Dr. Baris Güldali
(bguldali@s-lab.uni-paderborn.de)
ist Senior Researcher im s-lab – Software Quality Lab der Universität Paderborn (http://s-lab.upb.de). Seine Arbeitsschwerpunkte sind Requirements-Engineering und Testprozessverbesserung. Er ist Mitglied im Leitungsgremium der Fachgruppe TAV (Test, Analyse und Verifikation von Software) in der Gesellschaft für Informatik.

Masud Fazal-Baqaie
(mfazal-baqaie@s-lab.uni-paderborn.de)
ist Researcher im s-lab. Seine Forschungsinteressen umfassen die Entwicklung von projekt- und organisationsspezifischen Softwareentwicklungsmethoden. Er ist Mitglied im Leitungsgremium der Fachgruppe „Vorgehensmodelle für die betriebliche Anwendungsentwicklung“ in der Gesellschaft für Informatik.

Skalieren von großen agilen Projekten mit verteilten Backlogs

Agile Methoden befürworten selbstorganisierende Teams, die lauffähige Zwischenstände in kurzen Iterationen schrittweise entwickeln. Allerdings besteht in großen, verteilten Projekten mit mehreren Teams die besondere Herausforderung, die Aufgaben entsprechend zu definieren, zu verteilen, zu koordinieren und zu überwachen. Der Artikel beschreibt ein Vorgehen, wie diese Tätigkeiten mittels verteilter Backlogs unterstützt werden können, um den Herausforderungen verteilter Projekte zu begegnen.

Agiles Vorgehen ist die Antwort auf die sich schnell ändernden oder anfangs nur unvollständigen Anforderungen an die Software. Mit Hilfe von selbstorganisierenden Teams wird in kurzen Iterationen der gesamte Lebenszyklus von Softwareentwicklungsprozessen wiederholt durchlaufen und damit die Software schrittweise über lauffähige Zwischenstände zum auslieferbaren Endstand entwickelt.

Allerdings skalieren die Grundideen der Agilität zur Rollen- und Aufgabenverteilung und der Kommunikation nicht immer für Großprojekte, insbesondere wenn zeitlich, organisatorisch und örtlich verteilte Teams zusammenarbeiten müssen. Das beginnt beispielsweise bei der Teamgröße, die die empfohlene Größe (3 bis 9 nach [SuSch13]) von agilen Teams übersteigt. Darüber hinaus ist es nicht möglich ein morgendliches Statusmeeting (Daily Scrum) abzuhalten, wenn ein Teil des Teams wie bei Offshore-Projekten üblich in einer anderen Zeitzone verweilt.

Es existieren bereits verschiedene Ansätze (wie SAFe [Le11, Jan15] und LeSS [La05]), die sich mit dem Skalieren von agiler Entwicklung beschäftigen und die teilweise kontrovers diskutiert werden. Diese Ansätze adressieren verteilte Entwicklung mit über 20 Personen im Team und verfolgen meist das Prinzip *divide and conquer*, also das Beherrschen der Komplexität des Projektmanagements durch Aufteilen in kleine Subprojekte.

Diese Ansätze beschreiben die Prinzipien des Skalierens von agiler Entwicklung, allerdings fehlt es meist an operativen Beschreibungen, wie das Skalieren genau funktionieren soll. Mit unserem Beitrag möchten wir anhand eines Anwendungsbeispiels diese Lücke schließen und über

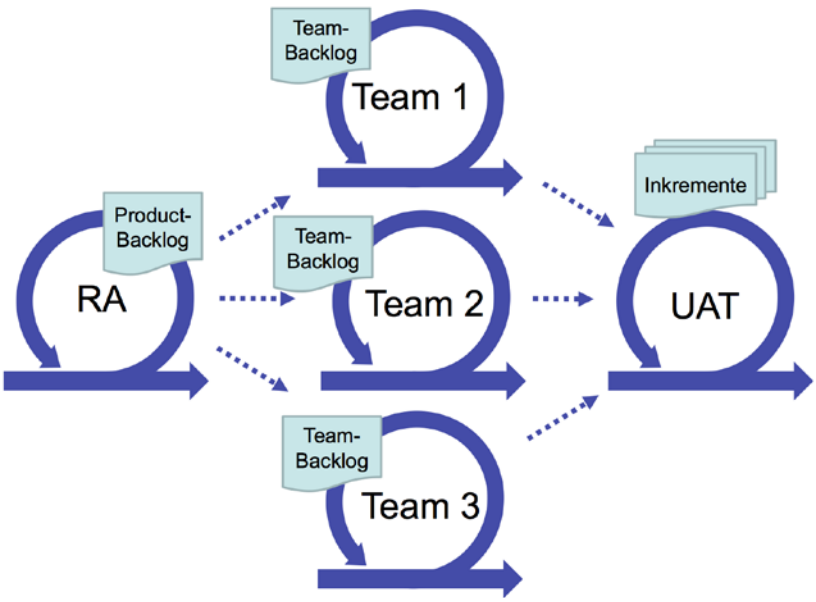

Abb. 1: Verteilung von Product-Backlogs auf mehrere Teams

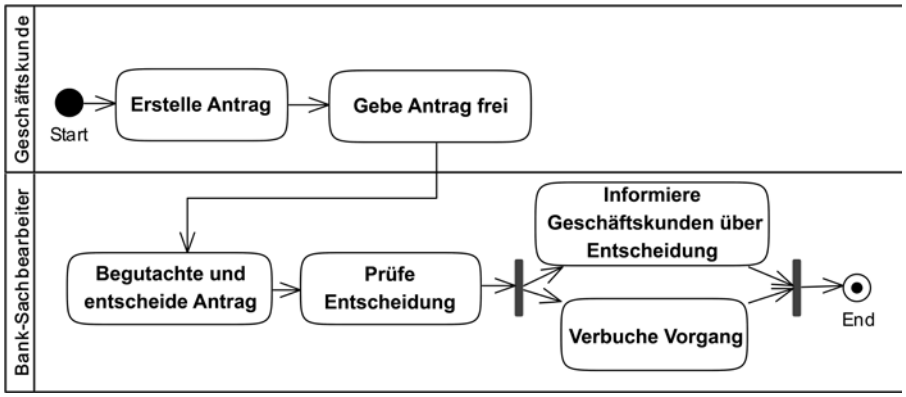

Abb. 2: Modellierung des Soll-Geschäftsprozesses „Antragsprozess“

unsere Erfahrungen mit der Aufgabenverteilung in verteilten, agilen Softwareprojekten berichten.

Lösungsansatz

Unser Lösungsansatz basiert auf unseren früheren Arbeiten zu Scrum, in denen eine dedizierte Phase für Anforderungsanalyse (RA - engl. Requirements Analysis) und Abnahmetests (UAT - engl. User Acceptance Test) explizit in Scrum verankert wurde [Gei12]. Dabei wird in der RA-Phase das Product-Backlog mit priorisierten funktionalen Anforderungen, aber auch mit nicht-funktionalen Anforderungen (Architektur-, Schnittstellen-, Performanz- und Sicherheitsanforderungen usw.) gefüllt. Anschließend werden die Entwicklungsaufgaben auf die unterschiedlichen Team-Backlogs verteilt.

Am Ende des Sprints müssen die Inkremente der Teams integriert und im Zusammenspiel getestet werden. Im Rahmen der iterativen und verteilten Entwicklung muss die Aufgabenbearbeitung überwacht und gesteuert werden (siehe Abbildung 1).

Analyse und Konzeption

In der RA-Phase werden als Erstes Anforderungen an das Softwaresystem systematisch erhoben und Systemkomponenten zugeordnet. Wir machen uns hier Techniken der objektorientierten Analyse (OOAD) mit der Unified Modeling Language (www.omg.org/spec/UML) zu Nutze. In unserem Ansatz modellieren wir ausgehend von Geschäftszielen die Soll-Geschäftsprozesse. Anschließend analysieren wir die grobe Soll-Architektur der Komponenten des Softwaresystems mit seinen Systemgrenzen und Schnittstellen. Im Weiteren ordnen wir dann Schritte der Geschäftsprozesse den System-Komponenten zu und formulieren Anforderungen an die Komponenten zur Erfüllung der Geschäftsprozessschritte.

Im Folgenden verdeutlichen wir unser Vorgehen an einem durchgängigen Beispiel aus der Finanzbranche.

Geschäftsziele und Geschäftsprozesse

In unserem Beispiel soll eine Online-Antragsplattform für Geschäftskunden einer Bank basierend auf einem Portalframework entwickelt und in die bestehende Systemlandschaft integriert werden. Beispielhaft formulieren wir die folgenden Geschäftsziele:

- Geschäftskunden sollen einen Finanzierungsantrag online einreichen können.
- Geschäftskunden sollen ihr Reporting online abrufen können.
- Anträge und Reporting sollen rund um die Uhr (24/7) gestellt beziehungsweise abgerufen werden können.

Mit Hinblick auf die Geschäftsziele und die bestehenden Ist-Prozesse werden Soll-Geschäftsprozesse gestaltet. In unserem Beispiel betrachten wir den zukünftigen Prozess, um einen Finanzierungsantrag einzureichen und um über diesen zu entscheiden.

In Abbildung 2 haben wir die Darstellung aus Platzgründen etwas vereinfacht und stellen nur den Gutfall dar. Der Geschäftskunde stellt online seinen Antrag. Ein Sachbearbeiter kann diesen prüfen und anschließend freigeben. Anschließend wird der Kunde über die Entscheidung informiert und parallel der Vorgang verbucht.

Soll-Architektur der Komponenten

Bei der Erstellung der Soll-Architektur modellieren wir die wichtigsten Komponenten des Systems mit ihren Schnittstellen. In unserem Beispiel ist das Backoffice-System nicht rund um die Uhr verfügbar. Deshalb wird gemäß dem Geschäftsziel eine Pufferungskomponente geplant, die Anfragen des Portals 24/7 entgegennimmt und das Reporting 24/7 verfügbar macht. Die Soll-Architektur entsteht im Zusammenspiel mit der Erstellung der Soll-Prozesse, da den Schritten des Soll-Prozesses die Komponenten zugewiesen werden, die diese Schritte realisieren sollen. So sollen die Aktivitäten der Geschäftskunden in unserem Beispiel durch das Geschäftskundenportal abgedeckt werden.

Abbildung 3 stellt die Systemarchitektur einschließlich bestehender Systeme (z. B. Reporting, Backoffice) und neu zu entwickelnder Komponenten (z. B. Geschäftskundenportal, Pufferungskomponente) dar. Ebenfalls dargestellt sind die Schnittstellen (z. B. AnträgeAbrufen, ReportingPuffern), die sich aus der Verteilung von Geschäftsprozessschritten auf Systemkomponenten ergeben. Wenn auch nicht ausdetailliert, so werden die Schnittstellen möglichst vollständig aufgezählt. Auch die Informationsobjekte werden benannt, die über diese ausgetauscht werden sollen. Die Systemgrenze (die dunkel gefärbten Komponenten) gibt an, welche Teile des Softwaresystems durch Projekt-

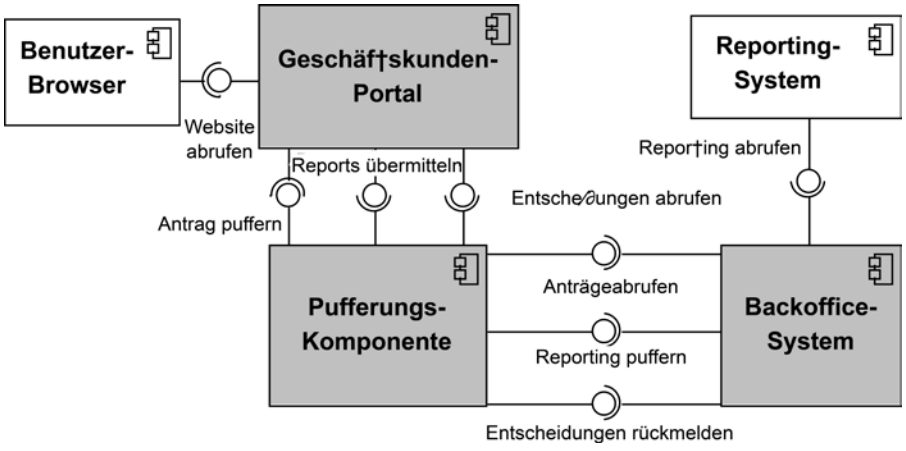

Abb. 3: Entwurf der System-Komponenten und ihrer Schnittstellen

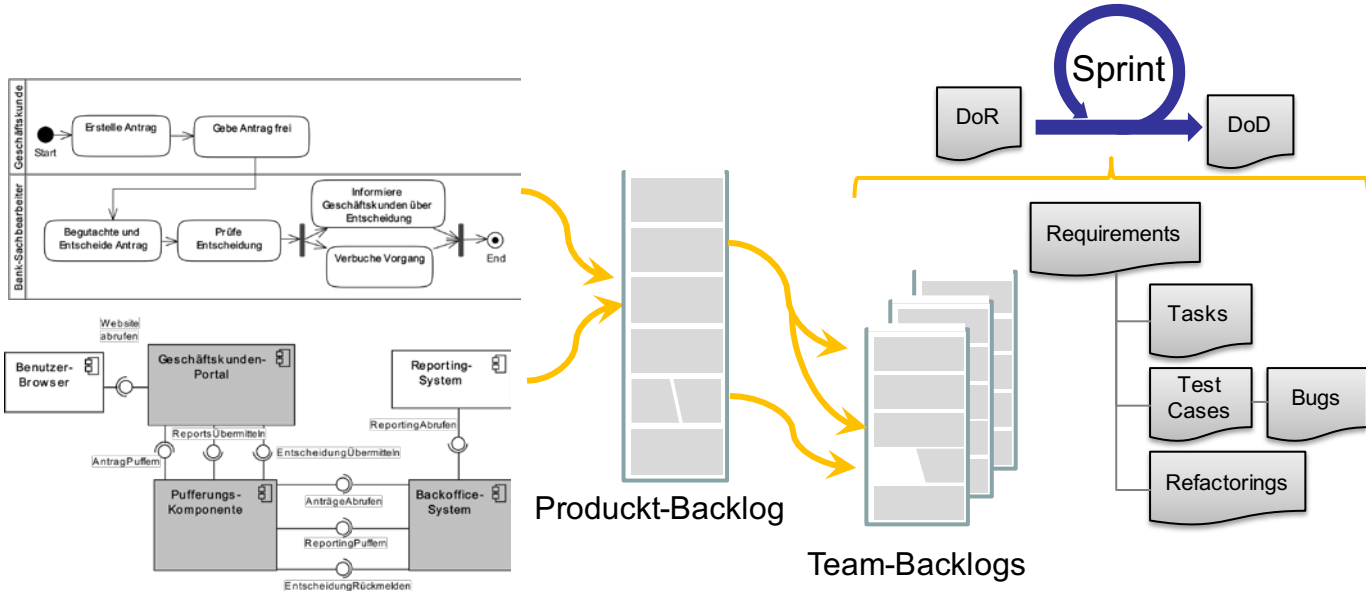

Abb. 4: Inhalte eines verteilten Team-Backlogs (angelehnt an die SAFe-Methode)

mitarbeiter beziehungsweise den Verantwortungsbereich des Projektes abgedeckt werden.

Ableitung von Anforderungen

Aus der Zuordnung von Geschäftsprozessschritten zu Komponenten und der Skizzierung von Schnittstellen und Informationsobjekten lassen sich beispielweise die folgenden Anforderungen an die Komponenten formulieren und in das Product-Backlog einfügen.

Anforderungen an das Geschäftskunden-Portal:

- „Als angemeldeter Geschäftskunde möchte ich Antragsdaten eingeben können, um einen Finanzierungsantrag an die Bank vorzubereiten".
- „Als angemeldeter Geschäftskunde möchte ich vorbereitete Antragsdaten freigeben können, um einen Finanzierungsantrag an die Bank zu stellen".

Anforderungen an die Pufferungskomponente:

- Die Pufferungskomponente muss Anträge 24/7 von dem Geschäftskundenportal entgegennehmen und puffern.
- Die Pufferungskomponente muss von dem Reporting-System übermittelte Reports an das Geschäftskundenportal weiterleiten.

Anforderungen an das Backoffice-System:

- Das Backoffice-System muss neue Anträge von der Pufferungskomponente periodisch abrufen.
- Das Backoffice-System muss neue Reports an die Pufferungskomponente übermitteln.

Verteilung

Auf Grundlage der RA-Phase verteilen wir die Aufgaben auf die Team-Backlogs nach

- fachlichen und technischen Kompetenzen der Teams,
- Entwicklungshoheit für die Systemkomponenten und
- Integrationshoheit für die Systemkomponenten.

Die Team-Backlogs müssen jeweils Backlog-Items zu verfeinerten Anforderungen (Requirements) aus dem Product-Backlog enthalten. Sie enthalten außerdem die von diesen Requirements abgeleiteten Entwicklungstätigkeiten (ToDos) und Testaktivitäten (Test Cases) (siehe Abbildung 4). Dazu kommen noch nach jedem Sprint Fehler (Bugs) und Verbesserungswünsche (Refactorings), die ebenfalls zu Requirements zugeordnet werden, falls möglich.

Wir erhalten „ein Netz" von verbundenen Elementen, das vom Product-Backlog über die verfeinerten Requirements in den Team-Backlogs bis zu ToDos und Test Cases reicht. Der Anteil der Backlog-Items in jedem Sprint ist Projekt- und Team-spezifisch zu bestimmen. In unserem Beispiel werden die in Tabelle 1 gezeigten Teams und Verantwortlichkeiten abgeleitet.

Überwachung und Steuerung

Um die Abstimmung der Vielzahl von Entwicklungs- und Testaktivitäten zwischen den Teams zu ermöglichen, sind spezielle „Definition of Ready"- (DoR) und „Definition of Done"-Kriterien (DoD) innerhalb von Requirements zu definieren. DoR-Kriterien definieren, welche Voraussetzungen erfüllt sein müssen, damit ein Team mit der Entwicklung eines Requirements starten kann (siehe [Be15]). Beispiele für DoR-Kriterien sind:

- Die technische Lösungsidee ist skizziert und entsprechende ToDos und Test Cases sind definiert.
- Die notwendige Schnittstelle „ReportingPuffern" ist bereits realisiert (entsprechende Anforderung erfüllt DoD-Kriterien).
- Die Lösungsidee ist mit den Fachabteilungen abgestimmt.

Ähnlich wie die DoR-Kriterien definieren die DoD-Kriterien die Nachbedingungen für die Realisierung einer Anforderung, damit diese als erledigt gilt. Beispiele für DoD-Kriterien sind:

- Die zugehörigen Test Cases wurden erfolgreich bestanden.
- Alle zugeordneten schwerwiegenden Fehler wurden behoben.
- Die Code-Qualität und die Einhaltung von nicht-funktionalen Anforderungen sind zufriedenstellend (z. B. Anzeige in unterstützten Browsern).

Die Erfüllung von DoR- und DoD-Kriterien beeinflusst den Start und Abschluss aller Entwicklungs- und Testaktivitäten. Requirements können nur in einen Sprint hineingenommen werden, wenn die entsprechenden DoR-Kriterien erfüllt sind.

Sie gelten erst als erfüllt, wenn die entsprechenden DoD-Kriterien erfüllt sind. Dabei gilt ein übergeordnetes Requirement aus dem Product-Backlog frühestens dann als erfüllt, wenn alle abgeleiteten Requirements erfüllt sind. Mit Anwendung der DoR- und DoD-Kriterien wird die folgende Arbeitsweise der verteilten Teams möglich:

- Jedes Team kann (zeitlich, örtlich) unabhängig von anderen Teams die Abarbeitung von Backlog-Items im agilen Verfahren vorantreiben. Jedes Team kann voraussetzen, dass benötigte Inhalte und Funktionen rechtzeitig da sind, wenn ein Sprint startet (DoR). Jedes Team muss sicherstellen, dass die Erwartungshaltung der anderen Teams an eigene Funktionen erfüllt ist (DoD). Zur Synchronisation werden eine DoR- und DoD-Checkliste jeweils vor dem Beginn und nach dem Abschluss eines Sprints unter Teams verteilt.
- Der Projektleiter kann den Fortschritt der Teams und den Erfüllungsgrad der DoD- und DoR-Kriterien teilautomatisiert überwachen, um Ursachen für eventuelle Verzögerungen oder Flaschenhälse im Prozess zu erkennen (z. B. offene ToDos, offene Anforderungen oder bestehende Fehler) und entsprechende Änderungen in Team-Backlogs rechtzeitig anstoßen. Als Mittel zur Überwachung setzen wir Status-Übersichten für Backlog-Items, Burndown-Charts und Velocity-Diagramme ein. Die Verknüpfung zwischen den Backlog-Items ermöglicht eine automatisierte Aussage über den Status der Requirements und führt damit zu einer manuellen Auswertung der DoD- und DoR-Kriterien.

Zusammenfassung

Wir haben einen Ansatz zur Verteilung und zur Koordinierung der Entwicklungs-Aufgaben in größeren Entwicklungsprojekten mit verteilten Teams vorgestellt. Bei der Verteilung von Aufgaben auf mehrere Teams kommt es darauf an, die fachlichen und technischen Kompetenzen der Teams zu berücksichtigen und technische Schnittstellen zwischen den zu entwickelnden Komponenten zu definieren. Definieren von organisatorischen Schnittstellen zwischen den Entwicklungsaktivitäten mittels DoR- und DoD-Kriterien gibt einerseits den Teams die Freiheit, die das agile Vorgehen erfordert, andererseits ermöglicht es dem Projektleiter die Überwachung und die Koordination der Teams.

Team 1	-Entwicklung des Geschäftsportals mit der Portaltechnologie -Schnittstellen zur Pufferungskomponente und zum Browser
Team 2	-Änderungen/Anpassungen an die vorhandenen Backoffice- und Reporting-Systeme -Schnittstellen des Backoffice-Systems zur Pufferungskomponente
Team 3	-Entwicklung der Pufferungskomponente -Schnittstellen der Pufferungskomponente zu den benachbarten Komponenten -Integration und Deployment der Systemkomponenten auf Test- und Abnahmeumgebungen -Einhaltung der nicht-funktionalen Anforderungen, wie Performanz und Sicherheit

Tabelle 1: Teams und Verantwortlichkeiten

In unserem Beispielprojekt mit einem Offshore-Partner in Indien konnten wir diesen Ansatz erproben und vom Erfolg in der Beherrschung der Projektkomplexität berichten. Dass bei solchen internationalen Projekten auch die interkulturelle Kompetenz in der Kommunikation und Problembehandlung eine wichtige Rolle spielt, könnte das Thema eines weiteren Artikels sein. ■

Literatur & Links

[Be15] J. Bergsmann, „Definition of Ready“ für agile Requirements, in: OBJEKTspektrum, Ausgabe Requirements Engineering/2015, siehe: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2015/Requirements_Engineering/bergsmann_OTS__Requirement_15.pdf

[Gei12] S. Geisen, B. Güldali, Agiles Testen in Scrum – Testtypen und Abläufe, in: OBJEKTspektrum, Ausgabe Agility/2012, siehe: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2012/Agility/geisen_gueldali_OS_Agility_2012_8e65.pdf

[Jan15] Th. Janning, On the SAFe way to Agility, Teil 1 und 2, in: OBJEKTspektrum, 2/2015 und 3/2015, siehe: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2015/02/janning_OS_02_15_2eE8.pdf und http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2015/03/janning_OS_03_2015_qOar.pdf

[La05] C. Larman, B. Vodde, Large scale scrum (LeSS), 2005-2015, siehe: http://less.works

[Le11] D. Leffingwell et al, Scaled Agile Framework (SAFe), 2011, siehe: http://www.scaledagileframework.com

[SuSch13] J. Sutherland, K. Schwaber, The Scrum Guide, 2013, siehe: http://www.scrum-guides.org/docs/scrumguide/v1/scrum-guide-us.pdf oder http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf