Martin Engstler et al. (Hrsg.): Projektmanagement und Vorgehensmodelle 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 109

Ganzheitliches Qualitätsmanagement in agilen Groß-Projekten

Masud Fazal-Baqaie[1], Baris Güldali[2] und Marvin Grieger[3]

Abstract: Agile Projekte sind populär und versprechen zeitnah lauffähige Zwischenstände von Software. Allerdings stellen agile Ansätze eine besondere Herausforderung für die Qualitätssicherung und das Testen dar. Gerade komplexe Geschäftsanwendungen in heterogenen Landschaften und mit verteilter Entwicklung erfordern ein Qualitätsmanagement, das die notwendige Transparenz schafft, aber die Entwicklung nicht ausbremst. In diesem Praxisbeitrag illustrieren wir an einem Fallbeispiel aus dem Bankenumfeld, die Einführung eines übergreifenden Qualitätsmanagements im Kontext von agilem Multiprojektmanagement und verteilter Offshore-Entwicklung.

Keywords: Software-Qualitätsmanagement, Development Governance, Test-Management, Agile Entwicklung, Scrum, Global Software Development

1 Einleitung

In vielen Bereichen werden klassische, schwergewichtige Vorgehensmodelle der Softwareentwicklung durch leichtgewichtige, agile Methoden [BS01], [Me14] abgelöst. Die Gründe dafür sind vielfältig. Zum einen erlaubt eine agile Entwicklung eine schnellere Anpassung an regulatorische Anforderungen bzw. den Markt, zum anderen können Kundenbedürfnisse besser berücksichtigt werden. Dies wird insbesondere durch den verringerten formalen Aufwand agiler Methoden ermöglicht, sowie durch verkürzte Entwicklungszyklen und dadurch verstärktes Kundenfeedback. Damit werden agile Methoden als Chance verstanden um angemessen auf aktuelle Herausforderungen wie z.B. die drängende Digitalisierung von Geschäfts- und Produktionsprozessen zu reagieren. In der Tat helfen agile Ansätze den Beteiligten bei der Entwicklung kundenzentrierter Lösungen, da bereits nach den ersten Sprints sichtbare Ergebnisse entstehen. Diese ermöglichen Verständnislücken frühzeitig aufzudecken und dadurch die Erwartungshaltung zu präzisieren.

Jedoch ist die Einführung agiler Entwicklungsmethoden nicht immer problemlos möglich, was wir in diesem Beitrag an einem Fallbeispiel aus dem Bankenumfeld demonstrieren. Eine große Hürde ist dort die Aufteilung in organisatorische Silos [Ht12, S.19], d.h. eigenständige Abteilungen mit eigens zugeordneten IT-Systemen. So existieren Fachabteilungen (und je Applikation ein Business Owner), IT-Entwicklungsabteilungen (und je Applikation ein Asset Owner) und Abteilungen zuständig für den IT-Betrieb. Eine optimale Berücksichtigung von Kundenbedürfnissen erfordert den Aufbruch dieser Silos, so dass eine bereichs- und systemübergreifende Softwareentwicklung erfolgt. Erst dann ist eine echte Kundenzentrierung möglich, weg von der Entwicklung separierter Applikationen, hin zu einer integrierten, kundenorientierten Lösung, wie sie die Digitalisierung, z.B. in der Bankenbranche, erfordert.

[1] S&N CQM GmbH, Klingenderstr. 5, 33100 Paderborn, masud.fazal-baqaie@sn-cqm.de
[2] S&N CQM GmbH, Klingenderstr. 5, 33100 Paderborn, baris.gueldali@sn-cqm.de
[3] s-lab – Software Quality Lab, Zukunftsmeile 1, 33102 Paderborn, marvin.grieger@s-lab.upb.de

Als eine weitere große Hürde, stellen agile Methoden vor allem auch das klassische Qualitätsmanagement (QM) [Ga03] vor neue Herausforderungen. Im Allgemeinen können Sachverhalte nicht mehr vorab vollständig geklärt werden, da Spezifikationen inkrementell erarbeitet und weiterverarbeitet werden [FGS15]. Folglich müssen die Qualitätskriterien angepasst werden: eine dokumentenzentrierte Qualitätssicherung muss auf eine Feature- oder Story-basierte Variante umgestellt werden. In einem Multiprojektmanagement-Umfeld, wie in unserer Fallstudie, ist darüber hinaus die Vergleichbarkeit von Reports notwendig, damit diese zu einem aussagekräftigen Gesamt-Reporting aggregiert werden können. Um Transparenz für das Management der verschiedenen Projekte zu erreichen, muss deshalb dafür gesorgt werden, dass Qualitätssicherungsmaßnahmen und -reports projektübergreifend vergleichbar sind.

In diesem Beitrag illustrieren wir an einem Fallbeispiel zur Digitalisierung von B2B- und B2C-Geschäftsprozessen aus dem Bankenumfeld, die Einführung eines übergreifenden Qualitätsmanagements im Kontext von Multiprojektmanagement und agiler, verteilter Offshore-Entwicklung. Dabei umfasst das Qualitätsmanagement sowohl Aktivitäten der Fachabteilungen als auch der IT-Entwicklungs- und IT-Betriebsabteilungen (intern wie extern) und deckt den gesamten Softwarelebenszyklus von den Anforderungen, über die Entwicklung, den Abnahmetest und den Betrieb ab. Kernbestandteile unserer Qualitätssicherung ist ein Mix aus verschiedenen Maßnahmen und die Messung ihrer Anwendung über Kennzahlen, welche spezifisch auf die Anforderungen agiler Ansätze zugeschnitten sind. Durch ein vereinheitlichtes Reporting über Dashboards wird eine projektübergreifende Übersicht und Vergleichbarkeit erreicht. Die Qualitätssicherungsmaßnahmen und zugehörigen Kennzahlen wurden über ein Reifegradmodell schrittweise in den zugehörigen Unterprojekten eingeführt, um den individuellen Reifegraden zu Beginn Rechnung zu tragen. Da es sich um ein laufendes Projekt handelt, berichten wir von dem aktuellen Stand und bisherigen Erkenntnissen.

Diese Arbeit beinhaltet drei Kernbeiträge. Zunächst schildern wir in Kapitel 2 den Projektkontext und identifizieren die resultierenden Herausforderungen (I) für das übergreifende Qualitätsmanagement von agil und verteilt entwickelten, komplexen Applikationen. Darauf aufbauend beschreiben wir in Kapitel 3 den von uns entwickelten Lösungsansatz (II) für das übergreifende Qualitätsmanagement, welcher die identifizierten Herausforderungen berücksichtigt. Abschließend berichten wir von der schrittweisen Einführung (III) des Lösungsansatzes über ein Reifegradmodell und diskutieren die bisherigen gewonnenen Erkenntnisse. In Kapitel 4 fassen wir unsere Erkenntnisse zusammen.

2 Projektkontext und Problemstellung

In diesem Abschnitt beleuchten wir den Hintergrund zu unserer Fallstudie. Wie erläutern in Abschnitt 2.1 den Projektkontext, d.h. das eigentliche Projektziel sowie die Projektorganisation. Anschließend gehen wir in Abschnitt 2.2 auf die bisherige Entwicklungsmethode und das zugehörige Qualitätsmanagement im Detail ein. In Abschnitt 2.3 diskutieren wir die Probleme, die aus der Kombination von klassischem Qualitätsmanagement und agilem Entwicklungsansatz resultieren.

2.1 Projektziel und Projektorganisation

Das Ziel des betrachteten Projekts ist die Einführung eines kundenzentrierten, digitalen Vertriebswegs für die Geschäftspartner und Endkunden einer Bank. Die Anbindung solcher Vertriebswege erforderte auf der technischen Ebene vor allem eine Öffnung und Anpassung existierender Backend-Systeme über webbasierte Schnittstellen für Geschäftspartner und Endkunden. Gleichzeitig müssen bestehende fachliche Geschäftsprozesse abteilungsübergreifend angepasst werden. Aufgrund der umfangreichen Änderungen wurde ein agiler Ansatz für dieses Unterfangen gewählt, welcher die Entwicklung in einzelne, koordinierte Sprints unterteilt.

Organisatorisch ist das Projekt in Teilprojekte zerlegt, die Abhängigkeiten untereinander besitzen. Gleichzeit sind die Teilprojekte in sich auch geographisch und organisatorisch heterogen. So wurden die reine Entwicklungsarbeit inkl. die Entwicklertests an einen Offshore-Entwicklungspartner ausgegliedert. Die Erfassung von Anforderungen, die Durchführung von Abnahmetests und der eigentliche Betrieb wird hingegen Onshore durchgeführt, wobei auch hier verschiedene Zulieferer involviert sind.

2.2 Entwicklungskontext und bisheriges Qualitätsmanagement

In der Bank folgte der bisherige Application Lifecycle [Ch08] eher einem klassischen Wasserfall-Ansatz, auch wenn die Entwicklung teilweise bereits agile Ansätze adaptierte. Es waren eher lange Zyklen mit lediglich ein oder zwei Releases pro Jahr je Applikation üblich.

Die Fachspezifikationen wurden im Vorfeld erstellt. Die Entwicklung selbst erfolgte dann entweder in einer großen Iteration oder in manchen Projekten auch iterativ in Sprints, einem eigenen Ansatz basierend auf Scrum und Prince2 folgend. Oft wurde auch das Entwicklungsmodell beteiligter Zulieferer übernommen. Der IT-Betrieb war klassisch nach ITIL organisiert mit den besagten ein bis zwei Releases pro Jahr.

Die Qualitätssicherung basierte bisher im Wesentlichen auf analytischen Maßnahmen [SL03, Sc12], die an diese langen Zeitfenster angepasst waren. So wurden Fachspezifikationen vor deren Umsetzung durch Entwickler und andere Fachabteilungen u.a. auf Vollständigkeit und Korrektheit geprüft. Da die Entwicklung durch erfahrene Entwickler und langjährige Partner-Zulieferer erfolgte, waren Programmier-Richtlinien als auch Unit-Tests eher unüblich. Die entwickelte Software wurde abschließend von den Fachabteilungen in Abnahmetests getestet und abgenommen. Regressionstests fanden manuell statt, Testautomatisierung war im Allgemeinen kaum im Einsatz. Das Projekt-Reporting wurde projektspezifisch aufgesetzt. Jedes Projekt definierte sein eigenes Test-Reporting. Dieses war also nicht projektübergreifend vergleichbar.

Mit der Umstellung auf einen agilen Application Lifecycle in Kombination mit dem verteilten Multiprojektmanagement-Umfeld stand das bestehende Qualitätsmanagement vor großen Herausforderungen:

- Die kurzen Zyklen machte die Umstellung von Dokumenten-zentrierten Fachspezifikationen auf Artefakt-zentrierte Feature-Spezifikationen notwendig. Diese sind

per Definition nie vollständig im klassischen Sinne und genügen auch anderen klassischen Qualitätskriterien nicht.

- Die hohe Frequenz der kurzen Zyklen und die hohe Zahl an On- und Offshore-Zulieferern, zusammen mit dem Verzicht auf systematische Qualitätssicherung bei den Entwicklern, mindert die Qualität und Wartbarkeit der Software-Systeme
- Die hohe Frequenz der Zyklen machten die Abnahmetests zu aufwendig und langwierig
- Die kurzen Zyklen und vielen Teilprojekte mit ihrem heterogenen Test-Reporting machte Qualitätsaussagen fast unmöglich und nicht vergleichbar
- Die hohe Zahl an Releases überforderte den Prozess der manuellen Produktivstellung durch die Unterschiede zwischen Test-, Abnahme- und Produktivumgebungen

2.3 Problemstellung und Anforderungen an ein ganzheitliches und durchgängiges Qualitätsmanagement

Basierend auf den vorangehenden Beschreibungen der Projektziele und des Entwicklungskontexts möchten wir in diesem Abschnitt näher auf die Probleme eingehen, die bei einem Wechsel auf eine agile Vorgehensweise und in einem heterogenen Multiprojektmanagement-Umfeld entstehen. Im Folgenden wird jedes Problem kurz illustriert und eine Lösungsanforderung abgeleitet:

- **Bestehende Qualitätskriterien wenig aussagekräftig:** Während in der Vergangenheit schwergewichtige Spezifikationen vorherrschten, herrschen jetzt leichtgewichtige und ausschnitthafte Spezifikationen vor. Damit sind Kriterien wie „Vollständigkeit“, „Vereinzelte Anforderungen“ und „funktionale Dekomposition“ von Anforderungen nicht mehr ausreichend und teilweise irreführend. *Anforderung A1: Es sind auf Agilität ausgelegte Qualitätskriterien notwendig.*
- **Verzicht auf systematische Qualitätssicherung der Entwicklung:** Während in der Vergangenheit in den Projekten mit wenigen Partner-Zulieferern und erfahrenen Entwicklern gearbeitet wurde, kollaborieren jetzt viele verschiedene Entwickler auf unterschiedlichem Niveau. Gleichzeitig müssen sie in kurzen Zyklen ihre Ergebnisse zu korrekter, lauffähiger Software integrieren und es bleibt ihnen weniger Zeit Fehler zu korrigieren und damit Fehler bei der Abnahme zu vermeiden [FR15].
  Anforderung A2: Es sind konstruktive und analytische Qualitätssicherungsmaßnahmen für die Entwickler notwendig.
- **Aufwendige Abnahmetests:** Während in der Vergangenheit nur ein bis zwei Releases pro Jahr abgenommen werden mussten, werden Releases jetzt alle 2-3 Monate produktiv genommen. Gleichzeitig ist der Scope der betroffenen Systeme und Fachabteilungen sehr viel größer, während der Zeitraum um Probleme zu beheben („Hotfixes“) kleiner wird. Damit sind ausschließlich manuelle Abnahmetests nicht mehr zu bewältigen.
  Anforderung A3: Der Einsatz von Testautomatisierung ist notwendig.

- **Heterogenes Test-Reporting:** In der Vergangenheit hat jedes (Teil-)Projekt sein individuelles Reporting durchgeführt und an das Steering Comitee berichtet. Metriken waren nicht projektübergreifend definiert. In einem agilen Multiprojektmanagement-Umfeld fehlt bei diesem Ansatz die Vergleichbarkeit, wodurch eine Bewertung erschwert wird. Beispielsweise definiert jedes Teilprojekt seine eigenen Kriterien bezüglich „done" und „getestet". Die hohe Frequenz der Zyklen macht aber eine Vergleichbarkeit und schnelle Bewertung der Gesamt-Qualität notwendig.
  Anforderung A4: Das Test-Reporting muss aussagekräftig, vergleichbar und aggregierbar werden.

- **Manuelle Produktivstellung:** Während in der Vergangenheit nur ein bis zwei Releases pro Jahr abgenommen werden mussten, werden Releases jetzt alle 2-3 Monate produktiv genommen. Damit wirken sich die bestehenden Unterschiede zwischen den Entwicklungs-, Abnahme- und Testumgebungen und damit verbundenen Konfigurationsaufwände stärker aus. Gleichzeitig sind viele voneinander abhängige Systeme beteiligt.
  Anforderung A5: Die Unterschiede zwischen den Entwicklungs-, Abnahme- und Produktivumgebungen müssen verringert und deshalb durch das Qualitätsmanagement erfasst werden.

Darüber hinaus ergibt sich aus den webbasierten Schnittstellen zu Geschäftspartnern und Endkunden die Notwendigkeit, Aspekte wie Security, Usability und Performance viel weitreichender zu berücksichtigen (*Anforderung A6*). In der Vergangenheit wurden Applikationen zum aller größten Teil in abgesicherten Intranet-Netzen und von Mitarbeitern genutzt.

Gleichzeitig ist durch die Kollaboration vieler verschiedener Mitarbeiter aus verschiedenen On- und Offshore-Organisationen das Qualitätsniveau zwischen den verschiedenen Teilprojekten sehr unterschiedlich. Wir haben mit Offshore-Partnern die Erfahrung gemacht, dass die Arbeitskultur sich von dem unterscheidet, was man aus Onshore-Projekten kennt. Oft mussten wir deshalb für Offshore-Entwickler unsere Erwartungshaltung bei den Entwicklungsaktivitäten detaillierter spezifizieren. Deshalb werden Qualitätsverbesserungsmaßnahmen benötigt, welche die Teilprojekte nicht überfordern aber mittelfristig auf das gleiche Niveau anheben. Dabei ist zu beachten, dass dies eine Transparenz über den Grad der Reife einzelner Teilprojekte erfordert, d.h., belastbare Aussagen über deren Qualitätsniveau mittels eines Reifegradmodells (*Anforderung A7*).

Zusammengefasst wird ein übergreifendes, ganzheitliches Qualitätsmanagement benötigt, in welchem die verschiedenen Belange adressiert und aufeinander abgestimmt sind. Genauer gesagt sollen analytische und konstruktive Maßnahmen (*ganzheitlich*) im gesamten Softwarelebenszyklus (*übergreifend*) eingesetzt werden. Durch den Einbezug von unterschiedlichen externen Zulieferern erfordert das unter anderem die Erfassung von Programmier-Richtlinien, die Durchführung von Code-Analysen sowie die Einführung automatisierter Entwickler-, Integrations- und Akzeptanztests. Zudem muss eine Verbesserung der Vergleichbarkeit zwischen projektspezifischen Entwicklungs-, Test- und Abnahmeumgebungen erreicht werden. Diese Maßnahmen erleichtern letztendlich eine stärkere Zusammenarbeit und Integration von Entwicklung und Betrieb.

3 Lösungsansatz für ganzheitliches, agiles Qualitätsmanagement

Nachdem der Projektkontext mit seinen Herausforderungen dargestellt wurde, beschreiben wir in diesem Abschnitt unseren Lösungsansatz zur Erfüllung der im Abschnitt 2.3 aufgestellten Anforderungen. Zunächst beschreiben wir die einzelnen Lösungselemente unseres Ansatzes. Danach gehen wir auf seine schrittweise Einführung und die Lessons Learned ein.

Unser ganzheitliches Qualitätssicherungskonzept ist in dem *agilen Prozess* und die dafür notwendige *Werkzeuginfrastruktur* verankert (siehe Abbildung 1). Der *agile Prozess* sieht folgendermaßen aus: Parallel zu den eigentlichen Entwicklungssprints erfolgt eine kontinuierliche gemeinsame Makroplanung aller Subprojekte (links in der Abbildung) [GF15]. Mit ihr findet eine grobe Zuordnung von Entwicklungsthemen für die unterschiedlichen Teams sowie die Aufteilung und die Koordination der Entwicklungszwischenergebnisse auf die Sprint-Backlogs der Teams statt. Dabei werden die Aktivitäten für die Entwicklung (Develop), das Integrieren und das Testen (Build & Test) und die Produktivstellung und den Betrieb (Deploy & Operate) in kurzen Zyklen durchgeführt. In jeder Iteration werden eine Auswahl von Product-Backlog Items zusammen mit Change Requests und Defects aus der vorherigen Iteration bearbeitet, so dass zusätzliche bzw. verbesserte Softwarefunktionen entstehen. Während ein Teil des agilen Teams sich den Implementierungs- und Entwicklertestaufgaben widmet, werden parallel Abnahmetestaktivitäten von den Fachabteilungen geplant.

Die *Werkzeuginfrastruktur* beschreiben wir im Folgenden: Die Aktivitäten werden von einer Werkzeuginfrastruktur für Issue Tracking, Continuous Integration [HF10, S.55] und Continuous Delivery [HF10, S.105] unterstützt, mit denen eine kontinuierliche Überwachung und Steuerung von Aktivitäten verschiedener Teams und des Qualitätsstatus der entwickelten Softwarekomponenten möglich werden (oben in der Abbildung). Die Details der Werkzeuginfrastruktur werden im Folge-Abschnitt näher beschrieben.

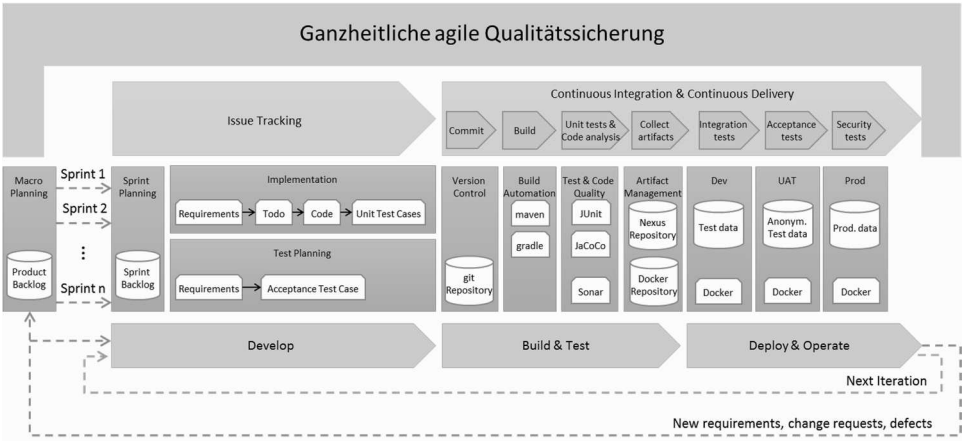

Abb. 1: Prozess zur agilen Entwicklung und ganzheitlichen Qualitätssicherung

3.1 Lösungselemente

In Abschnitt 2.3 haben wir die Anforderungen an ein ganzheitliches und durchgängiges Qualitätsmanagement skizziert. In unserem Qualitätsmanagement unterscheiden wir zunächst zwischen Produktqualität und Prozessqualität.

3.1.1 Produktqualität

Die Produktqualität unterteilt sich in interne und externe Produktqualität. Die interne Produktqualität macht sich während der Entwicklungsaktivitäten bei den Teammitgliedern durch eine bessere Wartbarkeit und Erweiterbarkeit des Programmcodes bemerkbar. Insbesondere adressiert sie die von uns formulierten Anforderungen A1, A2, A4, A5, A7. Die externe Produktqualität enthält alle Qualitätseigenschaften, die bei der Nutzung der Software durch die Endnutzer wahrgenommen werden. Dies umfasst z.B. die funktionale Korrektheit, die Benutzertauglichkeit oder Performanz-Eigenschaften und damit A3 und A6.

Folgende Maßnahmen, die wir in unserem Projekt eingeführt haben, sind Beispiele für die Verbesserung der internen Produktqualität:

- **Einheitliche, stufenweise Verfeinerung von Anforderungen:** Wir haben die Anforderungsformulierung vereinheitlicht und eine Struktur vorgegeben, die stufenweise verfeinert wird [FGS15]. Unsere Requirement-Artefakte enthalten im Product Backlog zunächst nur eine einfache User Story zu einem Feature. Im Laufe der Vorbereitung zu einem Sprint werden Requirements allerdings auch mit stichpunktartigen Kurzbeschreibungen z.B. zu Lösungsvarianten, Akzeptanzkriterien und Abhängigkeiten verfeinert. Insbesondere für die Offshore-Partner-Zulieferer sind die Kurzbeschreibungen und vorhergegangene Kommunikation dazu verpflichtend, bevor ein Requirement zu einem Sprint zugelassen wird. Anforderungsprozess und Struktur ersetzen Anforderungsdokumente und dazugehörige Qualitätsmetriken, die vorher bestanden. Diese Maßnahme adressiert vor allem Anforderung A1.
- **Einheitliche Code-Strukturen und Kodierungs-Richtlinien:** Die Auswahl von Build-Technologien geben die Code-Strukturen vor. Durch den Einsatz von einheitlichen Build-Tools, wie *maven* oder *gradle*, und durch die Vorgabe von einheitlichen Code-Strukturen zur Konfiguration und Parametrisierung von Quellcode werden die Subprojekte vergleichbar und wartbar. In Folgeschritten bringt das weitere Vorteile durch einheitliches Kompilieren und Aufdecken von Kodierungsfehlern und Verletzung von Kodierungs-Richtlinien. Diese Maßnahmen adressieren vor allem Anforderung A2.
- **Testautomatisierung für Entwicklertests:** Entwickler erstellen neben dem Produktivcode auch Unit- und Integrations-Testfälle (z.B. mit *JUnit*), die das Ziel haben, Fehler im Code vor der Integration und Installation aufzudecken und Regressionsfehler bei zukünftigen Änderungen zu vermeiden. Diese Maßnahmen adressieren vor allem Anforderung A2.
- **Continuous Integration:** Das Einchecken von Code im Versionsmanagement stößt einheitliche und automatisierte Build-Pipelines an, bei denen der Code kompiliert, getestet und nach Einhaltung von Code- und Test-Qualitätsregeln geprüft wird. Geprüft werden beispielsweise Testabdeckungsmaße (mittels *JaCoCo*), Verletzung

von Sicherheitseigenschaften oder Code-Wiederholungen (mittels *SonarQube*). Im Erfolgsfall werden die ausführbaren, qualitätsgeprüften Artefakte in ein Repository (z.B. *Nexus*) gespeichert. Diese Maßnahmen adressieren vor allem Anforderung A2 und A5.

- **Umstieg auf Container-Technologien:** Durch den Einsatz von Container-Technologien (z.B. *Docker*) werden die Software-Artefakte aus dem Artefakt-Repository je nach den Anforderungen der verschiedenen Integrationsumgebungen zusammengestellt und automatisiert installiert. Dadurch werden die Entwicklungs-, Abnahme- und Produktionsumgebung homogenisiert und die Installationsschritte beschleunigt. Diese Maßnahmen adressieren vor Allem Anforderung A5.

Die Software kann auf der Produktionsumgebung zur Nutzung durch die Endnutzer installiert werden, nachdem sie auf der Entwicklungs- und Abnahmeumgebungen von Entwicklungsteam und der Fachabteilung ausführlich getestet und freigegeben wurde. Die von Endnutzern wahrnehmbare „externe" Produktqualität prüfen wir durch folgende Maßnahmen:

- **Funktionale Abnahmetests:** Zusätzlich zu den Entwicklertests muss die Software aus Endnutzer-Sicht auf funktionale Korrektheit getestet werden. Auf Grundlage der Backlog-Requirements werden für die einzelnen Systemkomponenten funktionale Tests und für das Zusammenspiel der Komponenten End-to-End Tests durchgeführt. In der agilen Entwicklung kommt es darauf an, eine repräsentative Menge an Regressionstestfällen zur Wiederholung bei neuen Inkrementen zu definieren und möglichst viele dieser Testfälle zu automatisieren (z.B. mittels *Selenium* oder vergleichbare UI-Testtechnologien). Diese Maßnahmen adressieren vor Allem Anforderung A3.
- **Usability Tests:** Neben der fachlichen Korrektheit der Funktionen ist ihre Benutzertauglichkeit auch von großer Bedeutung. Die Software muss ohne langwierige Einführungen intuitiv und selbsterklärend nutzbar sein. In einer dedizierten Usability-Test Phase wird die Software von Beta-Testern eingesetzt und Verbesserungsbedarf sowie –wünsche aufgenommen. Diese Maßnahmen adressieren vor Allem Anforderung A6.
- **Lasttests und Performance-Monitoring:** Die Software muss auf die variierende Anzahl von gleichzeitigen Nutzern zu unterschiedlichen Zeiten eines Tages oder einer Arbeitswoche robust reagieren und keine langen Reaktionszeiten verursachen. Wir simulieren mittels Lasttest-Werkzeugen (z.B. *JMeter*) unterschiedliche Nutzerlasten und messen die Performanzeigenschaften der Software (z.B. CPU- und Speichernutzung). Diese Maßnahmen adressieren vor Allem Anforderung A6.
- **Security Tests:** Die Software ist öffentlich über das Internet verfügbar und muss Cyberattacken standhalten. Sonst drohen Ausfälle und Verluste von sensiblen Daten. Um Sicherheitslücken im Vorfeld aufzudecken führen wir statische Codeanalysen (z.B. mit *OWASP Plugins* in *Jenkins* und *SonarQube*) und dynamische Penetrationstests durch. Für das letztere sind zwar Community und Enterprise-Werkzeuge (z.B. *ZAP Tool* und *HPE Quality Center*) vorhanden, allerdings haben wir gute Erfahrung in der Zusammenarbeit mit Sicherheitsexperten gemacht, die zwar höhere Kosten verursachen, aber eine individuelle Betreuung ermöglichen. Diese Maßnahmen adressieren vor Allem Anforderung A6.

Um die Qualitätssicherungsmaßahmen zu überwachen und sie zu optimieren, haben wir Produkt-Metriken eingeführt, die wir regelmäßig messen und im Team diskutieren. Folgende Produkt-Metriken werden für die Produkt-Qualität verwendet:

- **Technical dept:** Mittels SonarQube erfassen wir die Verletzung der Kodierungsrichtlinien. Für jede Regel-Verletzung wird ein hypothetischer Aufwand berechnet, der notwendig wäre um diese Verletzung zu beheben. Die Teammitglieder versuchen bei ihren Code-Einreichungen möglichst viele Verletzungen zu beheben und keine neue zu verursachen. Eine entscheidende Maßnahme war, den Entwicklern vorzuschreiben, Benachrichtigungen für Verletzungen zu aktivieren. Somit wurde eine höhere Sensibilisierung für qualitativ bessere Code-Einreichungen erreicht.
- **Unit tests and code coverage:** Die Entwickler haben die Vorgabe, für jede neue Java-Klasse ausreichende Unit-Testfälle zu erstellen. Ausreichend heißt in unserem Fall, dass die Testfälle in der Summe eine Testabdeckung von min. 70% Anweisungsüberdeckung erreichen sollen. Dabei exkludieren wir den von unserem Webframework generierten Code von der Analyse und fokussieren uns auf den manuell programmierten Businesscode.
- **Code duplications:** Insbesondere in Offshore-Projekten haben wir die Erfahrung gemacht, dass Entwickler Codeteile oftmals kopieren und mehrfach verwenden. Um dem entgegenzuwirken, erwarten wir von Subprojekten eine hohe Modularisierung und eine Reduzierung von Codeduplikaten auf max. 5%. Dadurch wird die Wartbarkeit und damit Reaktionsgeschwindigkeit bei Änderungen und Bugfixing erhöht.
- **Functional quality:** Wir setzen die Anzahl von fehlgeschlagenen Testfällen zu ausgeführten Testfällen in Relation. Insbesondere bei kritischen Regressionstestfällen setzen wir nahezu eine 100%-ige Erfolgsquote voraus. Allerdings hängt die Freigabeempfehlung der Fachabteilung vom Typ des Subprojekts ab. Intranet-Applikationen müssen z.B. nicht das gleiche Qualitätsniveau aufweisen wie Internet-Applikationen.
- **Test quality:** Diese Metrik zeigt, wie ausführlich die Tests sind. Die oben definierte Functional-quality-Metrik setzt voraus, dass die Implementierungsergebnisse gegen die Anforderungen durch ausreichend Testfälle geprüft werden. Während der Anforderungsverfeinerung werden Abnahmekriterien für jede Anforderung definiert. Für jedes Kriterium müssen zwei oder mehr Testfälle erstellt werden, die neben den Standardfällen auch Alternativ- und Negativszenarien abdecken. Die Relation von Testfällen zu Abnahmekriterien ergibt die Test-quality-Metrik.

3.2 Prozessqualität

Neben dem Qualitätsmanagement für die Produktqualität haben wir auch großen Wert auf die Prozessqualität gelegt. Insbesondere die Aktivitäten der Fachabteilung, welche die Anforderungen definieren und die fachlichen Abnahmetests durchführen, werden mittels agilen Projektmagement-Techniken koordiniert und gesteuert. Für die Fortschrittsüberwachung haben wir Flowcharts und Burndown-Charts eingesetzt, welche die Fortschritte der erstellten Anforderungs- und Testtickets pro Sprint in Korrelation zu Entwickler-Todos und gefundenen Fehlern und den Umfang von Restarbeiten visualisieren (s. Abbildung 2).

Um die Subprojekte miteinander vergleichbar zu machen, und einen Gesamteindruck von Projektfortschritt und Produktqualität zu gewinnen, und damit der Anforderung A4 gerecht zu werden, erstellten wir aggregierte Berichte für alle Subprojekte. Dabei konnten wir feststellen, dass die Geschwindigkeit und Homogenität der Ticketverarbeitung stark variiert. Während Subprojekt 1 einen kontinuierlichen Fortschritt in der Erstellung von Requirements und Test Cases aufweist und bereits frühzeitig vorhandene Fehler aufdeckt (vgl. Abbildung 2, oben), erreicht das Subprojekt 2 trotz der steigenden Anzahl der Anforderungen keinen ausreichenden Testumfang und damit keine ausreichende Fehleraufdeckung. Auch die Abarbeitung von Anforderungen und Entwicklungsaufgaben zeigt, dass in Subprojekt 2 keine effektive und homogene Geschwindigkeit zu erkennen ist (vgl. Abbildung 2, unten).

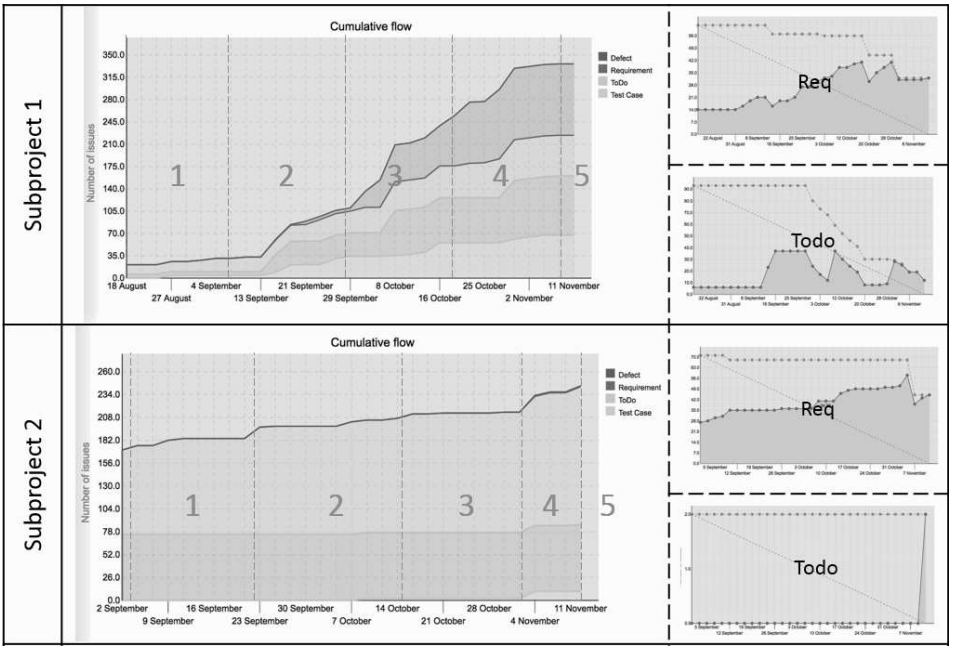

Abb. 2: Fortschrittsüberwachung mittels Flowcharts und Burndowns

Die Fortschrittsüberwachung hat insb. das Projektmanagement mit wertvollen Informationen beliefert, auf deren Grundlage Optimierungsmaßnamen eingeleitet wurden, bevor die Prozessdefizite zu Projektrisiken wurden.

3.3 Stufenweise Einführung

Da die Subprojekte unterschiedliche Ausgangssituationen haben und die Fertigkeiten zwischen Onshore- und Offshore-Teams stark variieren, haben wir eine stufenweise Einführung der Qualitätsmaßnahmen festgelegt (s. auch Anforderung A7). Dafür haben wir zusammen mit Teammitgliedern Maturity Levels definiert, die einen zunehmenden Umfang an Kennzahlen bei den Metriken vorsehen. Die Teams setzen sich eigene Ziele und legen ein Datum zur Erreichung dieser Ziele fest. Dadurch werden Teams motiviert, mit kleinen Schritten anzufangen und sich kontinuierlich zu verbessern. Tabelle 1 zeigt eine Auswahl

an Qualitätsmaßnahmen und den notwendigen Mindest-Qualitätsstatus, der erreicht werden muss, um einen Reifegrad bescheinigt zu bekommen. Viele dieser Metriken lassen sich in der Continuous-Integration-Umgebung automatisiert messen und darstellen, so dass der Qualitäts- und Reifegrad-Status für Projektteilnehmer transparent ist.

Maturity Level for internal Quality					
	1	**2**	**3**	**4**	**5**
Code resides in Git Repository	yes	yes	yes	yes	yes
Build automation	yes	yes	yes	yes	yes
Unit tests automated	-	yes	yes	yes	yes
Unit tests passed	-	-	100%	100%	100%
Code coverage enabled/measured	-	measurable	measured min. 5%	measured min. 50%	measured min. 70%
SonarCube rules	-	measurable	measurable	No blocker issues	No blocker and critical and major issues
Nexus deploy	-	yes	yes	yes	yes
…					
Subproject 1	x	x	x	Due Dec 2016	Due Feb 2017
Subproject 2	x	x	Due Oct 2016	Due Jan 2017	Due Apr 2017
…					

Tab. 1. Reifegrade für die Qualitätsthemen in Subprojekten

Die stufenweise Einführung der Qualitätsziele hat die Akzeptanz dieser Maßnahmen von Entwicklern erhöht und eine konstruktive Konkurrenzsituation insb. in den Offshore-Subprojekten bewirkt, so dass die Subprojekte schnell Fortschritte erzielt haben. Durch das entstandene Bewusstsein für Qualität erhoffen wir uns, eine bessere Akzeptanz für komplexere Qualitäts-Maßnahmen zu erzielen. Dies umfasst bspw. die Sicherstellung der Rückwärtskompatibilität von Micro-Services und Datenbanken oder effektivere Rollout-Prozesse durch die Nutzung von Virtualisierung.

4 Zusammenfassung

In der Softwareentwicklung ermöglichen agile Vorgehensmodelle zeitnahe, lauffähige Zwischenstände des zu entwickelnden Systems zu erstellen. Dies erlaubt eine schnellere Anpassung an geänderte Anforderungen und damit eine bessere Berücksichtigung von Kundenbedürfnissen. Jedoch ist ihre Einführung in einen etablierten Projektkontext mit vielen Herausforderungen verbunden, insbesondere in Bezug auf das Qualitätsmanagement. In diesem Beitrag haben wir basierend auf einem Fallbeispiel zur Digitalisierung

von Geschäftsprozessen aus dem Bankenumfeld und im Kontext von Multiprojektmanagement und Offshore-Entwicklung diese Herausforderungen zunächst identifiziert.

Darauf aufbauend wurden Anforderungen an ein ganzheitliches und durchgängiges Qualitätsmanagement bei einem Wechsel auf ein agiles Vorgehensmodell aufgestellt. Dabei standen unter anderem die Automatisierung von Entwicklungs- und Testaktivitäten und die Koordination und das Reporting von Testaktivitäten im Mittelpunkt. Diese Anforderungen wurden mittels durchgängigen Qualitätssicherungsmaßnahmen und entsprechende Werkzeuge umgesetzt. Ein für den Ansatz angefertigtes Programm an Kennzahlen hat ein projektübergreifendes Reporting und damit eine Transparenz der Produkt- und Prozessqualität für die Projektteilnehmer ermöglicht.

Zudem wurde ein Vorgehen für die stufenweise Einführung der Qualitätssicherungsmaßnahmen in einzelnen Teilprojekten definiert. Dessen Anwendung hat nach ersten Erkenntnissen wesentlich dazu beigetragen, das Bewusstsein für die Qualität der Software zu erhöhen und das Management der Projekte nach und nach zu verbessern.

Literaturverzeichnis

[BS01] Beedle, M.; Schwaber, K.: Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[Ch08] Chappell, D.: Was ist Application Lifecycle Management. White Paper, 2008.

[Ht12] Httermann, M.: Devops for Developers. Apress, Berkely, CA, USA, 2012.

[HF10] Humble, J.; Farley, D.: Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010.

[FGS15] Fazal-Baqaie, M.; Grieger, M.; Sauer, S.: Tickets without Fine - Artifact-based Synchronization of Globally Distributed Software Development in Practice. In Abrahamsson, P.; Corral, L.; Oivo, M.; Russo, B. (eds.): 16th Int. Conf. of Product Focused Software Development and Process Improvement. Springer, LNCS, vol. 9459, pp. 167-181, 2015.

[FR15] Fazal-Baqaie, M.; Raninen, A.: Successfully Initiating a Global Software Project. In: Industrial Proceedings of the 22nd European Systems Software & Service Process Improvement & Innovation Conference (EuroSPI²2015). WHITEBOX, Denmark, 2015.

[GF15] Güldali, B.; Fazal-Baqaie, M.: Skalieren von großen agilen Projekten mit verteilten Backlogs. In Sikora, E. (eds.): OBJEKTspektrum (Online Themenspecials), no. Agility/2015, pp. 1-4. SIGS DATACOM. 2015.

[Me14] Meyer, B.: Agile!: The Good, the Hype and the Ugly. Springer International Publishing, 2014.

[Sc12] Schneider, K.: Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement. Dpunkt Verlag, 2012.

[SL03] Spillner, A.; Linz, T.: Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester, dpunkt Verlag, 2003.

[Ga03] Gailn, D.: Software Quality Assurance: From Theory to Implementation. Addison Wesley, 2003.