrage [AMBY08] aus dem Jahre 2008 zum Beispiel zeigte dieses Ergebnis.

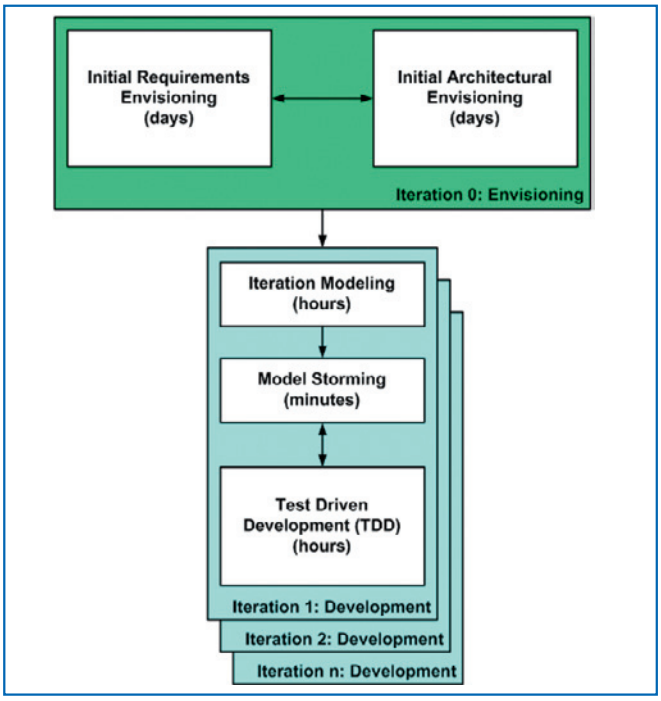


Abb. AMDD Lifecycle

Agile Modeling (AM)

Der Begriff Agile Modeling wurde im Jahre 2002 erstmals von Scott W. Ambler eingeführt (vgl. [AMB02].

Inzwischen ist Agile Modeling als agile Methode etabliert, die auf einer Reihe von Werten, Prinzipien und Practices basiert. Die Werte sind weitgehend von denen des eXtreme Programming (XP) abgeleitet (*Communication, Simplicity, Feedback, Courage*, vgl. [BEC00]) und durch einen weiteren ergänzt: Bescheidenheit (Humility).

Die für Agile Modeling genannten Prinzipien sollen an dieser Stelle nicht ausführlich wiedergegeben werden. Beispiele wie *Einfachheit voraussetzen, Änderungen begrüßen, zu einem bestimmten Zweck modellieren, lauffähige Software ist das eigentliche Ziel* und *mit leichtem Gepäck reisen* [1] machen aber bereits deutlich, worauf diese Prinzipien ausgerichtet sind: Modellierung soll im Kontext der agilen Softwareentwicklung geeignet gehandhabt werden, um die oben genannten Konflikte auszuschließen. Damit stellt sich die agile Modellierung nicht mehr als das auf den ersten Blick wahrgenommene Hindernis zur Agilität dar, sondern als ein Mittel, das agile Prozesse beflügeln kann.

Auf den Prinzipien bauen Best-Practices auf, die sich letztendlich zu einer Methode des Agile Modelings zusammenfügen – eine Methode, die sich durch Effektivität und Leichtgewichtigkeit auszeichnet. Eine Ausprägung der Methode ist das Agile Model Driven Design (AMDD), das sich als agile Antwort auf das klassische MDD versteht.

Im Unterschied zu klassischen MDD-Ansätzen (in wasserfallartigen Prozessen) verbieten es die Agile Modeling-Prinzipien jedoch, ausführliche Modelle mit Vollständigkeitsanspruch up-front zu erzeugen, denen dann eine Codierungsphase nachfolgt. Stattdessen werden u.a. die folgenden Best-Practices angewandt, um den AMDD-Lifecycle aufzubauen (siehe Abbildung).

- *Just Barely Good Enough (JBGE) artifacts:* Ein Modell soll nur gerade soviel Information enthalten, wie in der aktuellen Situation gebraucht wird. Überflüssige Arbeit für zukünftigen Bedarf, der dann vielleicht gar nicht entsteht, wird so vermieden.
- *Iteration Modeling:* Am Beginn jeder Iteration kann eine Modellierung vorgenommen werden, die sich auf die Analyse derjenigen Anforderungen bezieht, die in den Iterations-Backlog übernommen wurden. Diese Analyse soll die Abschätzung der jeweiligen Aufwände und damit der Iterationsplanung dienen.
- *Model Storming:* Mit dem Ziel, ein gemeinsames, weiterführendes Verständnis zu einer Anforderung zu erarbeiten und daraus Design-Entscheidungen abzuleiten, kann das Team ein Model Storming starten. Dies passiert in dem Moment, in dem die Realisierung der Anforderung beginnt, also just in time. Das Model Storming umfasst nur die Skizzierung von Lösungsansätzen und dauert typischerweise nur Minuten.
- *Test Driven Developement:* der aus XP übernommene, evolutionäre Ansatz, in dem zunächst ein Test für eine Funktionalität erstellt wird und danach gerade soviel produktiver Code erzeugt wird, um den Test zu bestehen. Die Tests – egal, ob es sich um Beschreibungen oder Code handelt – werden hierbei als eine Art „Anforderungsmodell" begriffen, das als Konstruktionsvorlage dient.

Die bisher genannten Practices beziehen sich alle auf eine Modellierung, die innerhalb jeder Iteration stattfindet. Damit folgt man der Idee, Modelle erst in dem Moment zu erzeugen, in dem diese Information für die Realisierung einer Anforderung gebraucht wird. In Systemen mit komplexen Anforderungen kann es aber darüber hinaus sinnvoll sein, eine initia-

Fußnoten:

[1] Frei übersetzt aus dem Original: Assume Simplicity, Embrace Change, Model With a Purpose, Working Software Is Your Primary Goal, Travel Light

le Modellierung vorzunehmen, die den eigentlichen produktiven Iterationen vorgelagert ist. Dies passiert, um früh die richtigen Prioritäten erkennen oder die richtigen architekturellen Weichen stellen zu können. Die Methode folgt also auch hier immer noch dem Prinzip, Informationen erst in dem Moment zu erzeugen, in dem sie gebraucht werden.

Zwei Agile Modeling Practices beschreiben dieses Vorgehen:

- Requirements Envisioning – wird zu Beginn des Projekts eingesetzt, um eine erste High-Level-Sicht auf die Anforderungen zu bekommen und den Projektumfang abzugrenzen.
- Architecture Envisioning – wird ebenfalls zu Beginn des Projekts durchgeführt, um eine High-Level-Architektur zu beschreiben. Damit wird die übergreifende technische Strategie dokumentiert.

Einsatz des Agile Modeling

Wie die klassische Modellierung auch, gewinnt Agile Modeling mit der Komplexität der Aufgabe oder der Organisation an Bedeutung. Insbesondere in einem Umfeld, in dem Skalierungsfaktoren wie Team- und Organisationsgröße, standortübergreifende Entwicklung, technische Komplexität und Compliance-Anforderungen eine Rolle spielen, können die gestiegenen Anforderungen an Kommunikation und Dokumentation gut durch Agile Modeling adressiert werden. So findet sich die AMDD-Methode auch in IBMs Disciplined Agile Delivery (DAD) Lifecycle innerhalb des Agile Scaling Models (vgl. [AMB09]) wieder, das beschreibt, wie agile Softwareentwicklung auch in komplexen Umgebungen ermöglicht wird.

Je nach Umfeld und Skalierungsfaktoren wird natürlich auch der Grad der benötigten Formalität bei der Modellierung variieren. Agile Modeling als eine Methode, die sowohl in kleinen Teams mit räumlicher Nähe funktionieren soll, wie auch in der großen, standortübergreifenden Arbeit auf Enterprise-Level, liefert deshalb keine Festlegungen auf Formalitätsgrade, Medien oder Werkzeuge. Es werden lediglich Vorschläge zur praktischen Handhabung der Modellierung gemacht, die sich innerhalb der Bandbreite zwischen einer informellen Skizzierung am Whiteboard bis zur formalen Beschreibung in UML2 inklusive nachgeschalteter Code-Generierung bewegt (vgl. [AGM-a] und [AGM-b]).

Literatur & Links

[AGIL] www.agilemanifesto.org

[BOO99] Booch, Rumbaugh, Jacobson, The Unified Modeling Language User Guide, Addison Wesley, 1999

[AMBY08] Scott W. Ambler, Modeling and Documentation Practices on IT Projects Survey Results: July 2008, www.ambysoft.com/surveys/modelingDocumentation2008.html

[AMB02] Scott W. Ambler, Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, Wiley, 2002

[BEC00] Kent Beck, Extreme Programming. Das Manifest, Addison-Wesley, 2000

[AMB09] Scott W. Ambler, The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments, IBM Rational Whitepaper, 2009, ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/raw14204usen/RAW14204USEN.PDF

[AGM-a] www.agilemodeling.com/essays/amddApproaches.htm

[AGM-b] www.agilemodeling.com/essays/simpleTools.htm

Andreas Entgelmeier

IT-Spezialist im Bereich Technical Sales der IBM Rational Software Brand. Zuständig für Beratung, Training und Coaching zu Methoden und Tools innerhalb des Application Lifecycle Managements, Requirements Managements und objektorientierten Analyse & Designs. 18-jährige Berufserfahrung mit Stationen in den verschiedenen Disziplinen der Softwareentwicklung, vom Design über die Implementierung bis zum Projektmanagement
(entgelme@de.ibm.com)

Softwareentwicklung in den Wolken

Die Bedeutung des Cloud Computing in den Unternehmen

Mehr als 35% der CIOs sehen Cloud Computing längst nicht mehr nur als Vision, sondern als eine ihrer wichtigsten Initiativen in Bezug auf die IT-Strategie ihres Unternehmens. Was vor Jahren mit der Konsolidierung und Virtualisierung einzelner Systeme begann, entwickelt sich heute zu anwenderorientierten und höchst effizienten Bereitstellungsmodellen für IT und Informationsservices. Diese zeichnen sich durch ein hohes Maß an Skalierbarkeit und Bedienungskomfort aus und lassen sich an den Anforderungen der Nutzer ausrichten.

Damit eröffnet Cloud Computing als strategisches IT-Konzept neue Möglichkeiten. Flexibel wie nie zuvor kann entschieden werden, welche IT-Dienste wie bereitgestellt bzw. aus Benutzersicht bezogen werden sollen. Doch was macht diese Flexibilität aus?

Cloud Computing – Revolution oder Evolution?

Bei Cloud Computing geht es vor allem darum, wie IT (z. B. Infrastruktur, Plattformen oder Anwendungen) als Service aus einer virtualisierten Umgebung bereitgestellt werden kann. Dabei ist es weniger die Virtualisierung oder die Bereitstellung der Services, was Cloud Computing ausmacht. Es ist vielmehr der hohe Grad der Standardisierung und Automatisierung, der die Selbstbedienung durch Endbenutzer evolutionär perfektioniert. Gerade deshalb redet man im Zusammenhang mit Cloud Computing über eine neue Art von Nutzungsmodell, das sich nicht nur durch Qualität und Geschwindigkeit, sondern vor allem auch durch eine hohe Agilität auszeichnet.

Die Cloud im Software Entwicklungs- und Testumfeld

Warum aber ist die Softwareentwicklung eine der ersten und häufigsten Anwendungsgebiete für eine Cloud Computing Infrastruktur?

- Effiziente Nutzung der Ressourcen:
  Wenn man weiss, dass heute nicht selten gut 50% der IT Infrastruktur für Softwareentwicklungs- und Testzwecke reserviert ist, diese aber gleichzeitig oft nur zu etwa 10% ausgelastet ist, liegt die Motivation für Cloud Computing klar auf der Hand.

- Standardisierung der Entwicklungsumgebung und der Methodik
  Einen weiteren positiven Nebeneffekt bietet die Standardisierung der Services und damit der Entwicklungs- und TestuUmgebungen. Während heute etwa 30% der Fehler passieren, weil manuell aufgesetzte Entwicklungs- und Testsysteme fehlerhaft oder unterschiedlich konfiguriert sind, sorgt die Standardisierung in der Cloud für eine entsprechende Qualitätssteigerung. Damit werden viele Probleme beim Ausrollen neuer Werkzeuge oder neuer Werkzeugversionen, die mit unvorhersagbaren Konfigurationen der unkontrolliert geänderten Entwicklungsumgebungen zu tun haben, minimiert. Ebenso kann man mit der Standardkonfiguration der Werkzeuge auch eine bessere Standardisierung der Entwicklungsmethodik erreichen. Diesen Teil könnte man auch als entscheidenden Schritt in Richtung Industrialisierung der IT ansehen, vergleichbar mit der Einführung von Lean Management und Six Sigma Prozeduren in der Fertigungsindustrie.

- Erheblich schnellere Bereitstellung der Entwicklungsumgebung:
  Beim Start neuer Projektteams ist bei herkömmlicher Vorgehensweise die Hardware und Softwarelizenzen für das Projekt zu beantragen. Die Hardware und Software müssen installiert und die Softwarewerkzeuge müssen entsprechend der Nutzung konfiguriert werden. Wenn die Vorgehensweise mit dem Werkzeug noch nicht gesetzt ist, wird selbst diese erst entwickelt und beschrieben. Es liegt auf der Hand, dass die Bereitstellungszeiten hier mehrere Wochen dauern können, die das Projekt verzögern, spätestens dann, wenn es ungeplante Änderungen im Projektplan gibt. Oft verbunden mit internen Debatten über Beschaffung von Hardware und Software. Mit einer Cloud Infrastruktur reduziert sich dieser Vorgang häufig auf wenige Minuten: Aussuchen des richtigen Cloud Images, Bereitstellen des Images, fertig ist die Entwicklungsumgebung. Wenn die Cloud Ressourcen als „Software as a Service“ bereitgestellt werden, kann das Projekt die Kosten eventuell selbst tragen und entscheiden, ohne langwierige Verhandlungen mit anderen Abteilungen.

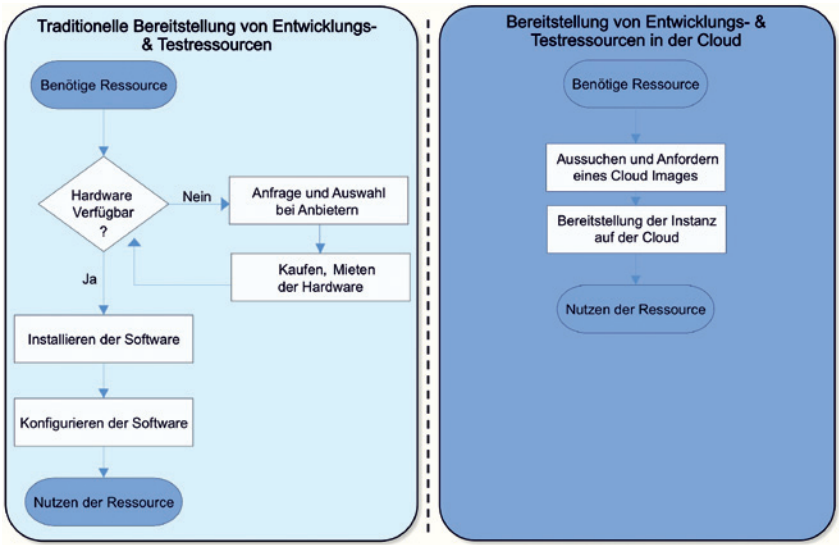


Abb. 1: Bereitstellung

Beim Test kommen die Vorteile vom Cloud Computing besonders zum Tragen:

- Gerade beim Test unterliegt die Nutzung der Hardware- und Softwareressourcen extremen Spitzen, während diese Infrastruktur ansonsten kaum genutzt wird. Beim Cloud Computing können die Ressourcen nach dem Test freigegeben und damit für andere Zwecke genutzt werden. Bei „Software as a Service“ belasten sie damit nicht mehr das Budget.

- Häufig ist eine Applikation für viele verschiedene Kombinationen von Betriebssystemen, Datenbanken, Applikationsserver usw. geschrieben. Das ergibt eine große Anzahl von Zielplattformen, die getestet werden muss. Ohne die Virtualisierung der Cloud würde das auch eine entsprechende Anzahl physikalischer Testumgebungen bedeuten. Mit der Cloud braucht man nur entsprechend viele Images zu konfigurieren, die ganz nach Bedarf genutzt werden können. Das ist eine große Kostenersparnis und sichert die Verfügbarkeit der Testumgebungen.

Bei den offensichtlichen Vorteilen haben viele große Unternehmen bereits einzelne Aspekte einer Cloud Infrastruktur, wie die Virtualisierung, realisiert. Cloud Computing mit seinen Möglichkeiten der automatisierten Bereitstellung und Verwaltung der virtuellen Umgebungen ist ein entscheidender Evolutionsschritt dieser Ansätze.

Über Public Cloud Ansätze ergeben sich aber auch ganz neuen Möglichkeiten für Unternehmen jeder Größe, die damit einen kostengünstigen und einfachen Zugang zu Entwicklungsumgebungen bekommen, die für sie ohne Cloud unwirtschaftlich wären.

Unterschiede in den Cloud-Angeboten

Während immer mehr Anbieter den Markt mit Cloud-Angeboten bedienen, wächst die Unsicherheit darüber, was die jeweils richtige Cloud-Strategie ist.

Zunächst müssen unbedingt Cloud-Angebote, die für den privaten Nutzer geschaffen sind, von Angeboten unterschieden werden, die für Unternehmensanforderungen geeignet sind. Insbesondere beim Thema Sicherheit, aber auch bei den zugesicherten Services haben Unternehmen viel höhere Anforderungen. Datensicherheit, Datenschutz, Ausfallsicherheit usw. spielen hier eine entscheidende Rolle.

Die IBM hat ihre Cloud-Lösungen ganz an diesen Ansprüchen ausgerichtet und bietet die ganze Breite an denkbaren Cloud-Strategien an:

Von fertigen virtuellen Umgebungen, die auf einer öffentlich von IBM betriebenen Infrastruktur laufen (d.h. Public Cloud), bis hin zum Aufbau einer Cloud Infrastruktur beim Kunden selbst nach seinen individuellen Wünschen mit individuellen Cloud Images.

Was bedeutet das für die Softwareentwicklung?

Wie jede Änderung, wird es für manchen Beteiligten in der Softwareentwicklung schwierig sein, sich mit diesem neuen Modell anzufreunden. So wird dem Entwicklungsprojekt eine fertige Entwicklungsumgebung vorgegeben, wo heute diese Umgebung noch häufig vom Projekt aufgebaut wird.

Andererseits bedeutet das aber, dass sich das Entwicklungsprojekt mehr mit seiner eigentlichen Aufgabe beschäftigen kann: Die Applikation in der geplanten Zeit, mit dem geplanten Budget und zur Zufriedenheit des Auftraggebers zu entwickeln und auszuliefern.

Michael Schüttler

Geschäftsverantwortung für das Portfolio, dessen Vertrieb und die Projektumsetzung der Service Product Line für Middleware Services. Außerdem der zentrale Ansprechpartner für Cloud Computing im IBM Infrastruktur Service Bereich.
(schuettl@de.ibm.com)

Andreas Essigkrug

Teamleader IBM Rational Service für Prozess-, Produkt-, Projekt-, Portfoliomanagement und Enterprise Modernization. Koautor von „Rational Unified Process kompakt". Aufgabenschwerpunkte in den Bereichen Softwareentwicklungsprozesse und Werkzeugunterstützung für den gesamten Softwareentwicklungszyklus.
(andreas.essigkrug@de.ibm.com)

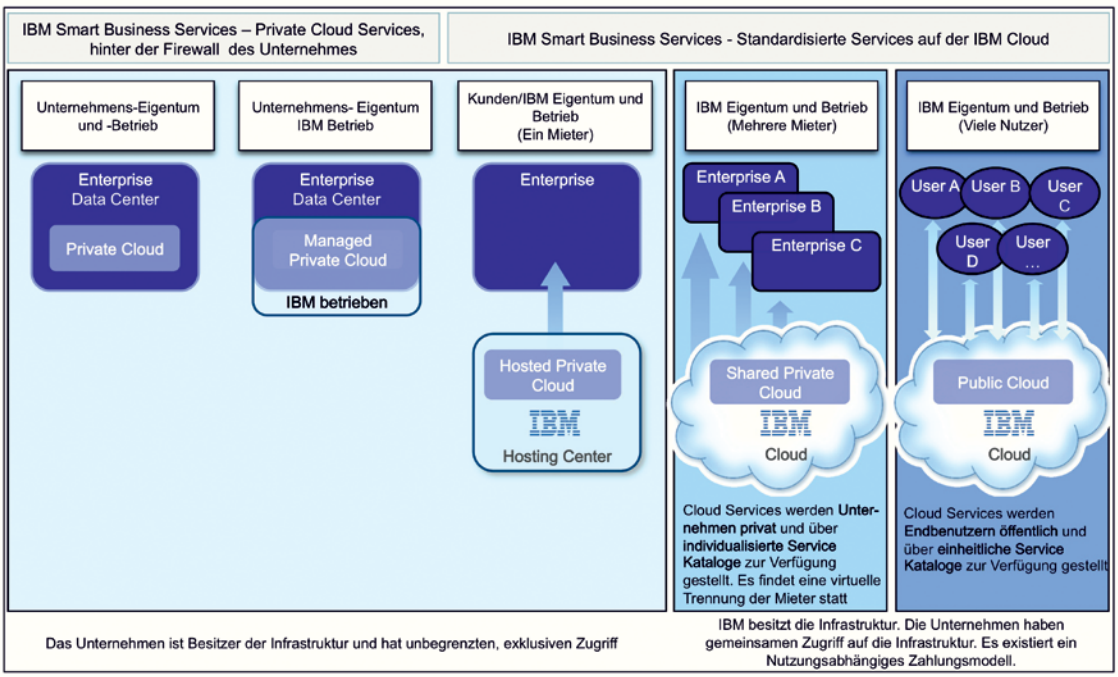


Abb. 2 IBM Cloud Modelle

Agile Methoden und Verfahren bei der Entwicklung integrierter Systeme

Wenn es darum geht, was agiles Software-Engineering für die Entwicklung von Embedded Systemen bedeuten könnte, sind bei unzulänglich informierten Technologieexperten zwei Meinungen vorherrschend. Sie denken entweder:

- „Mit Agilität können wir all unsere Probleme lösen."

oder:

- „Nur ein absoluter Anfänger würde bei der Entwicklung von Embedded Systemen agile Methoden einsetzen."

Beiden Meinungen ist eines gemeinsam: Sie zeugen von mangelhafter Kenntnis dessen, was Agilität tatsächlich bedeutet.

Nicht viel anders jedoch verhält es sich mit dem Begriff „Embedded Systeme". Auch hier macht sich bisweilen große Unklarheit breit: Was verbirgt sich denn eigentlich dahinter? Die daraus resultierenden Fehlinterpretationen und unterschiedlichen Meinungen führen zu falschen Schlussfolgerungen hinsichtlich des möglichen Nutzens, den die agile Entwicklung integrierter Systeme mit sich bringt. Fangen wir also von vorn an: Was sind Embedded Systeme und was bedeutet „agil"? Wenn es uns gelingt, uns auf eine Definition dieser Begriffe zu einigen, werden wir die potenziellen Vorteile, die sich durch die agile Entwicklung integrierter Systeme ergeben, besser nachvollziehen können.

Embedded Systeme – Definition

Die präzise Begriffsbestimmung von Barr und Massa ist ein hilfreicher Ausgangspunkt: „Ein Embedded System ist eine Kombination aus Computerhardware und -software – sowie bei Bedarf weiterer mechanischer oder elektronischer Komponenten – zur Ausführung einer dedizierten Funktion."[1] Wir werden später auf diese Aussage zurückkommen. Doch in der Zwischenzeit dient sie uns als Arbeitsdefinition eines integrierten Systems.

Agilität – Definition

Durch eine nähere Abgrenzung dieses Begriffs können wir besser erkennen, welche mögliche Funktion Agilität bei der Entwicklung von Software für Embedded Systeme übernimmt.[2] Zunächst ist es wichtig zu erkennen, dass Agilität kein einzelnes, einfaches Konzept darstellt:

- Erstens gibt es Methoden zur agilen Softwareentwicklung (Scrum, XP, DSDM, RUP, OpenUP usw.)
- und zweitens gibt es agile Verfahren zur Umsetzung von Softwareprojekten (zum Beispiel fortlaufende Integration, Planning Poker, Retrospektiven usw.).

IBM intern verwenden wir bisweilen eine Variante dieser Definition, anhand derer wir unseren Teams erklären, was es bedeutet, eine agile Herangehensweise in einem konkreten Softwareentwicklungsprojekt zu nutzen: "Agilität bedeutet, basierend auf dem kontinuierlichen Feedback von Stakeholdern und auf der Grundlage von Anwendungsfällen in einer Serie von kurzen Iterationen hochwertigen sofort einsetzbaren Code zu generieren."

Mit anderen Worten: Statt bestimmte Methoden oder Verfahren anzuwenden, sollen sich unsere Teams auf Ergebnisse konzentrieren. Dabei verlassen wir uns auf die Fähigkeit der Teams, eigenständig die für das jeweilige Projekt am besten geeigneten Maßnahmen und Vorgehensweisen auszuwählen. Wir unterstützen die Verwendung unterschiedlicher agiler Methoden und Verfahren für eine strukturierte Softwareentwicklung.

Agilität für Embedded Systeme

Angesichts dieser Definition sollten Überlegungen zur Anwendung einer agilen Strategie in erster Linie im Hinblick auf die dem Embedded System zugrunde liegende Software und deren Entwicklung angestellt werden. Es lässt sich ohne Übertreibung sagen, dass bei steigender Komplexität der Software in einem Embedded System die disziplinierte Implementierung einer agilen Software-Engineering-Methode von Vorteil ist.[3] So erfordert beispielsweise ein einfacher Thermostat zur Anzeige – und zur eventuellen Übertragung – von Daten vergleichsweise einfache Software. Ein intelligentes Airbagsystem hingegen, das sich auf das Gewicht und die physischen Eigenschaften eines Insassen einstellt und bei dem es um die Sicherheit von Menschenleben geht, erfordert eine weitaus umfangreichere Programmierung. Bei derartig komplexen Projekten kann sich die jeweils ausgewählte Entwicklungsmethode drastisch auf die Qualität und Effizienz des Endprodukts auswirken – das heißt, es ist umso wichtiger, eine agile Methode und agile Verfahren in Erwägung zu ziehen.

Tatsächlich weist für Embedded Systeme entwickelte Software einige spezielle Merkmale auf. Ganz allgemein muss die Software so spezifisch entwickelt sein, dass sie sich genau in dem Embedded System ausführen lässt, dessen Bestandteil sie werden soll. Da die Computerhardware sowie die einzelnen mechanischen und elektronischen Komponenten möglicherweise noch nicht vollständig zur Verfügung stehen, kann sich die

Fußnoten:

[1] Michael Barr und Anthony Massa, Programming Embedded Systems: with C and GNU Development Tools, 2. Auflage (Peking: O' Reilly, 2006), 1

[2] Es sei noch einmal ausdrücklich betont, dass es sich empfiehlt, zu Beginn einer Diskussion um den Begriff „agil" das Agile Manifest (http://agilemanifesto.org/) sowie die darin enthaltenen zwölf Prinzipien (http://agilemanifesto.org/principles.html) zu erörtern. Im vorliegenden Artikel wird davon ausgegangen, dass der Leser mit den Grundzügen des Manifestes und den darin vermittelten Werten vertraut ist.

[3] Dies mag für manchen schwer verständlich sein. Als die agile Softwareentwicklung noch in ihren Anfängen steckte, herrschte die allgemeine Auffassung, dass sich diese am besten – wenn nicht sogar ausschließlich – auf kleine Projekte anwenden lässt. Im Verlauf unzähliger IBM interner Software-Engineering-Projekte konnten wir feststellen, dass der hohe Grad an Disziplin, den Agilität voraussetzt, viel wichtiger für große als für kleine Projekte ist, die mit so gut wie jeder Methode umgesetzt werden können.

Durchführung gründlicher Tests als schwierig erweisen (durch Modellierung lässt sich dieses Problem jedoch entschärfen). Es sind unter Umständen strengere Testverfahren als für einen Allzweckcomputer notwendig – es ist weitaus komplizierter, Software für den Betrieb einer Kühlanlage als eine Anwendung für einen Laptop zu aktualisieren. Darüber hinaus sind einige Embedded Systeme sicherheitskritisch: Der zuvor erwähnte Airbag, Herzschrittmacher, Röntgenapparate und sonstige medizinische Geräte sind hervorragende Beispiele hierfür.[4]

Ein zentraler Aspekt der oben angeführten Definition für Embedded Systeme lässt sich allerdings nach wie vor nicht eindeutig klären: „Ein Embedded System ist eine Kombination aus Computerhardware und -software – sowie bei Bedarf weiterer mechanischer oder elektronischer Komponenten – zur Ausführung einer dedizierten Funktion." Ein MP3-Player gilt als Embedded System und die „dedizierte Funktion" wäre die Musikwiedergabe, so lässt sich annehmen. Doch was bedeutet es, wenn der iPod nano plötzlich auch das Aufzeichnen von Videos ermöglicht? Handelt es sich immer noch um ein Embedded System, zumal das Gerät nun über mehr als eine dedizierte Funktion verfügt? Oder macht die Videofunktionalität den iPod dadurch unmittelbar zu einem „Allzwecksystem"? Bemerkenswerter ist jedoch die Tatsache, dass PDAs und Mobiltelefone, ursprünglich als Embedded Systeme konzipiert, in zunehmendem Maß tatsächlich viel eher als Universalgeräte aufzufassen sind. Somit steigt die Wahrscheinlichkeit, dass agile Methoden in den kommenden Jahren bei der Entwicklung von Embedded Systemen eine immer wichtigere Rolle spielen werden. Der Grund dafür ist in der Ähnlichkeit dieser Systeme mit Universalrechnern zu sehen, bei denen sich Agilität bereits bewährt hat.

Die beiden wichtigsten Punkte hinsichtlich der Entwicklung von Embedded Systemen unter Anwendung agiler Methoden lassen sich somit folgendermaßen zusammenfassen:

- Agilität kann sich für jegliche Art der Softwareentwicklung als vorteilhaft erweisen.
- Embedded Systeme werden immer komplexer, so dass eine klare Abgrenzung zu Allzweckcomputern nicht mehr eindeutig möglich ist.

Zusätzlich zu dem offenkundigen Nutzenpotenzial agiler Methoden bestehen weitere Vorteile durch Anwendung spezieller agiler Verfahren in einer Entwicklungsumgebung für Embedded Systeme. Folgende grundlegende Fragen verdeutlichen dies:

- Kann ein Team bei der Entwicklung von Embedded Software von Pair Programming oder der testgesteuerten Entwicklung (Test Driven Development, TDD) profitieren? Bei der Entwicklung von Software für Universalrechner ergeben sich durch Pair Programming oder TDD unbestreitbare Vorteile. Es gibt bei der Entwicklung von Embedded Systemen keine speziellen Aspekte, welche diese Vorteile schmälern könnten. Fehler so frühzeitig wie möglich beheben und den optimalen Code schreiben – darum geht es beim Pair Programming und bei der testgesteuerten Entwicklung. Nichts davon steht im Konflikt mit den spezifischen Besonderheiten von Embedded Systemen.
- Kann ein Team bei der Entwicklung von Embedded Software von täglichen Scrums (auch als sogenannte tägliche Stand-up-Meetings bekannt) profitieren? Auch hier ist es ohne weiteres möglich, dieses Verfahren ebenso für Software einzusetzen, die für Embedded Systeme entwickelt wird. Worin bestünde für das Team der Nachteil, vergangene Ereignisse und aktuelle Vorgänge nachvollziehen oder mögliche Schwierigkeiten erkennen zu können? Und würde sich zudem nicht auch die Kommunikation dadurch verbessern, dass Mitglieder des Teams, das sich mit den Nicht-Softwarekomponenten des integrierten Systems befasst, an den täglichen Scrums teilnehmen?
- Kann ein Team bei der Entwicklung von Embedded Software von stabilen Iterationen profitieren? Zweifellos kann es schwierig sein, in den ersten Iterationen die Software so gründlich zu testen, wie es dem Wunsch mancher Beteiligten entspräche – doch ist das fortwährende Sicherstellen eines möglichst stabilen Codes nicht auch ein Vorteil?

Mit diesen Fragen soll lediglich die Tatsache unterstrichen werden, dass der Weg zu den Vorteilen agiler Methoden und Verfahren über eine sorgfältig erarbeitete Strategie führt. Dies trifft gleichermaßen auf die Entwicklung von Software für Embedded Systeme und für Nicht-Embedded Systeme zu. Jedem beliebigen Softwareprojekt kann Agilität zugutekommen. Angesichts der zunehmenden Komplexität von Embedded Systemen und des immer breiteren Leistungs- und Funktionsspektrum der mit Embedded Software ausgestatteten Geräte werden agile Softwaremethoden und -verfahren voraussichtlich immer wichtiger. Die Softwareentwicklung ist keine leichte Aufgabe. Es hat sich gezeigt, dass Teams regelmäßiger Erfolge verzeichnen können, wenn sie auf agile Konzepte zurückgreifen.

Die Bedeutung der Prinzipien schlanker Softwareentwicklung

In den meisten Abhandlungen zum agilen Software-Engineering wird die schlanke Softwareentwicklung ebenfalls thematisiert, die sich zum großen Teil an den drei ausgezeichneten Büchern von Mary und Tom Poppendieck orientiert.[5] Die Bedeutung schlanker Konzepte für integrierte Systeme lässt sich nicht genügend hervorheben. Die Prinzipien einer schlanken Softwareentwicklung, an deren Erarbeitung Mary und Tom Poppendieck federführend beteiligt waren, leiten sich von den Grundsätzen der schlanken Produktion ab. So wie diese Richtlinien den Produktionsbereich geradezu revolutioniert haben, so bewirkt auch die schlanke Softwareentwicklung tief grei-

Fußnoten:

[4] An An dieser Stelle sei noch einmal an das berüchtigte Krebsbestrahlungsgerät Therac-25 erinnert. Aufgrund eines Software-Design-Fehlers verursachte der Linearbeschleuniger Mitte der 80er-Jahre durch Verabreichung einer 100-mal höheren Strahlungsdosis als vorgesehen den Tod von mindestens zwei Patienten. Des Weiteren gilt es zu berücksichtigen, dass Software, die für Allzweckcomputer entwickelt wurde, in manchen Fällen zur Ausführung auf sicherheitsrelevanten Systemen kommt.

[5] Unter schlanker Softwareentwicklung sind grundlegende Ansätze für moderne, auf Agilität basierende Entwicklungsprojekte zu verstehen. Siehe Mary und Tom Poppendieck: Lean Software Development: An Agile Toolkit (Upper Saddle River/NJ, Addison-Wesley, 2003); Implementing Lean Software Development: From Concept to Cash (Upper Saddle River/NJ, Addison-Wesley, 2007) und Leading Lean Software Development: Results are Not the Point (Upper Saddle River/NJ, Addison-Wesley, 2010). Der Zusammenhang zwischen Agilität und einer schlanken Vorgehensweise wird bereits im Untertitel ihres ersten Buchs deutlich: Schlanke Verfahren können als Teilaspekt der Agilität betrachtet werden.

fende Veränderungen für das Software-Engineering. Da bei integrierten Systemen Hardware und Software kombiniert eingesetzt werden, ist es keine Überraschung, dass ein schlankes Vorgehen besonders Erfolg versprechend ist. Verschwendung zu vermeiden und den gesamten Wertschöpfungsprozess im Blick zu behalten, sind zwei der primären schlanken Prinzipien, die hierbei anwendbar und entscheidend sind.

Eine der größten Schwächen bei der Entwicklung von Embedded Systemen hängt oft damit zusammen, dass die Softwareentwickler zunehmend unter Druck geraten, während sie darauf warten müssen, dass eine funktionsfähige Hardware zur Verfügung steht. Schlanke Methoden – beispielsweise die Reduzierung oder Vermeidung von Übergaben, die Minimierung organisatorischer Schranken und die Fokussierung auf das Produkt als Ganzes anstelle einer punktuellen Konzentration auf einzelne Bestandteile oder nur der Hardware- bzw. Softwarekomponenten – können erhebliche Verbesserungen für die harmonische Durchführung von Projekten zur Entwicklung von Embedded Systemen bewirken.

Regulierung und Zertifizierung sicherheitskritischer integrierter Systeme und agiler Methoden

Sobald eine Zertifizierung oder die Einhaltung vorgegebener Regularien für ein Embedded System erforderlich sind, sind wir versucht, Agilität mit einem gewissen Maß an Skepsis zu betrachten – schließlich ziehen wir laut dem Agilen Manifest „funktionierende Software einer umfassenden Dokumentation“ vor. Manch einer sieht in dieser Aussage eine Rechtfertigung dafür, jegliche Dokumentation zu unterlassen – eine absurde Vorgehensweise in den Fällen, in denen Regulierungen oder Zertifizierungen ein Muss sind.

Im Rahmen eines agilen Projekts jedoch dokumentieren die Teammitglieder die jeweiligen Vorgänge nur soweit erforderlich. Setzt eine bestimmte Regulierung eine Dokumentation voraus, ist diese erforderlich und muss durchgeführt werden. Im Idealfall wird dabei nur die tatsächlich notwendige Dokumentation angefertigt, nicht mehr und nicht weniger. Wie jedoch bereits zuvor dargelegt wurde, geht es bei Agilität um viel mehr als nur um Dokumentation. Betrachten wir noch einmal genauer einige spezielle agile Verfahren sowie deren Zweckdienlichkeit bzw. Bedeutung im Zusammenhang mit der Entwicklung qualitativ hochwertiger Software:

- **Testgesteuerte Entwicklung** ist ein Verfahren zur Verbesserung der Codequalität. Bei ordnungsgemäßer Ausführung handelt es sich sogar um eine selbstdokumentierende Aktivität: Es existieren Tests für fertiggestellten Code, die dessen Richtigkeit belegen. Kann ein Team *ohne* testgesteuerte Entwicklung Vorschriften einfacher einhalten? Die Antwort lautet ganz klar: Nein.
- **Pair Programming** ist eine Taktik zur Verkürzung der Zeitspanne zwischen dem Auftreten und dem Erkennen eines Fehlers. Kann ein Team *ohne* Pair-Programming-Vorschriften einfacher einhalten? Die Antwort lautet erneut: Nein.
- **Refactoring** ist „eine genau festgelegte Methode zur Umstrukturierung von vorhandenem Code, wobei die interne Struktur, nicht jedoch das externe Verhalten geändert wird.“[6] Durch diesen systematischen Ansatz zur Neustrukturierung von Code kann ein Team den betreffenden Code besser verstehen und verwalten. Ist ein unverständlicher, schwierig zu verwaltender Code für sicherheitsrelevante Systeme zu bevorzugen? Auch hier lautet die eindeutige Antwort: Nein.
- Schließlich sei noch das als **kontinuierliche Integration** bezeichnete agile Verfahren genannt. Eines der größten Hindernisse für die Gewährleistung hoher Softwarequalität stellt die späte Integration von Softwarekomponenten dar – einfach ausgedrückt: Probleme, die im letzten Moment erkannt werden, lassen sich am schwierigsten lösen. Folglich ist es für den Erfolg agiler Konzepte notwendig, häufig oder vielmehr kontinuierlich Softwarekomponenten zu integrieren, damit Software beständig (idealerweise in zunehmendem Umfang auch automatisch) getestet werden kann und Risiken auf ein Minimum begrenzt werden.

Es gibt noch zahlreiche weitere agile Verfahren, die im Allgemeinen ebenso zu einem hohen Qualitätsniveau führen und sich in keinerlei Weise nachteilig auf die Einhaltung regulatorischer Vorgaben oder Zertifizierungen auswirken (Automatisierung, Stand-up-Meetings, Codierungsstandards, nachhaltige Arbeitsgeschwindigkeit, Demos usw.).

Ganz ähnlich zeichnen sich auch agile Methoden (z. B. Scrum, XP, DSDM, RUP) durch diesen wichtigen dynamischen Aspekt aus: die Arbeit in kurzen, stabilen, zeitlich exakt festgelegten Iterationen. Die Auswirkungen auf die Qualität sind erheblich – Komponenten werden häufiger und regelmäßiger integriert (d. h. der Zeitraum zwischen Einführung und Entdeckung eines Fehlers wird verkürzt), es erfolgen vermehrt Tests für Einzelelemente, Demonstrationen für funktionierende Software sind möglich usw. Obwohl Regulierungen und Zertifizierungen häufig große Herausforderungen darstellen, verhelfen agile Verfahren und Methoden ohne Zweifel in jedem Fall zu einer Verbesserung der Softwarequalität und versetzen das Team in eine bessere Position für erfolgreiche Zertifizierungen anstatt diese zu behindern.

Es gibt in der Tat Herausforderungen bei der Dokumentation und diese müssen beseitigt werden. Doch die Probleme im Bereich der Softwarequalität sind im Hinblick auf Regulierungen und Zertifizierungen eine noch dringlichere Angelegenheit. Anhand agiler Verfahren und Methoden lässt sich diese Situation beträchtlich verbessern. Bezüglich der Dokumentation sind zwei einfache Aspekte gründlich zu überdenken:

- Welche Dokumentation ist tatsächlich erforderlich?
- Wie lässt sich diese Dokumentation am effizientesten bereitstellen?

Sobald eine klare Antwort auf diese Fragen feststeht, kann ein Plan erarbeitet werden, der festlegt, wie viel Dokumentation für ein konkretes Embedded Systems-Projekt erforderlich und angemessen ist. In einem weiteren Plan lässt sich eine geeignete Projektsteuerung festlegen.

Erweiterung agiler Konzepte für Embedded Systeme durch Modellierung

Die Modellierung ist hervorragend dazu geeignet, agile Verfahren und Methoden zu ergänzen und somit zu Effizienz- und Produktivitätssteigerungen von Teams beizutra-

Fußnoten:

[6] Siehe Definition von Martin Fowler unter: http://www.refactoring.com/.

gen. Durch Modellierung lassen sich lange Wartezeiten aufgrund nicht verfügbarer Hardware und folglich verzögerte Softwaretests vermeiden. Somit spielt die Modellierung eine immer bedeutendere Rolle bei der erfolgreichen Entwicklung von Embedded Systemen.

Agile Entwicklung – Begriffe und Definitionen

Testgesteuerte Entwicklung (Test Driven Development): Bei diesem Verfahren wird zuerst der Test vor dem Code geschrieben (damit der geschriebene Code unverzüglich überprüft werden kann). Zudem müssen Entwickler nur den zur Einhaltung der jeweiligen Anforderung notwendigen Code schreiben (nicht mehr und nicht weniger).

Pair Programming: Bei dieser Arbeitstechnik erstellen zwei Programmierer den Code, wobei sie abwechselnd Code schreiben („Fahrer") und den vom anderen geschriebenen Code überprüfen („Beifahrer"). Der „Fahrer" erstellt den Code, der „Beifahrer" ermittelt Fehler oder Verbesserungsmöglichkeiten. Auf diese Art wird die Gesamtqualität des Codes optimiert.

Tägliche (Daily) Scrums (bzw. Stand-up-Meetings): Im Idealfall werden diese 15-minütigen Teambesprechungen täglich abgehalten. Die Kommunikation erfolgt geregelt, effizient und offen. Dabei geht es um folgende drei Fragen: Was wurde gestern erledigt? Was wird momentan bearbeitet? Welche Hindernisse bestehen?

Kontinuierliche (Continuous) Integration: Hierbei integrieren einzelne Teammitglieder ihren Code kontinuierlich, sodass Probleme, die häufig aufgrund von unregelmäßiger Integration von Code entstehen, auf ein Minimum begrenzt werden.

Nachhaltige Arbeitsgeschwindigkeit (Sustainable Pace): Da das Team seine Arbeitsgeschwindigkeit (Menge an Aufgaben, die pro Iteration durchgeführt wird) überwacht, kann es festlegen, wie viele Aufgaben regelmäßig fertigzustellen sind, ohne die Mitglieder des Teams zu überfordern.

Oftmals wird Modellierung bei der Entwicklung von Embedded Systemen eingesetzt, um das Design zu verfeinern (eine iterative Vorgehensweise, bei der schrittweise der Detaillierungsgrad erhöht wird). Das in der Entwicklung befindliche System wird durch eine simulierte Modellausführung getestet. Dadurch ist es möglich, das System frühzeitig auf Stabilität und Vollständigkeit hin zu prüfen und mögliche Deadlocks bei der Interaktion der unterschiedlichen Prozesse zu ermitteln. Im gesamten Modell können auf diese Weise Fehler behoben werden, ohne dass der eigentliche Code auf dem Zielsystem ausgeführt werden muss. Das bedeutet, nur das Verhalten der Ablaufsteuerung, die Integration sowie ähnliche Aktivitäten sind im Zielsystem zu testen. IBM Rational Rhapsody, eine modellgesteuerte Umgebung, sowie Tools zur Erweiterung, etwa IBM Rational Rhapsody TestConductor und IBM Rational Test RealTime, eignen sich für diesen Zweck.

Fazit

Agile Softwaremethoden und -verfahren können in Kombination mit Modellierung und sonstigen Strategien die Grundlage eines Prozesses bilden und dadurch für eine effektivere Bereitstellung von Embedded Systemen sorgen. Wir haben die Erfahrung gemacht, dass eine disziplinierte Anwendung agiler Konzepte, wobei Teams nach eigenem Ermessen die jeweils wirksamsten Methoden und Verfahren auswählen, die besten Aussichten auf Erfolg hat. Wenn agile – oder beliebige sonstige – Strategien als vorgefertigte, unflexible Modelle implementiert und unabhängig von der aktuellen Ausgereiftheit des Teams und des Projekts bestimmte Methoden und Verfahren durchgesetzt werden, lassen Schwierigkeiten nicht lange auf sich warten. Das heißt also, Agilität ist weder die perfekte Methode noch eine typische Anfängermethode, es handelt sich vielmehr um ein bestimmtes Konzept – ein Konzept allerdings, das im Hinblick auf die Entwicklung von Embedded Systemen ausgezeichnete Vorteile birgt. ■

Renate Stücka

Senior Market Manager verantwortlich für Marketing und Kommunikation im Bereich IBM Rational Software. Zudem Erfahrungen im Business Development und Partnermanagement. Mehr als 10 Jahre in verschiedenen Positionen verantwortlich für Marketing, Vertrieb und Entwicklung komplexer Software- und Kommunikationsprodukte.
(renate.stuecka@de.ibm.com)

Ted Rivera

Senior Architect in IBM-Teams mit besonderem Fokus auf Verbesserungen im Software Engineering bei kommerziellen Softwareprojekten. Langjährige Erfahrung in Entwicklung, Test, Support, Dokumentation und Lieferung kommerzieller Software. War verantwortlich für die Entwicklung von praktischen Trainingsmaterialien über Agile und Schlanke Methoden und Practices, anerkannter Experte für die positive Transformation von Softwareentwicklungsorganisationen.
(trivera@us.ibm.com)

Intelligente Technologien für einen smarten Planeten

Was bedeuten 3 Millionen Zeilen Programmcode für einen Koffer?

Sie bedeuten, dass man am Flughafen Amsterdam Schiphol bald jedes Jahr 70 Millionen Gepäckstücke zuverlässig und effizient bewegen kann – 20 Millionen mehr als zuvor. Das automatisierte Gepäcksystem ermöglicht es, die Kapazität um bis zu 40 % zu steigern. Denn die Nachfrage wächst beständig, und Schiphol ist heute bereits einer der verkehrsreichsten Flughäfen in Europa. Das System basiert auf IBM Rational® und Tivoli® Software und läuft auf Power Systems™ Servern. Ein smartes Unternehmen braucht intelligente Software, Systeme und Services.

Machen wir den Planeten ein bisschen smarter. **ibm.com**/gepaeck/de

IBM®

Hier werden Daten sichtbar gemacht, die bei der Gepäckabfertigung am Flughafen Amsterdam Schiphol entstehen.