Werner Schoepe

(E-Mail: werner.schoepe@de.ibm.com)
ist Field Technical Professional bei der IBM in Düsseldorf. Mit seiner 6-jährigen Erfahrung bei IBM Rational ist er vertriebsunterstützend für Rational Produkte tätig. Einen Schwerpunkt seiner Tätigkeit bildet die Beratung bei der Einführung von jazzbasierten Produkten wie Rational Team Concert bei Kunden.

Auch in großen Projekten zum Erfolg – Agilität richtig skalieren

Das agile Manifest und dessen Prinzipien [agi] sind nun mehr als neun Jahre alt und man beobachtet bei vielen Unternehmen agile Projekte. Agile Methoden werden nicht mehr nur in kleinen, lokalen, überschaubaren Projekten eingesetzt, sondern auch bei großen und sogar verteilten. Einige große Unternehmen wie auch IBM setzen agile Softwareentwicklung sogar schon auf breiter Front ein und erzielen deutliche Wettbewerbsvorteile durch die verbesserten Vorgehensweisen. IBM hat z. B. in einem mittelgroßen Projekt (ca. 90 Personen) das Produkt Rational Team Concert an 7 Standorten verteilt und agil entwickelt. Viele Eigenschaften, die die wesentlichen Stärken der agilen Softwareentwicklung ausmachen, geraten in großen Projekten jedoch schnell in Gefahr. Zum Beispiel geht in großen Projekten leicht der Fokus auf das erste Wertepaar des agilen Manifests verloren: „Menschen und deren Zusammenarbeit sind wichtiger als Prozesse und Werkzeuge". Die Zusammenarbeit ist oft gefährdet, kann aber mit einer neuen Generation von Werkzeugen verbessert werden. Im Folgenden betrachten wir daher, welche agilen Skalierungsmöglichkeiten es gibt, welche Herausforderungen dabei auftreten und wie ihnen mit Werkzeugen begegnet werden kann.

Vom *Core Agile Development* zum *Disciplined Agile Delivery*!

Agile Projekte sind originär durch kleine Teams, die zusammen an einem Ort arbeiten, gekennzeichnet. Die Teams arbeiten eigenverantwortlich und verfolgen einen mehrwertgetriebenen Lebenszyklus, d. h. mit jeder Iteration wird die Software für die Anwender wertvoller, denn in jeder Iteration werden neue Anforderungen umgesetzt und einsetzbarer Code erstellt. Dabei werden die Stories mit dem größten Nutzen für das Geschäft vorrangig realisiert. Hier ist Scrum als Vorgehensmodell am meisten verbreitet. Abbildung 1 veranschaulicht exemplarisch dieses Vorgehen.

Die Einbettung des agilen Vorgehens sowohl in den gesamten Produktzyklus als auch in die Unternehmensumgebung sieht Scott Ambler [Amb] in den agilen Modellen nur unzureichend berücksichtigt und betrachtet diese Problematik als ersten Skalierungsfaktor. Er präzisiert das agile Vorgehen, indem er die Entwicklung auf dem Zeitstrahl in Phasen einteilt (siehe Abbildung 2) und er nennt dies „Disciplined Agile Delivery" (DAD). Die Begriffe der Phasen sind manchem aus dem Rational Unified Process (RUP) bekannt – inhaltlich verbirgt sich jedoch eine viel leichtgewichtigere Vorgehensweise:

- **Inception:** Agile Projekte fallen nicht einfach vom Himmel, vielmehr sind bestimmte vorbereitende Arbeiten notwendig, wie z. B. ein leichtgewichtiges Requirement Engeneering mit der Absteckung des Projektumfangs, der

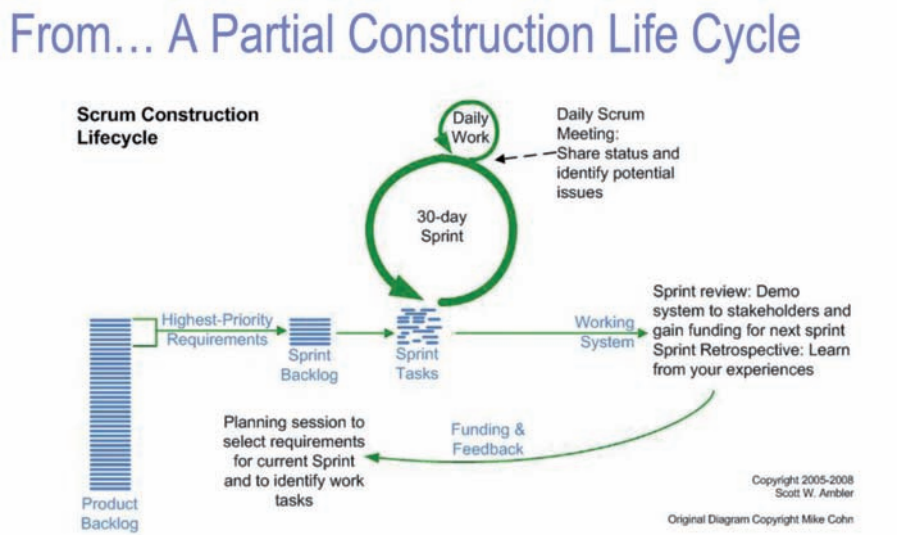

Abb. 1: Agile Softwareentwicklung mit Scrum

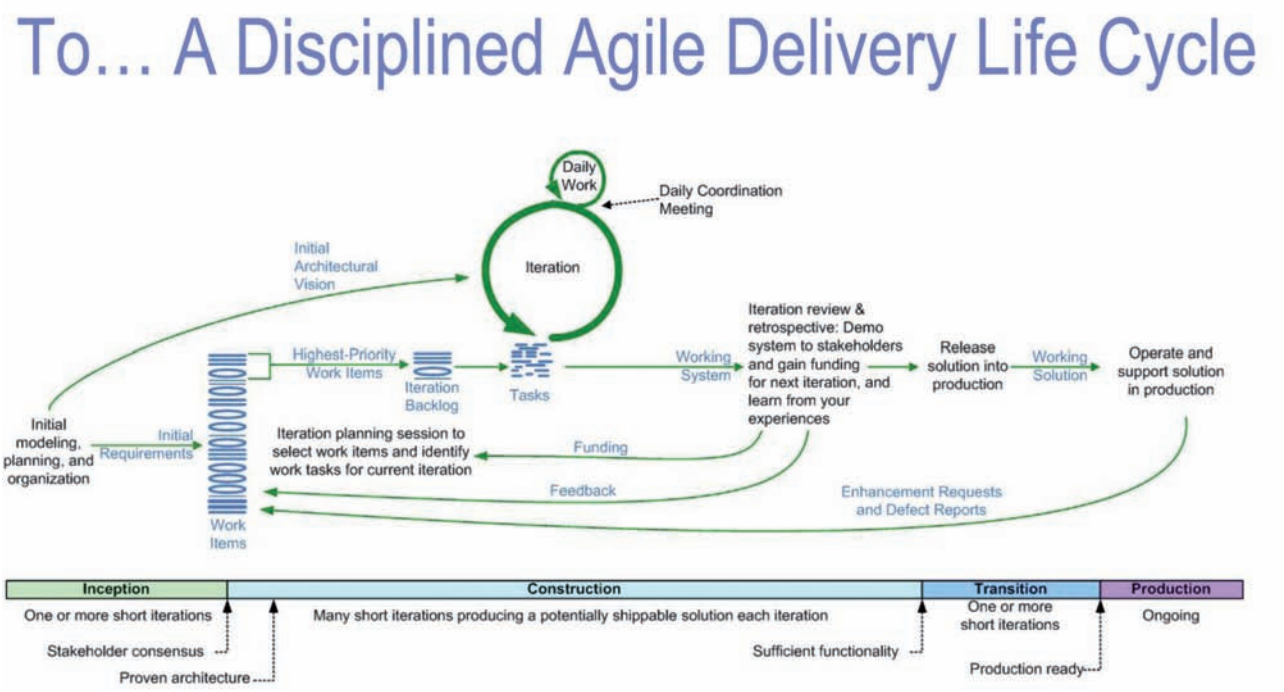

Abb. 2: Vollständiger agiler Lebenszyklus

initialen Projektfinanzierung, der Teambildung, dem Aufbau der Arbeitsumgebung und möglicherweise schon ein erstes initiales Increment. Viele „Agilisten" versuchen dem mit einer Iteration 0, einem Sprint 0 oder einer „Warm-Up-Phase" zu begegnen. Der Charakter dieser Iteration unterscheidet sich jedoch deutlich von der folgenden, sodass eine Auslagerung in eine separate Phase sinnvoll erscheint.

- **Construction:** Diese Phase bildet den iterativen Kern, der weitgehend mit einem agilen Prozeßmodell abgebildet werden kann. In Abbildung 2 ist die Ähnlichkeit mit Scrum nicht zu verkennen. Die Terminologie ist etwas verändert und verallgemeinert.
- **Transition:** In der *Transition*-Phase werden das System getestet und für die Praxis ‚robust' gemacht, die Benutzer geschult, die Dokumentation vervollständigt und schließlich die Software in Produktion gegeben.
- **Production:** Die Produktion sollte mitbetrachtet werden, weil von hier viele Änderungs- und Erweiterungswünsche sowie Fehlermeldungen kommen werden, die vom agilen Team zu bearbeiten sind.

Dieser umfassende Lebenszyklus ist also im Großen sogar sequenziell und im Kleinen iterativ. Dabei bildet die Construction-Phase den iterativen, agilen Kern und wird umklammert von der Inception-Phase am Anfang und der Transition- und der Production-Phase am Ende.

Ein großer Vorteil iterativer Prozesse ist die Möglichkeit, schon frühzeitig dem Projektrisiko entgegenzuwirken, indem die besonders risikoträchtigen Teile des Projekts in frühen Iterationen bearbeitet werden. In agilen Projekten kann es zu Zielkonflikten mit dem mehrwerteorientierten Vorgehen kommen. „Disciplined Agile Delivery"-Teams betrachten unterschiedliche Projektrisiken und finden hier eine angemessene Balance.

In diesem „Disciplined Agile Delivery"-Prozess wird der Backlog auch von der Produktion befüllt. Mit dieser ganzheitlichen Sicht der zu erledigenden Aufgaben wird die Iterationsplanung erleichtert und in Abbildung 2 wurde der Begriff *Backlog* mit „Work Item-Liste" allgemeiner bezeichnet.

Auch wenn agile Teams eigenverantwortlich arbeiten, sich selber steuern und dabei ihre Geschwindigkeit ständig messen, so müssen sie im Rahmen der IT-Governance auch laufenden Reportingverpflichtungen ans höhere Management nachkommen. Hierbei sind die vorgegebenen Metriken zu beachten. Mit komfortablen Werkzeugen, die sich in der Arbeitsumgebung des Entwicklers (z. B. Eclipse oder Visual Studio) einbetten, ist dies weitgehend automatisiert und damit ohne nennenswerten Mehraufwand für den Entwickler möglich. Mit den Produkten der Jazz-Plattform findet er hierfür komfortable Unterstützung [Jazz].

Viele Unternehmen haben die Problematik erkannt und sind zu Lösungen gekommen, die dem „Disciplined Agile Delivery" entsprechen: im Kern haben sie einen agilen Prozess in ihren umfassenden Entwicklungszyklus eingebettet. Der Erfolg gibt ihnen recht, andererseits hätten sie oft viel Zeit und Geld gespart, wenn sie auf einen vorgedachten, ausgearbeiteten „DAD"-Prozess wie dem Agile Unified Process (AUP) [AUP] zurückgegriffen hätten.

Agility@Scale™ – Herausforderungen bei skalierten agilen Projekten [ScA]

Der Projektumfang vieler agiler Projekte hat sich jedoch erweitert und man kann folgende Skalierungsfaktoren unterscheiden:

1. **Teamgröße:** Die agilen Praktiken sind für Teamgrößen von ca.10 Personen designed und mit Erfolg erprobt. Was ist jedoch, wenn die Anwendung größere Teams erfordert von z. B. 100 Personen? Wie ist die gute Kommuni-

kation und Zusammenarbeit, auf die für den Erfolg beim agilen Entwickeln besonderer Wert gelegt wird, aufrechtzuerhalten? Die papierbasierten Standardmethoden versagen hier schnell.

2. **Geografische Verteilung:** Wird das Team räumlich getrennt, entstehen sofort die gleichen Kommunikationsprobleme wie bei Punkt 1. Dabei muss nicht immer gleich ein Team in Übersee sein, sondern auch übliche Gegebenheiten, z. B. dass Kollegen auf einer anderen Etage sitzen, häufiger im Homeoffice arbeiten oder Partnerfirmen in einem anderen Gebäude/einer anderen Stadt mitarbeiten, erschweren die Kommunikation schon deutlich. Verteilt sich das Team auf unterschiedliche Zeitzonen, wird die Zusammenarbeit zusätzlich erschwert.
3. **Regulatorische Bestimmungen:** Manche Firmen unterliegen regulatorischen Bestimmungen, die von außen vorgegeben sind. Entwicklung im medizinischen Bereich sei hier als Beispiel genannt. Die Entwickler haben zusätzlichen Aufwand, die Regularien zu interpretieren, die Ziele vorgeben. Sie haben aber keine konkreten Anweisungen, wie diese umzusetzen sind. Komplexität und Aufwand erhöhen sich damit, oft droht ebenfalls eine erhebliche Bürokratie zu entstehen, um den Bestimmungen zu genügen – und Prozessbürokratie will man ja gerade in agilen Projekten vermeiden.
4. **Anwendungskomplexität:** Manche Anwendungen sind in ihrer Komplexität recht überschaubar. Eine „normale" Datenbankanwendung mit Zugriff vom Web sollte heute relativ gut einschätzbar sein. Entwickelt man hingegen ein Flugkontrollsystem, so hat man in vielerlei Hinsicht eine erhebliche Komplexitätssteigerung, auf die man beim Entwickeln mit besonderen Maßnahmen reagieren wird. Auch in technischer Hinsicht wird man unterschiedliche Komplexitätsgrade beobachten: Neu zu schaffende Anwendungen sind oft leichter zu realisieren als große, gewachsene, alte Anwendungen im heterogenen Umfeld.
5. **Organisatorische Verteilung und Komplexität:** In manchen Projekten ist das Team aus Mitgliedern verschiedener Abteilungen, Partnerfirmen oder Freiberuflern zusammengesetzt. Die Zusammenarbeit kann hier stark von den vertraglichen Rahmenbedingungen und politischen Aspekten und nicht von einem offenen, vertrauensvollen Verhältnis zueinander, wie es essenziell in agilen Projekten gefordert ist, geprägt sein. Wenn in verschieden Unternehmensteilen widersprüchliche Arbeitsweisen und Visionen existieren, erhöht sich die Komplexität in agilen Projekten erheblich. Auch kulturelle Unterschiede spielen oft eine große Rolle.
6. **Unternehmenseinbettung (Enterprise Discipline):** Die Unternehmen erzielen Kostenvorteile durch eine gemeinsame Nutzung von IT-Infrastrukturen und -Architekturen. Nicht immer drängen sich diese Architekturen dem agilen Team für die schnellste Lösung auf. Hier muss das Team eng mit den Architekten zusammenarbeiten um Lösungen zu schaffen, die konsistent in die Unternehmenslandschaft passen.

Jedes Projekt ist für sich individuell und man findet eine andere Zusammenstellung der Skalierungsfaktoren. Entsprechend wird man die agilen Praktiken immer aufs Neue auswählen müssen. Hier findet sich auch eine Parallele zum RUP, den man am Anfang von Projekten immer zuschneidern muss. Aus Unkenntnis der großen Menge von Prozessteilen, Artefakten etc. wird dieser Prozess oft nur unzureichend durchgeführt und im Ergebnis kommt oft ein zu fetter Prozess heraus. Um das zu vermeiden wird hier empfohlen, umgekehrt vorzugehen, d. h. man nimmt sukzessive sinnvolle Praktiken hinzu bis das Portfolio angemessen ist. Einsparungen bei der agilen Entwicklung ergeben sich insbesondere daraus, möglichst wenig zu machen und viel überflüssigen „Müll" zu vermeiden.

Werkzeuge bei der Lösung von Skalierungsproblemen

Viele der genannten Herausforderungen sind zunächst einmal mit „konventionellen" Methoden anzugehen. „Scrum of Scrums" helfen bei der Abstimmung in größeren Projekten. Mit Feature-Teams lässt sich der Kommunikationsbedarf zwischen den Teams reduzieren [Woo10]. Eine gute technische Kommunikationsinfrastruktur sollte selbstverständlich sein [BuH].

Bei großen Projekten wird man aber nicht umhinkommen, Werkzeuge zum Management des Application Lifecyles einzusetzen (ALM). Eine ganze Reihe von Artefakten und Praktiken (Backlogs mit Stories, Continuous Integration, TDD) beim agilen Entwickeln lassen sich gut durch Werkzeuge verwalten bzw. unterstützen.

Der Produkt Backlog muss von einem geeigneten Werkzeug gemanagt werden. In vielen agilen Projekten wird er mit Excel-Sheets verwaltet. Dies endet bei entsprechender Größe und Verteilung unter vielen Teams oft in einem Albtraum. Die Versionen der Excel-Dateien werden zwangsläufig irgendwann auseinanderlaufen. Vieles von dem agilen Produktivitätszugewinn wird wieder verschenkt. Die Minimallösung ist hier ein zentrales, mehrbenutzerfähiges Repository, das alle Stories aufnimmt und verwaltet. Damit haben alle Stakeholder einen gemeinsamen, immer aktuellen Blick auf die Anforderungen.

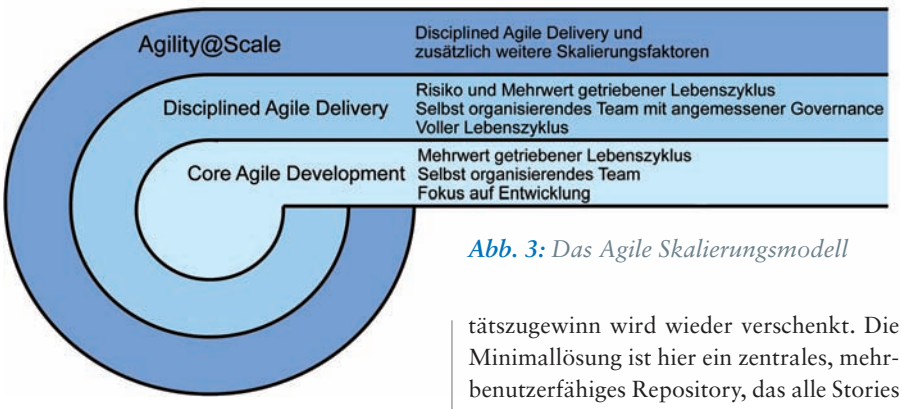

Abb. 3: Das Agile Skalierungsmodell

Auch lassen sich damit unterschiedliche Sichten auf die Menge der Anforderungen leicht realisieren: Neben dem Produkt Backlog und dem Sprint Backlog, die beide Stories des gesamten Projektes enthalten, wird man teamorientierte Sichten finden. Das Team möchte die eigenen Stories und

Aufgaben im Überblick haben und verwalten. Aber auch komponentenorientierte und domainorientierte Sichten sind nützlich.

Die tägliche Planung muss durch Werkzeuge unterstützt werden, das Arbeiten mit Tasksboards, wie in kleinen Teams üblich, eignet sich nicht. Im Gegensatz zur realen Wand, verschafft die elektronische Variante Überblick auch über Standorte hinweg. Sie wird aber nur angenommen, wenn sie sich homogen in die Arbeitsumgebung des Entwicklers integriert, sodass er beim Pflegen seiner Daten keinen Werkzeugwechsel vornehmen muss. Erledigungen von Aufgaben können leicht im Tool vermerkt und automatisch in den zugehörigen Stories aggregiert werden. Der Entwickler hat dabei keinen Mehraufwand, die Projektleitung hat immer einen aktuellen Überblick zum Stand des Projektes. Hier sollten nur aggregierte Daten visibel werden, um die Vertraulichkeit im Projektteam zu wahren.

Agile Teams können durch Werkzeuge auch eine Prozessunterstützung bekommen. Sie möchten sich ständig verbessern und deswegen auch möglichst leicht Prozessanpassungen vornehmen können. Dies gilt gewiss vor einem Projekt, aber häufig auch nach einer Retrospektive am Ende eines Sprints.

Im regulatorischen Umfeld werden die Werkzeuge die historische Entwicklung von den Artefakten aufzeichnen sowie die Einhaltung von Prozessen gewährleisten.

Skalierte agile Projekte mit der Jazz-Plattform

Eine neue, von Rational entwickelte Palette von Werkzeugen [Jazz] berücksichtigt die Belange von großen agilen Teams in besonderer Weise. Dies konnte dadurch erreicht werden, dass ein erfahrenes agiles Team um Erich Gamma sich die Plattform zum Entwickeln selber schaffen durfte. Es hat schon sehr früh die Jazz-Plattform mit Rational Team Concert (RTC) genutzt, um das Produkt zu entwickeln. Ständig hat das Team gespürt, was noch zu verbessern ist und was noch fehlt. Architektonisch ist Jazz die allgemeine Plattform für Softwareentwicklungswerkzeuge, welche Basisfunktionalitäten zur Verfügung stellt. Werkzeuge integrieren sich nur noch in die Plattform und nicht mehr untereinander. Rational Team Concert war das erste Produkt auf dieser Plattform. Fast alle Paradigmen der agilen Softwareentwicklung und die oben erwähnten Herausforderungen werden hier unterstützt. Allerdings nicht, wie häufig am Markt sichtbar, als eine Sammlung einzelner Werkzeuge, sondern voll integriert in einem Produkt. Aus diesen Integrationen kann das agile Team besondere Vorteile ziehen, die wir im Folgenden betrachten möchten.

Ein **Instant Messaging System** hat Rational tief in Team Concert eingepflanzt: Wie zu erwarten, findet sich ein Chat-Fenster als Eclipse-View in der Arbeitsumgebung. In einer „Team View“ hat der Benutzer Überblick zur Teamzusammensetzung und bekommt angezeigt, ob die Teammitglieder online sind. Der eigentliche Mehrwert ergibt sich aber daraus, dass in allen Views, in denen die User als Eintrag auftauchen, gleich deren Verfügbarkeit angezeigt wird. Der Benutzer kann sofort einen Chat starten und der Partner bekommt implizit einen Link zum Kontext in sein Chatfenster mitgeliefert. Mit einem Klick hat er dasselbe Artefakt auf dem Schirm und die Diskussion kann ohne aufwendiges Suchen starten. Rational bezeichnet diese Funktionalität als **„Collaboration in Context“**.

Für Team Concert wurde ein Configuration Management neu entwikkelt, das agile Praktiken wie Refaktoring, Continuous Integration besonders gut unterstützt. Möglich wird dies durch ein Konzept von Change Sets. Alle Änderungen werden in einem oder mehreren Change Sets zusammengefasst. Die Change Sets kann man fast wie ein Objekt behandeln, sodass viele Standard Use Cases für den Benutzer recht einfach werden. Zum Beispiel lassen sich Änderungen sehr einfach wieder rückgängig machen, Fehlerbehebungen sind auch in anderen Softwareversionen leicht einzubringen. Eine tiefe Integration mit dem Change Management System macht das Navigieren vom Work Item zum Code und umgekehrt effizient möglich.

Ein mächtiges Buildmanagementsystem ist in RTC integriert. Jeder Build ist ein eigenes Objekt in dem zentralen, vernetzten Repository. Der Benutzer kann natürlich zunächst ein lokales Build anstoßen. Versagt das globale Build, so finden sich in diesem Build-Objekt wichtige Informationen für die Fehleranalyse. Folgende Fragestellungen des Bearbeiters erhält er beantwortet:

- Was hat sich im Code seit dem letzten erfolgreichen Build geändert? Welche Aufgaben wurden bearbeitet und welcher Code hängt damit zusammen? Möglicherweise müssen diese Änderungen wieder rückgängig gemacht werden.
- Beim Debuggen stellt sich die Frage, wie der Quellcode zum Zeitpunkt des Builds ausgesehen hat? Dies wurde automatisch in Form eines Snapshots im Repository „eingefroren“ und ist jederzeit leicht (agil) wieder herstellbar.
- Welche Unit-Tests wurden mit welchem Ergebnis ausgeführt? Falls hier ein Fehler auftrat möchte man zum Testfall navigieren können.
- Wie sehen die Log-Files aus?

Im Kontext des Builds kann mit einem Klick ein „Defekt“ eröffnet werden, der einen Link auf den Build Record enthalten wird. Somit sind alle Informationen automatisch miteinander verlinkt und beim Lösen des Problems leicht verfügbar.

Bei Projekten, die **regulatorische Bestimmungen** zu beachten haben, ist man oft gezwungen alle Softwareentwicklungsaktivitäten zu dokumentieren. Rational Team Concert übernimmt dies für die Entwickler weitgehend, indem die Historie der Objekte automatisch festgehalten wird.

Team Concert verfügt über ein ausgefeiltes **„Notification“-Konzept**. Ziel ist es den Benutzer über alle Aktivitäten im Team auf dem Laufenden zu halten. Die Ereignisse werden in einen Event-Fenster innerhalb seiner Arbeitsumgebung angezeigt. Themen (z. B. eine Story, eine Aufgabe) können gezielt abonniert werden und man kann so seinen individuellen Informationsbedarf zusammenstellen. So kann er z. B. Meldungen bekommen, ob sich irgendetwas in einem bestimmten Work-Items verändert hat, ob es erledigt wurde, ob Defects von denen er abhängig ist, behoben wurden oder ob ein Build fehlgeschlagen ist. Im Event ist natürlich gleich der Link zum Objekt vermerkt, sodass man bequem dort hinnavigieren kann.

RTC weiß, in welchem Stadium sich gerade ein Projekt befindet und unterstützt hierdurch gewisse, vom Team selbst aktivierte Regeln (**Process Awareness**). So kann es sich in einem Sprint aus Qualitäts-

gründen eine Phase „End Game“ definieren und das System wacht darüber, dass dem System jetzt keine neuen Entwicklungen übergeben werden, sondern nur noch Fehlerbehebungen.

Der Projektüberblick wird über vordefinierte (natürlich selbst konfigurierbare) **Dashboards** ermöglicht. Auch die Velocity wird mit Burndown-Charts und anderen Visualisierungen dargestellt.

Die Projekt-Owner und andere Stakeholder haben auch die Möglichkeit, über eine umfassende und sehr komfortable Web 2.0-Oberfläche mittels Browser auf die Projektinformationen zuzugreifen.

Mit all diesen Funktionalitäten erfüllt RTC alle Anforderungen, die agile Teams an Werkzeuge stellen. Bei Disciplined Agile Delivery, wo man auch unabhängige Tester findet, kann man als Testmanagement Werkzeug **Rational Quality Manager** hinzunehmen. Der Quality Manager, mit dem man Tests systematisch verwaltet und ausführt, ist ebenfalls jazzbasiert und eine Integration zu Rational Team Concert ist „out-of-the-box“ gegeben.

Rational Team Concert ist das erste Produkt von Rational, das konsequent auf die Bedürfnisse agiler Teams nach der Maxime „Team First“ zugeschnitten ist, es berücksichtigt das erste agile Wertepaar [agi] in besonderer Weise. Unzählige Funktionen wurden geschaffen, die das Leben des agilen Teams leichter machen. Gerade in skalierten Projekten tragen die integrierten „Collaboration-Features“ zum Projekterfolg bei.

Kleinen Teams, die aus Kostengründen gerne auf Open Source-Werkzeuge zurückgegriffen und solche Integrationen mühsam und teuer von Hand erstellt haben, stellt IBM eine sehr leistungsfähige kostenfreie Version (Express-C) zum Download auf „jazz.net“ [Jazz] zur Verfügung. ■

Referenzen

[agi] www.agilemanifesto.org

[Amb] Scott W. Ambler: http://www.ambysoft.com/essays/agileLifecycle.html

[Jazz] Home Page von Jazz: www.jazz.net

[AUP] The Agile Unified Process: http://www.ambysoft.com/unifiedprocess/agileUP.html

[Woo10] Elizabeth Woodward: A Practical Guide to Distributed Scrum, IBM Press; 1 edition (July 1, 2010)

[ScA] Scaling Agile: An Executive Guide
ftp://public.dhe.ibm.com/common/ssi/sa/wh/n/raw14211usen/RAW14211USEN.PDF

[BuH] Baron, Pavlo; Hüttermann, Michael: Fragile Agile: Agile Softwareentwicklung richtig verstehen und leben; Hanser Verlag