QM in agilen IT-Projekten – quo vadis?

Dr. Christian Brandes
Michael Heller, imbus AG

imbus weltweit
www.imbus.de

QM in agilen IT-Projekten – quo vadis?

Dr. Christian Brandes, imbus AG
Michael Heller, imbus AG

imbus macht
Testen einfach

Autoren:

Dr Christian Brandes

Datum: 20. April 2017
Version: Version 1.0
Status: Final

Michael Heller

imbus AG
Kleinseebacher Str. 9
91096 Möhrendorf
Tel. +49 9131 7518-0
Fax +49 9131 7518-50
info@imbus.de
www.imbus.de

Die finale Publikation ist bei Springer erhältlich via http://dx.doi.org/10.1365/s40702-016-0215-z

Inhalt

Abstract:

Agile Vorgehensweisen erfreuen sich unverändert hoher Verbreitung in IT-Projekten. Der Beitrag untersucht zunächst einige „populäre“ Missverständnisse rund um die Kompatibilität von QM & Agilität. Dann fasst er zusammen, welche unverzichtbaren Best Practices sich in den letzten Jahren zur agilen Qualitätssicherung herausgeschält haben. Und als Abschluss beleuchtet er verschiedene Trends in diesem Bereich, die sich abzuzeichnen beginnen, und bewertet diese kritisch.

Eine Geschichte voller Missverständnisse

Agile Verfahren des Projektmanagements und der Softwareentwicklung haben sich in den letzten Jahren etabliert und strahlen zunehmend in andere Bereiche und Prozesse in Unternehmen aus (etwa Portfolio-Management oder Betrieb). Der Anspruch agiler Methoden – kürzere Time-To-Market bei gleichbleibender oder sogar besserer Produktqualität – wird aber nicht in jedem Projekt erreicht. Erschwerend kommt hinzu, dass etablierte Rollen wie Test- oder Qualitätsmanager von vielen agilen Evangelisten schlicht ignoriert zu werden scheinen. Schauen wir uns deshalb die aktuelle Situation zunächst etwas genauer an und beginnen mit einigen verbreiteten „Missverständnissen zur Agilität“.

„Agil dokumentiert nichts“

Das agile Manifest ([**1**]) formuliert vier grundsätzliche Werte, von denen einer lautet: „Working software over comprehensive documentation“. Dies besagt, dass agile Teams den Wert lauffähiger Software höher einschätzen als den zugehöriger umfangreicher Dokumentation. Die Intention ist klar: Man will vermeiden, zum Liefertermin einen Berg Dokumente, aber ein nur teilweise produktiv nutzbares Produkt abzuliefern. Tatsächlich wird dieser sinnvolle Gedanke aber immer noch gerne in die radikale Formulierung „Im Agilen wird nichts dokumentiert!“ übersetzt – obwohl das so nirgends behauptet wird.

Der scheinbare Widerspruch lässt sich leicht auflösen, indem man zu der Formulierung greift: „Es wird nur dokumentiert, was einen Wert oder Nutzen für irgendeinen Stakeholder hat.“ (Anders ausgedrückt: Es soll nichts mehr nur um des Dokumentierens Willen dokumentiert werden.) Dies erlaubt es nicht nur, in Branchen mit regulatorischen Anforderungen agil zu arbeiten, indem das Team alle Dokumente, die beispielsweise für ein späteres Audit zwingend benötigt werden, in die „Definition of Done“ packt. Dieselbe Lösung schlägt natürlich auch eine simple und funktionierende Brücke zum Qualitätsmanagement des jeweiligen Unternehmens, wenn man alle Dokumente, die das QM verlangt, genauso behandelt. Hier vertragen sich QM und Agilität also problemlos.

Eigentlich wäre damit zu dem Thema alles gesagt, gäbe es nicht noch die Beobachtung in der Praxis, dass die Erwartungshaltung von Auftraggebern „ich kriege Software jetzt viel schneller, weil nur noch codiert und nichts mehr dokumentiert wird“ auf diese Weise wirksam torpediert werden kann. Der fachliche Bedarfsträger muss damit leben, in einem Entwicklungszyklus „weniger Features“ zu bekommen als ihm vorschwebte und/oder ihm versprochen wurde – weil das Team zu jedem Feature eben auch die nötige Dokumentation erstellen muss. Leider wird als Reaktion dann der schwarze Peter den agilen Methoden zugewiesen, nach dem Motto: „Das ist also auch nicht besser.“ Das ist falsch: Vorhandene und nicht wegdiskutierbare Prozessrestriktionen sind eben diesem Prozess geschuldet und nicht dem gewählten Entwicklungsvorgehen.

„Test- und Qualitätsmanager gibt es in agilen Projekten nicht“

Wenn man sich den aktuellen Scrum-Guide ([2]) durchliest, stößt man hinsichtlich Team-Rollen auf sehr deutliche und bestimmte Aussagen: Neben dem „Product Owner“ (als „Single Point of Contact“ für das Projektteam) und dem „Scrum Master“ (als Coach und „Impediment-Beseitiger“) gibt es nur noch das Team selbst, genannt „Entwicklungs(!)team“. Mehr noch: „Scrum kennt keine Titel außer ‚Entwickler’ für Mitglieder des Entwicklungsteams. (...) Es gibt keine Ausnahmen von dieser Regel.“ Domänen wie „Test“ sollen tatsächlich explizit nicht genannt werden! (Diese Sprechweise stellt übrigens sogar eine Verschärfung der Aussagen aus früheren Versionen des Scrum-Guides dar.)

Nun ist „agil“ nicht zwangsläufig identisch mit „Scrum“, aber Scrum ist nach wie vor das am weitesten verbreitete agile Framework ([3]). Folglich nimmt es nicht wunder, wenn wir bei Kundenteams immer wieder auf besorgte Rückfragen treffen wie:

- „Ich bin fachliche/r Tester/in – muss ich jetzt programmieren lernen, wenn in meinem Unternehmen künftig agil gearbeitet werden soll?“
- „Ich bin Testmanager – kann ich mich nun also einen neuen Job suchen, wenn meine Firma auf agile Entwicklung umsteigt?“

Die Intention der Scrum-Autoren zur Team-Zusammensetzung lautet: Nicht jede/r soll „alles“ können, sondern das Team als Ganzes soll alle Fähigkeiten mitbringen, die zur Produktentwicklung und Qualitätssicherung benötigt werden. Jede/r soll alle seine Kenntnisse und Stärken in das Projekt einbringen und sich seinerseits nicht an zu enge Rollenbilder klammern. Dass man über dieses sinnvolle Zielbild (das in der Praxis gleichwohl nur selten vollständig umgesetzt anzutreffen ist) aber eine Art sprachlichen Gleichmacher stülpt – „du warst vielleicht früher einmal Testexperte oder Qualitätsmanager, aber fortan sollst auch du wie alle anderen ‚Entwickler’ genannt werden!“ – ist zumindest fragwürdig. Kein Wunder also, dass das Missverständnis „In agilen Teams gibt es keine Tester bzw. Qualitätsspezialisten mehr!“ sich verbreiten konnte.

Die praxistaugliche Lösung besteht darin, sich als Angehöriger dieser „von Scrum entsorgten“ Rollen ins agile Team zu integrieren und sein Wissen einzubringen, ohne auf Titel oder Spezialisierung zu pochen (bereit sein, dazuzulernen und auch mal andere Aufgaben zu übernehmen ist wesentlich in agilen Teams – Programmierkenntnisse sind also sicher kein Schaden, aber auch kein Muss!). Wir kennen Beispiele, in denen das funktioniert und z.B. Testexperten gern gesehene Sparringspartner in einem „Pairing“ mit einem Entwickler sind. Ein anderer Lösungsweg ist in neueren Frameworks für „agile Unternehmen“ zu finden, worauf wir später noch eingehen werden. Leider kennen wir aber auch beunruhigende Gegenbeispiele – etwa den Fall eines Testmanagers, der sich erfolgreich in ein agiles Team integriert hat, dem aber (da er ja jetzt „nur noch normales Teammitglied wie alle anderen auch“ ist) im Zuge dessen das Gehalt gekürzt wurde. Motivierend dürfte sich das sicher nicht ausgewirkt haben.

„Prozessreifegradmodelle passen nicht zu agilen Prozessen“

Reifegradmodelle wie SPICE oder CMMI, die auch Qualitätsmerkmale für ein QM formulieren, gelten agilen Teams häufig als zu schwergewichtig, zu phasenorientiert und zu formal und dokumentenlastig. Insbesondere scheinen sie der dem Agilen inhärenten Flexibilität zuwider zu laufen, schon allein weil sie die Existenz eines dokumentierten und standardisierten Entwicklungsprozesses propagieren – schließlich will man sich als Team selbst organisieren und jederzeit Veränderungen am Vorgehen vornehmen können. Auch dieser Widerspruch ist bei genauerer Betrachtung aber nur ein scheinbarer: Reifegradmodelle wie SPICE verlangen nicht, dass ein vorgegebener Prozess stur nachimplementiert und eingehalten wird, sondern vielmehr, den eigenen Prozess dahingehend weiterzuentwickeln, dass er gewisse Qualitätsmerkmale erfüllt (siehe Abbildung 1)!

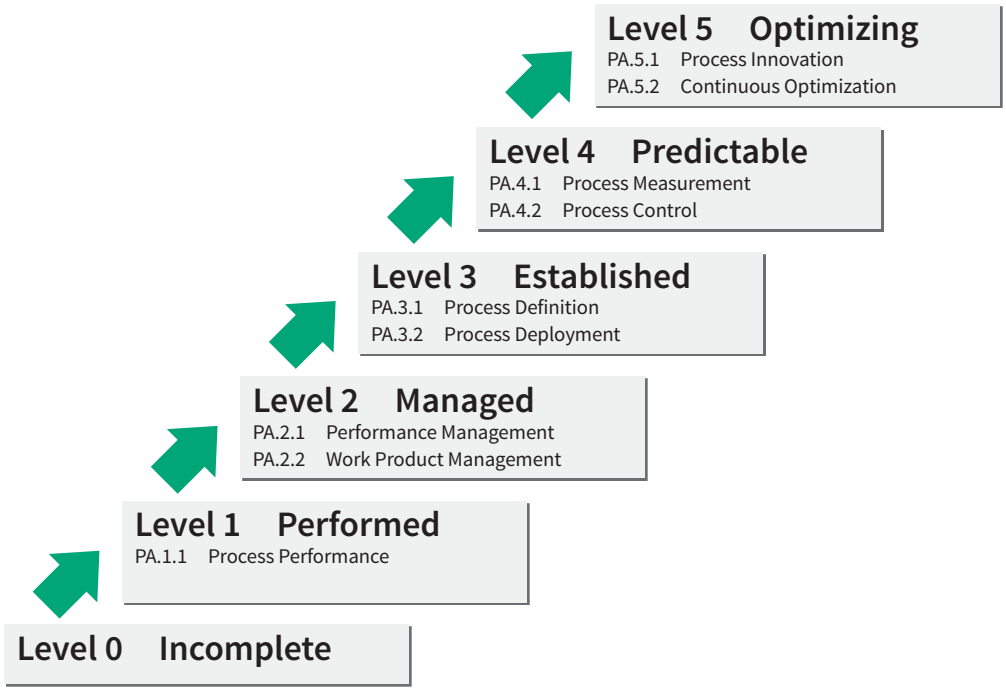

Abbildung 1: Reifegrad-Stufen am Beispiel SPICE

Details zu dieser Fragestellung sind zu finden in ([4]). Hier sei nur darauf hingewiesen, dass Reifegradmodelle auch in agilen Settings angewendet werden können, wenn bei der Anwendung durch den Auditor einige agile Besonderheiten berücksichtigt werden. Und es gibt schlanke Bewertungsmodelle etwa für die Testqualität, die aufwandsarm projektbegleitend angewendet werden und dem agilen Teams wertvolle Verbesserungsvorschläge geben können – insbesondere zur Integration von Test- und QS-Maßnahmen in den selbstgewählten Entwicklungsprozess.

Übrigens sollte jeder Qualitätsmanager eine agile Idee wie etwa das Meeting „Retrospektive“ nach jedem Scrum-Sprint mit Begeisterung begrüßen, stellt es doch nichts anderes dar als die regelmäßige und institutionalisierte „kontinuierliche Prozessverbesserung“. Prozessreife und Agilität sind also alles andere als Gegensätze.

Best Practices zu agiler Qualität

Die Autoren dieses Beitrags sind selber seit vielen Jahren in agilen Kundenprojekten unterwegs – als Mitarbeiter in Scrum-Teams, als Coaches oder auch als Trainer. Dabei hatten wir als Angehörige eines Unternehmens, das sich seit fast 25 Jahren erfolgreich auf Test und Qualitätssicherung spezialisiert hat, stets das Thema „Qualität“ im Fokus – und zwar sowohl die Prozess- als auch die entstehende Produktqualität. Eine der wichtigsten Erkenntnisse der letzten Jahre ist für uns, dass die meisten agil arbeitenden Projekte entwickler-getrieben sind (so sind die meisten uns bekannten Scrum-Master ehemalige Entwickler oder Architekten) und deshalb leider meistens die Test- und QS/QM-Themen zu kurz zu kommen drohen. So gibt es immer noch Scrum-Befürworter, die ernsthaft der Meinung sind, dass es in agilen Projekten nur noch zwei Arten von Tests gibt: Unittests (durch die Entwickler) und Akzeptanztests (durch den Product Owner). Das ist ein sehr einfaches und bequemes Bild, vor allem aber ist es zu einfach und vielen Produktrisiken schlichtweg nicht angemessen.

Qualitäts-Chancen, die Agilität mit sich bringt

Viele QS-Aktivitäten, die wir uns in nicht-agilen Projekten seit Jahren als Tester wünschen, rennen eigentlich in agilen Projektteams offene Türen ein. Dazu gehören eine frühzeitige Beteiligung von Testern im Entwicklungsprozess (Testen ist keine „späte Phase“!), idealerweise schon bei der Anforderungserstellung; frühe Testspezifikation; Berücksichtigung von Testautomatisierung durch Architektur und Entwicklung; sowie professionelles Versions- und Konfigurationsmanagement auch im Test. „Quality from the beginning“ ist ein in der agilen Community weit verbreiteter Slogan – er müsste nur noch konsequent in die Tat umgesetzt werden. Nancy van Schooenderwoert hat dies sehr treffend auf den Punkt gebracht: „If you shoot for quality, speed will come as a side effect; if you shoot for speed, quality will NOT come as a side effect.“ ([5]) Denn: QS von Anfang an hilft spätere Korrekturzyklen zu sparen – eine komplett unoriginelle Erkenntnis, die aber unzweifelhaft wahr ist und von den Vätern der agilen Bewegung auch berücksichtigt wurde.

„Agiles Testen“ ist übrigens kein „neues“ oder „anderes“ Testen. Eine geeignete Definition dafür lautet unseres Erachtens:

Agiles Testen bedeutet, vorhandene und bewährte Testtechniken so im agilen Entwicklungsprozess zu verankern, dass sie die Ziele des agilen Vorgehens unterstützen, nämlich:

- Schnelles Feedback
- Hoher Automatisierungsgrad
- Geringer Management-Overhead
- Enge Zusammenarbeit im Team. ([6])

Dass „geringer Overhead“ nicht „gar kein Management mehr“ heißt, dürfte klar sein. (Generell empfiehlt es sich in agilen Teams, genau auf die Trennung zwischen Aufgabe und Rolle zu achten. Auch wenn es keinen Test- oder Qualitätsmanager mehr als Rolle gibt, heißt das nicht, dass die Aufgaben der Rolle verschwunden sind.) „Agiles Testen“ liefert, zielgerichtet eingesetzt, frühzeitig Informationen zur Produktqualität und leistet seinen Beitrag zur Prozessqualität, indem es etwa Maßnahmen wie eine Retrospektive für die Diskussion von Maßnahmen zur (Test-)Prozessverbesserung nutzt.

Schauen wir also nun etwas genauer auf einige „best practices“, die sich nach unserer Wahrnehmung in erfolgreichen agilen Projekten als unverzichtbar für Test und Qualitätssicherung erwiesen haben:

Agile Teststrategie

Im ISTQB®-Standard ([7]) gibt es ein Artefakt namens „Testkonzept“, in dem die projektspezifische Teststrategie dargelegt wird. Für ein solches Dokument bieten diverse Standards geeignete Vorlagen (hier lohnt etwa auch der Blick zur „neuen Softwaretest-Norm“ ISO 29119 [8]).

Wenn Sie diesen kurzen Textabschnitt den Mitgliedern eines agilen Teams vorlegen, werden Sie mit hoher Wahrscheinlichkeit kollektives Stirnrunzeln und Naserümpfen ernten. Die Begriffe „Standard“, „Dokument“ und „Norm“ lösen dort häufig initiale Abwehrreflexe aus. Dabei ist Prozessreife – wozu die Nutzung praxisbewährter und nutzbringender Ergebnisse gehört – jedem agilen Team ein immanentes Anliegen. Aber die Sorge, von starren und dokumentenlastigen Prozessvorgaben „ausgebremst“ zu werden, ist halt in vielen Köpfen noch allzu gewärtig.

Auf das Thema dieses Abschnitts bezogen heißt dies: Auch agile Teams benötigen eine Teststrategie! Diese muss aber NICHT zwingend aussehen wie ein Testkonzept nach IEEE 829, das Gefahr läuft, als „Schrankware“ ungelesen in einem Regal zu verstauben. Es gibt Formate, die zu agilen Teams kompatibler sind – etwa eine „Definition of Test“ ([6], [9]) auf Grundlage der agilen Testquadranten nach Brian Marick ([10]). Wenn diese ihren Zweck erfüllt, vom Team akzeptiert und gelebt und gepflegt wird und die Frage „Was soll wie getestet werden?“ zielführend beantwortet, sollte kein Qualitätsmanager der Welt daran etwas zu mäkeln haben.

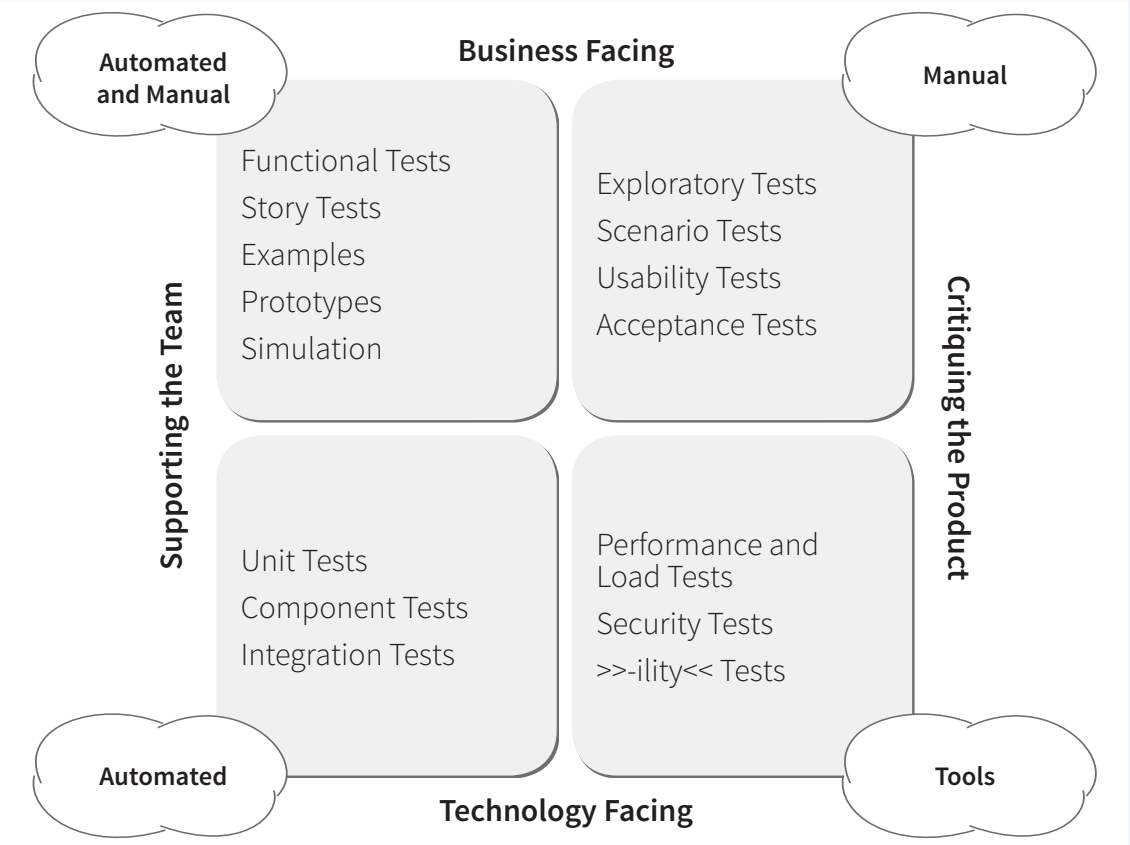

Abbildung 2: Agile Testquadranten nach B. Marick

Anders gesagt: Alles, was etwa der ISTQB®-Standard an guter und bewährter Testpraxis empfiehlt (Einsatz von Testmethoden, risikobasiertes Testen, u.s.w.), lässt sich problemlos auch in ein agiles Setup integrieren. Entscheidend ist auch hier wieder die Frage: Hilft es dem Team, und stimmt das Verhältnis aus Aufwand und Nutzen? Es spricht überhaupt nichts dagegen, in agilen Projekten zu verlangen, dass Requirements (wie etwa User Stories) eine Risikobewertung tragen müssen, oder dass Fehler, deren Behebung sich nicht innerhalb des laufenden Entwicklungszyklus' wird leisten lassen können oder die gar eine andere Organisationseinheit beheben muss, in einem definierten Werkzeug als formales Ticket zur Nachverfolgung dokumentiert werden. Beides unterstützt das Testvorgehen und damit die entstehende Produktqualität.

Zuletzt: Wenn die QM-Vorgaben eines Unternehmens gewisse Ergebnisse und Dokumente (etwa Testprotokolle) verlangen, ist es wie gesehen ein Leichtes, dies in eine agile Team-Charta aufzunehmen. Und dass die „agile Teststrategie" als Teil der kontinuierlichen Prozessverbesserung etwa in Sprint-Retrospektiven regelmäßig kritisch hinterfragt und vom Team gemeinsam verbessert werden kann, steht außer Frage. Es spricht also nichts dagegen, die Anforderungen eines QM-Systems in agilen Projekten umzusetzen – so zumindest unsere Projekterfahrung. Man sollte sich aber dem Umstand, dass QM- Vorgaben von agilen Teams auch mal kritisch hinterfragt werden, ebenso unvoreingenommen stellen.

Testbare Anforderungen

In vielen agilen Projekten – nehmen wir als weit verbreitetes Beispiel die „User Stories" aus Scrum – wird verlangt, dass Anforderungen Akzeptanzkriterien tragen. Nach unserer Erfahrung verdienen aber die meisten entstehenden Kriterien diesen Namen nicht: Anstatt auszudrücken, was prüfbar erfüllt sein soll, damit der Auftraggeber („Product Owner") das Projektergebnis abnimmt, formulieren sie ihrerseits wieder lediglich weitere Anforderungen, bei denen überhaupt nicht klar ist, wie man sie denn nun prüft und entscheidet, ob sie erfüllt sind oder nicht. Das Ideal wäre, mit automatisierten Akzeptanztests zu arbeiten. (Als Spielart könnte man dies als „test first" aufziehen und würde bei „acceptance test driven development/ ATDD" landen.) Das Minimum sollten nach unserer Erfahrung Beispiele sein – der Ansatz „Specification By Example" (**[11]**) erfreut sich zunehmender Beliebtheit.

Außerdem sollten Anforderungen eine Risikobewertung tragen, die etwa in einem Planning-Meeting dem Team hilft, den Testaufwand entlang der definierten Teststrategie festzulegen und als Aufwandsschätzung in den Planungsprozess einzubringen. Risikobasiertes Testen, ein fundamentales Erfolgsprinzip, passt problemlos in ein agiles Vorgehen.

Und nicht zuletzt haben Tester in agilen Teams die Möglichkeit, nicht testbare Anforderungen abzulehnen – sei es, dass sie unverständlich sind, zu groß oder zu vage. Auf diese Weise lässt sich das Qualitätsmerkmal „Testbarkeit" gut im agilen Prozess verankern, wobei sich ein professioneller Tester nicht darauf beschränken sollte, fehlende Qualität von User Stories anzumäkeln: Er sollte auch konstruktive Mitarbeit an Verbesserungen anbieten – etwa in Form eines „Pairings" mit dem Anforderungs-Autor. Darauf werden wir später noch einmal eingehen.

Reife Testautomatisierung

Testautomatisierung bezeichnet in der Regel die Automatisierung der Testdurchführung, d. h. geeignete Werkzeuge (sogenannte »Testroboter«) geben Tests automatisiert z. B. über Nacht zuverlässig und reproduzierbar in das Testobjekt ein und protokollieren die Testergebnisse. Nun gibt es aber unterschiedlich reife Formen von Testautomatisierung: Lässt man sich etwa auf das Aufzeichnen manueller Testdurchführungen ein (genannt »Capture & Replay« oder »Record & Playback«), wird man mit hoher Wahrscheinlichkeit mittelfristig vor massiven Wartungs- und Pfegeproblemen stehen. Letzteres, auch bekannt als »Anwachsen der technischen Schuld« (increasing technical debt) ist einer der Hauptgründe dafür, warum so viele Testautomatisierungen scheitern und nur eine kurze Lebensdauer haben. Dies bezeichnen wir als „unreife" bzw. nicht nachhaltige Automatisierung.

Agile Projekte nun sehen in der Automatisierung von Tätigkeiten, die sinnvoll und nutzbringend automatisiert werden können, generell einen hohen Wert. Für Komponententests beim Entwickler ist dies in der Regel eine Selbstverständlichkeit: Die heute gängigen Unittest-Frameworks am Markt sind ja gerade deshalb so erfolgreich und akzeptiert, weil sie zu automatisierten Tests führen und den Entwickler in seiner gewohnten Tätigkeit abholen. Trotzdem gilt es auch hier, auf die Qualität der Tests und die Lesbarkeit des Testcodes zu achten! Regeln wie »Clean Code« ([12]) oder »Collective Code Ownership« (gemeinsame Code-Verantwortung) sollten genauso für den Testcode wie den Programmcode gelten.

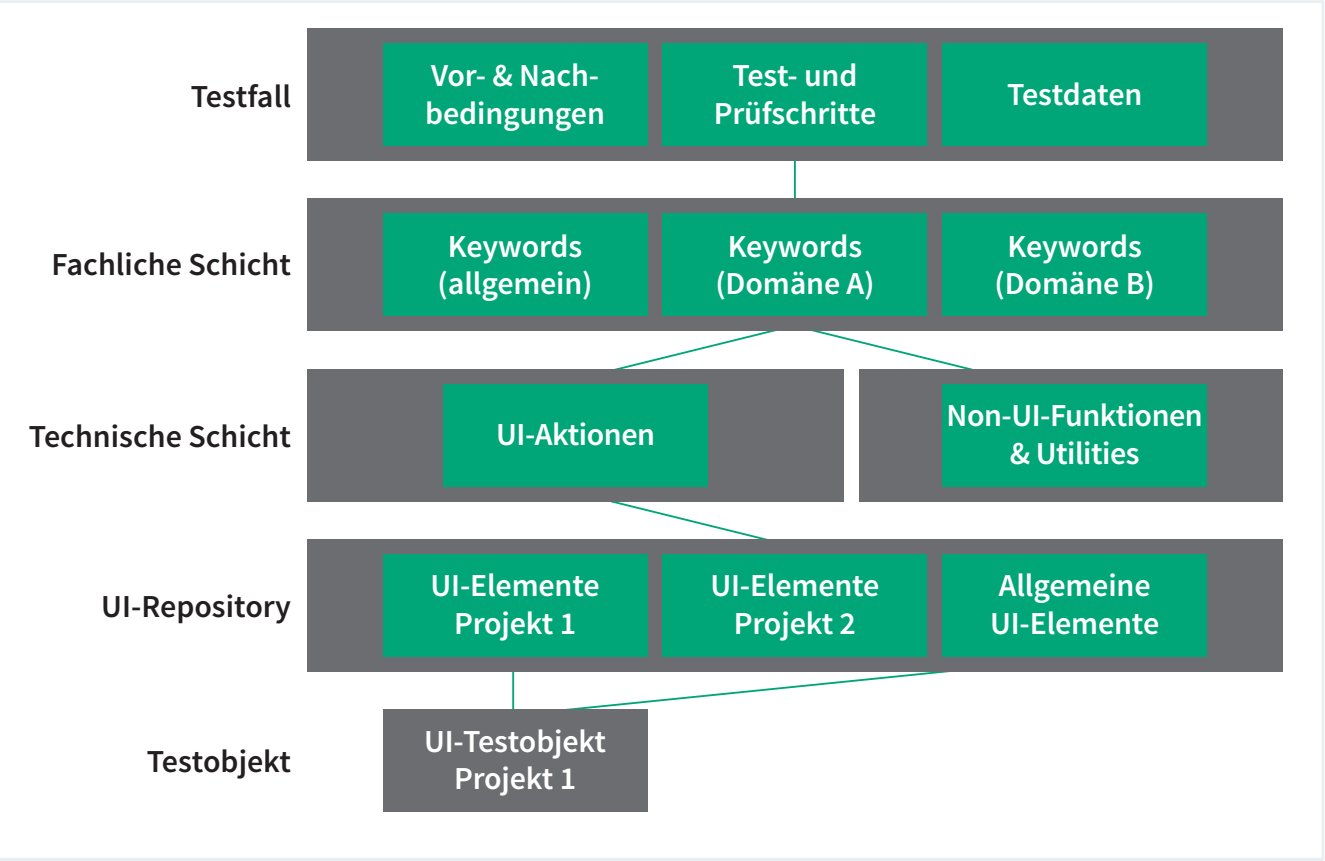

Abbildung 3: Reife Testautomatisierung trennt modular zwischen Testlogik und Implementierung

Für Systemtests und Abnahmetests – also Tests aus Benutzerperspektive – sieht die Lage indes deutlich anders aus: Zwar existieren verschiedene Frameworks für »automatisierte Akzeptanztests« am Markt, doch sind die Erfolge nicht immer zufriedenstellend. Nötig sind hochwertige und reife Automatisierungslösungen. Letztlich verlangt dies nach einer tragfähigen Automatisierungsarchitektur mit Schichtenbildung und Modularisierung (siehe Abbildung 4). Insbesondere datengetriebenes und schlüsselwortbasiertes Testen sind in der Praxis bewährte Ansätze, die dafür sorgen, dass die Erstellung neuer und die Anpassung vorhandener automatisierter Testfälle überhaupt mit den kurzen Zyklen agiler Projekte mithalten können! Anders gesagt: Aus wirtschaftlichen Gründen sollte Testautomatisierung immer einen hohen Reifegrad aufweisen – und in agilen Projekten verschärft sich diese Qualitätsanforderung zu einem unverzichtbaren Muss.

Continuous Integration

Wenn Continuous Integration (CI) zur Sprache kommt, gilt der erste Gedanke im Regelfall den Werkzeugen. Die meisten agilen Teams und Projekte, denen wir in unserer Tätigkeit begegnet sind, hatten entweder etablierte Toolketten für CI oder waren gerade dabei, sie einzuführen. Grundlage ist ein Tool zum Konfigurationsmanagement und zur Versionsverwaltung. Neben einem CI-Server (hier kommen sowohl etablierte Open-Source-Lösungen als auch ALM-Tools in Frage) umfasst die Minimalmenge für CI typischerweise Tools für Whitebox-Tests und Unittests des Entwicklungscodes, den Compiler, und natürlich Tools für den automatischen Buildprozess.

Abbildung 4: Adressierte Themen in einer Werkzeugkette zur kontinuierlichen Integration

Laufen diese Werkzeuge auf den Rechnern der Entwickler und auf einem weiteren Rechner für zentrale Builds, der ähnlich konfiguriert ist wie die einzelnen Entwicklungsrechner, so lassen sich schon nutzbringend Unittests (auch genannt automatisierte Entwicklertests oder Komponententests) automatisieren und im Team gemeinsam nutzen. Spannend wird es beim Deployment in komplexere Zielumgebungen – denn wie diese Zielumgebung aussieht, entscheidet ja darüber, welche Arten von automatischen Tests danach überhaupt möglich sind. Und in der Qualität der Tests schlummert natürlich das Herzstück der CI – vor allem hochwertige Tests bieten das Potenzial für eine Steigerung der Softwarequalität. Die Werkzeuge am Markt bieten inzwischen einiges an Hilfestellung für das Handling von Zielumgebungen. Oft spielt Virtualisierung dabei eine große Rolle, denn meistens ist es preisgünstiger, eine virtuelle Maschine anstelle realer Hardware automatisiert zu bestücken.

Manchmal stößt man auf Aussagen wie: „Der Aufwand für die Einführung von CI-Tools macht sich bereits nach einem Monat wieder bezahlt, da CI wesentlich zur Effizienzsteigerung der Entwicklung beiträgt.“ ([13], S. 403). Es gibt durchaus viele Projekte, die genau diese Erfahrung machen, und wenn es um ein neu startendes Softwareprojekt geht, stehen die Chancen auch gut. Genauso häufig wird CI aber ein Fall von „verbrannter Erde“: Mit hohem Aufwand wurde mit CI-Toolketten der Versuch gestartet, teamübergreifende Tests zu automatisieren und regelmäßig durchzuführen (weil man das ja agil so macht). Und irgendwo

auf dem Weg sind dann die Aufwände davongelaufen, bevor genug Nutzen sichtbar wurde. Es gab viele Unittests – aber zwei Drittel davon melden monatelang „rot“, und neue Features sind gerade viel wichtiger, so dass keine Zeit im Team bleibt, sich um die Wartung und Pflege der automatischen Tests zu kümmern. Der Weg zur agilen Reife liegt auch hier in einem konsequenten Quality-First-Mindset: Wenn jeder automatisierte Test, der nicht PASSED ist, sofort einen Kümmerer findet, wird auch der Aufwand für die Fehlersuche in den automatischen Tests leichter handhabbar.

Ownership für die CI-Tests ist also ein zentraler Erfolgsfaktor. „Kümmern“ bedeutet natürlich nicht nur, gefundene Bugs möglichst sofort zu fixen, sondern auch Testumgebungen, Testdaten und Testfälle angemessen zu warten und lauffähig zu halten. Dabei hilft ganz wesentlich eine Kerndisziplin klassischen Testmanagements: das risikobasierte Testen. Denn leider interpretieren agile Teams CI oft als „möglichst viel automatisiert testen“. Also wird kaum jemals die Entscheidung getroffen, dass Tests (insbesondere wartungsintensive Systemtests auf der GUI-Ebene) nicht „genug“ Risiko abdecken und abgeschaltet werden können oder schlankeren/früheren Tests weichen dürfen. Und am Ende zieht dann jemand die Notbremse, und die ganze Regressions-Testsuite wird auf Eis gelegt.

Die Erkenntnis, dass eine Konzentration auf das Wesentliche zum CI-Erfolg beiträgt, ist übrigens keineswegs neu: Schon die Väter des Extreme Programming (XP) hatten den 10-Minuten-Build zu einem „Must Have“ erkoren. Neben der technischen Exzellenz, also in diesem Fall der Investition in schnelle Computerhardware, Datenbanken und Netzwerke und performanceoptimiert ablaufende automatische Tests, bedeutet die Forderung, dass via CI 10 Minuten nach dem Einchecken von Code zurückgemeldet wird, wie gut sich dieser Code mit der Arbeit von Teamkollegen und der Arbeit anderer Teams integriert, eben auch professionelles Management der Werthaltigkeit von Regressionstests. Zu erwähnen ist in diesem Zusammenhang noch der übliche Nightly-Build, in großen Projekten auch oft ein Weekly-Build am Wochenende. Diese Anteile der CI bieten meist dem Großteil der automatisierten Systemtests eine Heimat, weil Systemtests naturgemäß nicht so schnell ablaufen wie Unittests. Das zur Verfügung stehende Zeitfenster wächst in der Nacht bzw. am Wochenende auf mehrere Stunden oder gar zwei Tage. Die Aufgabe, für eine risikoorientierte Auswahl der durchgeführten Tests zu sorgen, bleibt dabei erhalten.

Wie im Abschnitt „Reife Testautomatisierung“ beschrieben, ist für Systemtests und Abnahmetests eine tragfähige Automatisierungsarchitektur nötig. Dazu gehören natürlich auch passende Tools:

- Testroboter, die Skriptsprachen mitbringen und GUI-Elemente robust und zuverlässig erkennen
- Testmanagementsysteme, die Keywords unterstützen und die Automaten steuern können.

Damit Systemtests im CI nutzbar sind, ist ein teamübergreifendes automatisches Deployment in eine gemeinsame Testumgebung und ein gemeinsames Testdatenmanagement erforderlich, bei dem der Pflegeaufwand für die automatisierten Tests ihrem Nutzen nicht davonläuft. In großen Projekten ist keines dieser Themen billig oder schnell zu haben.

Apropos nicht billig: Vermehrt versuchen auch die Hardwarekollegen im Zusammenspiel mit den „Softwerkern“ den Nutzen agiler Vorgehensweisen abzugreifen. In einem unserer letzten Projekte durften wir zum Beispiel mit agilen Teams arbeiten, die Software für die Steuerungscomputer eines Herstellers von Wälzstraßen für Stahl entwickeln. Was bedeutet die Aussage „Das Gesamtsystem ist neu gebaut“ in einem solchen Kontext? Im produktionsnahen Gesamtsystem gäbe es definitiv heißen Stahl, und so eine Testumgebung ist nicht bezahlbar und nicht per CI im 10-Minuten-Takt beherrschbar. Andererseits ist es auch beliebig teuer, Integrations-Fehler erst beim Endkunden zu entdecken, der seine Fabrik schon gebaut hat. Einen Lösungsansatz bot hier – wie in vielen hardwarenahen Fällen – der Einsatz von Simulatoren. Im Fall der Wälzstraße war für die bereitgestellten Simulationen sehr viel Wissen und physikalisches Know-How aus dem Bereich der Werkstoffwissenschaften nötig, um die Software zu entwickeln, die virtuell viele der Einstellmöglichkeiten und Sensoren einer echten Stahlwälzstraße bereitstellt.

Dieses zugegebenermaßen extreme Beispiel macht ein Kernthema von CI gut sichtbar: Die Frage, wieviel Aufwand getrieben werden soll, um zu einem testbaren Gesamtsystem zu integrieren, muss sich einerseits an Wirtschaftlichkeitsbetrachtungen orientieren. Andererseits ist schon der automatisierte Entwicklertest, der über die reine eigene Komponente hinausgeht und als Integrationstest einen „Mock“ (Platzhalter) für andere Komponenten benötigt, oft in Gefahr, dem Featuredruck zum Opfer zu fallen. Theoretisch schätzt ja das Team seine Aufwände inklusive aller nötigen qualitätssichernden Maßnahmen ab, so dass genug Zeit für die Erstellung und Pflege von Mocks eingeplant sein sollte. Aber gegen den mächtigen Marktdruck sind nach unserer Erfahrung auch agile Projekte nicht gefeit. Und ohne eine solide Testabdeckung in den verschiedensten Schichten des zu entwickelnden Gesamtsystems wird auch die Qualitätssteigerung nicht erreicht, die nötig ist, um eine agile Zusammenarbeit mit weniger formalen Quality Gates erst realistisch zu machen. Umgekehrt wird ein Schuh daraus: Dort, wo agile Projekte noch nicht in der Lage sind, kontinuierlich zu integrieren und zu testen, sollten sie auch die klassischen Quality Gates – z.B. Systemintegrationstests als formale Teststufen – nicht aufgeben. Eine schöne Begrifflichkeit dafür bieten „Undone Departments“, die wir später noch erläutern werden.

Am Ende jeder CI-Toolkette findet sich ein Berichtswesen (Reporting). Oft gibt es große Bildschirme in Teamräumen, auf denen zu sehen ist, wie die letzten Builds funktioniert haben. Das ist hilfreich, denn es fördert das Quality-First-Denken im Team. Typisch ist dabei die Ermittlung einer Code-Abdeckung der durchgeführten Tests. Oft werden wir nach einer allgemeinen Zielgröße für die Code-Abdeckung gefragt, aber diese Frage führt aus zwei Gründen auf die falsche Fährte: Einerseits hat jedes Projekt andere Voraussetzungen – es macht daher meist mehr Sinn, die zeitliche Entwicklung der Code-Abdeckung innerhalb eines Projektes zu beobachten. Andererseits ist der Testerblick für eine Bewertung der Abdeckung von automatischen Tests viel wichtiger als eine reine Reporting-Kennzahl. Teams, die ihre Unittests im Pairing zwischen Tester und Entwickler möglichst hochwertig vorantreiben, kommen meist weiter als Teams, die formal eine 100%-Codeabdeckung fordern und mit viel Aufwand und Fleißarbeit von Entwicklern wenig aussagekräftige Tests für Legacy Code nachrüsten, der im schlimmsten Fall noch nicht mal häufig von Änderungen betroffen ist. Das ist eine der Chancen von agiler QS – die Fälle, in denen Systemtester von den Unittests nur wissen, dass es sie „vermutlich“ gibt, werden deutlich seltener.

Blick nach vorn

Nun schauen wir noch voraus und betrachten einige der Trends, die sich rund um Agilität & QM abzeichnen. Diese berühren verschiedenste Bereiche – von der Unternehmensorganisation über Projekt-Artefakte bis hin zur Aus- und Weiterbildung.

Agil „im Großen“

In vielen Unternehmen, die auf den agilen Zug aufgesprungen sind, kam die ursprüngliche Initiative aus den Reihen der Entwickler selbst. Der Fokus lag dann darauf, einzelne Teams in die Welt agiler Werte einzuführen und im Laufe der Zeit zunehmende Teamreife zu erreichen. Agile Teams konnten auch vielfach sichtbar ihren Mehrwert unter Beweis stellen. Die Managementsicht auf Agilität in der Softwareentwicklung veränderte dadurch im Laufe der Jahre ihre Einschätzung von „exotische und undisziplinierte Spielerei unserer Entwicklungsabteilung“ zu „State-of-the-Art und Königsweg zum Kostensparen“. In beiden Fällen geht das nicht unbedingt mit einem Verständnis der zugrunde liegenden Werte einer Lean-Agile-Leadership einher. Zugleich wuchsen aber auch die Größe und Komplexität der Projekte, die von agilen Teams abgewickelt werden. Verschiedene Frameworks, die über den ursprünglichen Scrum Guide hinausgehen, entstanden. Diese Frameworks entwickeln Vorschläge, wie teamübergreifende Qualität etabliert werden kann. Denn hier liegt ein potenzieller Schwachpunkt jedes agilen Teams: Die Qualität endet bei einer „Definition of Done“, die mit den Mitteln des Teams erreichbar ist. Außerdem optimieren stark selbstorganisierende Teams verwendete Werkzeuge auch sehr stark auf die eigenen Bedürfnisse hin. Es besteht nicht viel Anreiz zu teamübergreifenden Kompromissen oder gar unternehmensweiter Standardisierung.

Eine Gemeinsamkeit verschiedener agiler Frameworks sind mithin „übergreifende Teams“ oder Zentralteams, die mit anderen Teams bei der Qualitätskontrolle und bei technischer Infrastruktur zusammenarbeiten. Um drei Beispiele herauszugreifen:

SAFe®

Das recht bekannte „Scaled Agile Framework“ ([14]) wurde zum Jahresbeginn 2016 in einer neuen Fassung 4.0 vorgestellt. Es führt viele über den Scrum Guide hinausgehende Begriffe, Teams und Rollen ein. Insbesondere kennt es ein „System Team“ und verankert dort ein „end-to-end solution testing“. Es bietet damit den klassischen Systemintegrations-Testern eine Heimat an.

LeSS

Das “Large-Scale Scrum Framework” ([15]) prägt den Begriff der “Undone Departments” (siehe Abbildung 5).

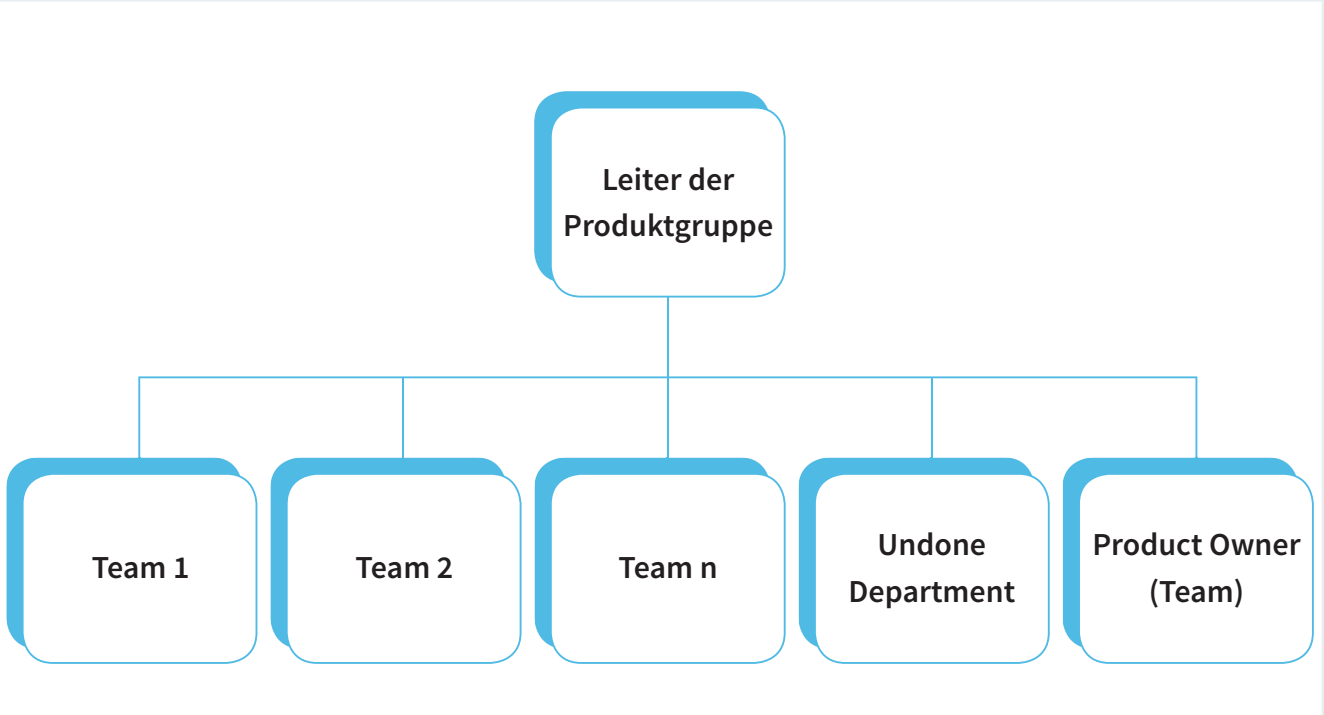

Abbildung 5: „Undone Department“ als Zentralteam im LeSS-Framework

Zu den „Undone Departments“ zählt LeSS Qualitätssicherung, Testabteilungen, Architektur-Teams und Business-Analyse-Gruppen. Gleichzeitig fordert LeSS, dass man idealerweise keine Undone Departments brauchen sollte. Aber manchmal seien agile Teams eben (noch) nicht in der Lage, wirklich potenziell auslieferbare Softwareinkremente in jedem Sprint zu erzeugen, und hierbei könnten sie hilfreich sein.

Nexus™

Dieser neue Spieler im Reigen der „großen“ agilen Frameworks hat die Bühne im August 2015 betreten ([16]). Unterzeichner ist wie beim Scrum Guide Ken Schwaber, und wie auch der Scrum Guide ist die Beschreibung sehr knapp gehalten. Der Nexus-Guide propagiert sog. „Nexus Integration Teams“.

In der Zusammenfassung: Die Definition of Done einer User Story im Sprint lässt mit zunehmender Komplexität zunehmend große Lücken. Diese müssen gefüllt werden - egal ob durch SAFe® System Teams, Nexus Integration Teams oder eben Less® Undone Departments. Wir halten den Begriff des „Undone Departments“ für eine durchaus elegante Generalisierung von „übergreifende Teams“, die sich gut in den agilen Sprachgebrauch einreiht.

Im Geiste der Lean-Agile-Leadership funktionieren Zentralteams am besten im Sinne eines „Servant Leaders“ ([17]). Sprich: Die zentralen Teams müssen den Qualitätsgedanken gleichzeitig einfordern und coachen, aber auch die Transformationsstrategien aus den Ideen der Einzelteams heraus schrittweise bewältigen. Um klarzustellen, dass neben dieser Grundhaltung auch wirklich Ahnung über Lean und Agile wichtig ist, erweitert SAFe® die Forderung zum „Lean Thinking Manager Teacher“ ([18]). Die Erfahrung zeigt: Die Mischung macht’s. Wer konkrete Toolunterstützung im Angebot hat, kann vermutlich leichter den Qualitätsgedanken im Sinne eines Angebots und weniger den erhobenen Zeigefinger in die Teams einbringen. Gleichzeitig darf die Gesamtqualität nicht dem meist heterogenen Reifegrad einzelner Teams überlassen bleiben. Die Frameworks für agile Großprojekte bündeln daher oft klassische Qualitätssicherungs- und Systemtest-Themen mit Themen, die ursprünglich in der Entwickler-Domäne zu finden sind (z.B. einer zentralen Betreuung von Branching-Strukturen und Tools zur Continuous Integration).

Wichtiger Tipp aus Systemtest-Sicht: Große Projekte sollten zentrale Strukturen aufbauen, die Unterstützung für ein gemeinsames Tooling bei der Automatisierung von Systemtests bieten. Der Teil 5 der Norm ISO 29119 ([8]) beschreibt mit „Keyword Driven Testing“ einen praxisbewährten Ansatz. Gerade die Maxime „jedes Team sucht sich selbst die besten Tools zusammen“ kommt im Großen wie gesagt schnell an ihre Grenzen. Die Zentralteams tragen in der Praxis die verschiedensten Namen, denn sie spiegeln immer auch die historische Entwicklung im Unternehmen wieder. In Projekten sind wir z.B. auf Begriffe wie „Alpha-Gruppe“ gestoßen. Hier lauern unter Umständen klassische Projektmanagement- bzw. Wasserfall-Denkweisen: eine übergeordnete Einheit „pusht“ Arbeitspakete in die untergeordneten Teams. Solche Entwicklungsstufen sind aber nicht untypisch für agile Transformationen, die ja inzwischen nicht mehr nur „bottom-up“ bei den Einzelteams beginnen, sondern zunehmend auch „top-down“ aufgrund einer Managemententscheidungen ablaufen. Der Nexus Guide formuliert als Forderung an das Zentralteam: „They should use bottom-up intelligence.“. Das ist leichter gesagt als getan.

Zur Rolle des Qualitäts- bzw. Testmanagers

Nochmal zurück zu „Test- und Qualitätsmanager gibt es in agilen Projekten nicht“. Leider gehört es für uns zum Projektalltag, von – ehemaligen – Testmanagern zu hören, dass bei einem „Going Agile“ plötzlich niemand mehr sagen konnte, wo denn der Platz sein soll, an dem zukünftig ihre Fähigkeiten und ihr Wissen gefragt sein werden. Dabei ist der unbedingte Fokus auf „Built-In-Quality“ und mithin auch professionelle QS eine der treibenden Kräfte im Agilen. Hier liegen die eingangs beschriebenen Qualitäts-Chancen, die Agilität mit sich bringt. Wie kann es da sein, dass so mancher Scrum Master keine Testmanager mehr zu brauchen glaubt?

Wer wenig Zeit hat, in den verschiedensten Beschreibungen von Frameworks zu schmökern (eine durchaus empfohlene Lektüre, denn dort findet sich viel Praxisrelevantes), dem sei zumindest der Nexus Guide ans Herz gelegt. Wir sehen dort die ehemaligen Testmanager als wertvolle Teammitglieder im Nexus Integration

Team. Die Rolle des Nexus Teams ist im Guide folgendermaßen definiert: „Nexus Integration Team Members are responsible for coaching and guiding the Scrum Teams in a Nexus to acquire, implement, and learn (…) practices and tools. Additionally, they coach the Scrum Teams on the necessary development, infrastructural, or architectural standards required by the organization to ensure the development of quality Integrated Increments. ([16])".

Zu Beginn einer agilen Transformation kann das bedeuten, ganz klassisch mit Teststufen aus dem allgemeinen V-Modell zu arbeiten, die in den Nexus Teams verantwortet werden - für Testaufgaben, die eben noch nicht in einen Sprint vorgezogen werden können. Im Laufe der Reifung sollten dann aber die Nexus Teams kleiner werden. Das ist im Sinne von Lean Agile Leadership gedacht und ganz ohne Wunschdenken. Wunschdenken (wishful thinking) wäre: „wir haben uns jetzt ja agil organisiert und haben unsere Unittests, und rein deshalb brauchen wir keine Systemintegrationstest mehr". Um einen Wunsch aus Sicht der Autoren zu äußern: Wir wünschen uns, dass Testmanager sich der Herausforderung Transformation mit Freude stellen – ganz im Sinne eines „Lean Thinking Manager Teacher". Und die gute Nachricht ist: wir kennen neben den erwähnten Schmerzen in der Übergangsphase auch sehr viele persönliche Berichte von Menschen, für die dieser Weg erfolgreich war.

„User Story Workshops"

Ein agiler Umbruch im Verhältnis zwischen Testern, Testmanagern und den anderen Projektbeteiligten schimmert im Nexus Guide an folgender Stelle durch: „Product Backlog items are often resolved to a granularity called thinly sliced functionality." Was bedeutet die Forderung, dass Anforderungen „dünn geschnitten" (thinly sliced) sein müssen? Der Umbruch besteht darin, dass Tester in agilen Teams aufhören sollten, sich über ungenaue oder schlecht testbare Anforderungen nur zu beschweren: Stattdessen sollten Tester ungefragt und immer wieder den Product Ownern und Entwickler-Kollegen ein Dienstleistungsangebot machen („Manager Teacher"), das auf Dauer nicht abgelehnt werden kann und darf: „Wir sind Testexperten und wir sind hier. Was steht als Nächstes im Team an? Lasst es uns gemeinsam vorbereiten und testbar gestalten!" Der zugehörige Prozess ist in der agilen Literatur bekannt und wird mit „Backlog Refinement" oder „Backlog Grooming" bezeichnet. Warum es sich trotzdem lohnt ihn zu erwähnen: Viel zu häufig erleben wir in agilen Teams dann doch, dass die Tester eine User Story im Sprint Planning zum ersten Mal zu Gesicht bekommen - und dann nur noch schnell den Testaufwand schätzen dürfen. Und das reicht einfach nicht für gutes agiles Arbeiten.

Der ISTQB® Agile Tester Lehrplan ([19]) formuliert dazu im Abschnitt „Kollaborative Erstellung von User-Stories": „Für die Zusammenarbeit bei der Erstellung der User-Story können Techniken wie z. B. Brainstorming oder Mind Mapping genutzt werden.". Das ist schon mal ein Anfang. Ein sehr schönes weiterführendes Angebot zu diesem Thema findet sich bei „Discover to Deliver" ([20]). Der Name ist Programm: entdecken (discover), um dann liefern zu können (deliver). Ellen Gottesdiener und Mary Gorman schlagen dort vor, 7 sog. Dimensionen einer User Story zu beleuchten, um ein tieferes Verständnis für die Anforderungen zu bekommen. Der praxistaugliche Ansatz betont übrigens, wie wichtig es ist, Dinge zu visualisieren, um sie besser diskutieren zu können.

Und um in diesem Zusammenhang noch einen dritten Ansatz ins Spiel zu bringen: Modellbasiertes Testen (Model Based Testing) kann auch für agile Teams eine Bereicherung sein. Schon ein GUI-Mock-Up ist in gewissem Sinne ein grafisches Modell, das den Projektbeteiligten hilft, gemeinsam über zukünftige Features zu reden. Und man kann sogar weiter gehen und Abläufe oder Strukturen als formale Modelle in entsprechenden Werkzeugen „modellieren". Solche Modelle können nicht nur in einem User-Story-Workshop die Teamdiskussion bereichern, sondern sie bieten die Chance auf mehr Automatisierung, genauer: eine Automatisierung des Testdesigns. Mit Hilfe von entsprechenden Werkzeugen können sogar Testfälle aus solchen Modellen abgeleitet werden ([21]).

Testbarkeit via Architektur

Die Zusammenarbeit zwischen QS/QM und Architektur wird in Zukunft definitiv an Bedeutung gewinnen. Ziel sollte eine frühzeitige Kommunikation sein, um konstruktive QS-Maßnahmen bereits in den ersten Architekturentscheidungen geeignet vorzusehen. Auf diese Weise lässt sich das oft stiefmütterlich behandelte nichtfunktionale Qualitätsmerkmal „Testbarkeit“ zielführend und kostenminimierend berücksichtigen. Details hierzu sind zu finden in **[22]**.

Das grundsätzliche Thema „Design for Testability“ ist zwar nicht neu, aber den Kinderschuhen bisher nicht wirklich entwachsen. Auch hier bieten agile Teams Chancen – etwa mittels kleiner Schritte, die sich inkrementell dem Ziel nähern (**[13]**). Den Chancen steht aber auch ein zunehmender Erfolgsdruck gegenüber: Schaut man sich Themen wie „Internet of Things“ oder „Cloud Testing“ an, wird schnell klar, dass aus Qualitätssicht Integrationstests, Testumgebungen und Testdatenmanagement allesamt sowohl an Bedeutung als auch an Komplexität zunehmen (**[23]**). Nötig sind schon heute geeignete Test-Schnittstellen (nach ISTQB®: „Points of Control“ bzw. „Points of Observation“) sowie ein funktionierendes Konfigurationsmanagement. Tatsächlich sind Architektur, QM und agile Teams hier gemeinsam gefordert.

„DevOps“

Das agile Mindset einer respektvollen und vertrauensvollen Zusammenarbeit lässt sich nicht nur auf Tester und Architekten anwenden, sondern auf die Bereiche Entwicklung (DEVelopers) und Betrieb (OPerations). Das Ergebnis wird derzeit unter dem Kunstwort „DevOps“ zusammengefasst. Auslöser ist ein Zielkonflikt zwischen den beiden Bereichen, der durch agile Entwicklungsprozesse noch verschärft wird: Während agile Teams in kurzen Releasezyklen Software an den Kunden ausliefern wollen, hat der Betrieb traditionell das Bestreben, Änderungen an stabil laufender Software möglichst zu vermeiden. Beide wollen damit im Unternehmen jeweils ihren Wert unter Beweis stellen: Die Entwicklung liefert schnell und oft nützliche neue Features, der Betrieb sorgt für zuverlässig und stabil laufende Software. Gibt es Probleme, suchen beide schnell die Schuld beim jeweils anderen. Dies künftig zu vermeiden, ist der Anspruch der DevOps-Bewegung. Für unser Thema besonders spannend: Die gemeinsame Klammer für beide heißt „Qualität“!

DevOps versucht, die „Mauer“ zwischen Entwicklung und Betrieb einzureißen – durch gemeinsam genutze Werkzeuge, etwa für Konfiguration, Deployment, Umgebungsmanagement. Stichworte sind „Infrastructure as Code“ und „Continuous Delivery“. Aus QM-Sicht stecken hier tatsächlich Chancen – durch gemeinsam genutze (und qualitätsgesicherte) Artefakte, konsequentes Versions- und Konfigurationsmanagement und umfassendes Denken in „quality first“. Besonders charmant daran ist, dass viele der unter „DevOps“ anzutreffenden Ideen als konstruktive QS-Maßnahmen bewertet werden können, während etwa die typische Betriebsaufgabe „Monitoring“ (die natürlich nicht verschwindet) rein reaktiv wirkt. Andererseits bietet sich hier zugleich eine Kennzahl an, die ein kritisches Firmen-Management vom (auch monetären) Nutzen von DevOps-Neuerungen überzeugt: Veränderungen erzeugen Kosten, und wenn das Monitoring belegt, dass Probleme und Fehler (mit anschließender kostspieliger Behebung) nachweislich abnehmen, hat sich die Investition in DevOps-Mechanismen gelohnt. Fehlerkosten sind ja generell ein guter Hebel, um Veränderungen zu mehr Qualität zu motivieren.

Anders gesagt: Ein Qualitätsmanagement sollte derartige Bestrebungen wohlwollend begleiten, natürlich auch kritisch hinterfragen, und entsprechende Qualitätsmetriken anbieten, an denen sich die Befürworter agiler Veränderungen messen lassen wollen.

ISTQB® agil

Zertifikate als Nachweis einer erfolgreichen Aus- und Weiterbildung werden in vielen agilen Teams ebenfalls oftmals kritisch betrachtet. Entscheidender bzw. mindestens genau so wichtig sind aus Teamsicht „weiche Faktoren“ wie das Auftreten eines potenziellen Mitarbeiters, seine Kommunikations- und Integrationsfähigkeit – einfach ob „die Chemie passt“. Hier lauert vermutlich ein ähnliches Missverständnis wie bei „agiler Dokumentation“: Die höhere Wertschätzung von „soft skills“ heißt nicht, dass erfolgreich abgelegte Prüfungen überhaupt keinen Wert besitzen.

Der weltweite ISTQB®-Standard ([7]) zur Ausbildung von Softwaretestern ist nicht zuletzt deswegen eine Erfolgsgeschichte geworden, weil er ein einheitliches Vokabular im Test zu etablieren half. Dies kann auch in agilen Teams kein Schaden sein. (Und jeder ISTQB®-zertifizierte Tester wird in einem Projekt oder Unternehmen, in dem sich andere Begriffe etabliert haben, auf diese umschwenken und zu einem Mapping in der Lage sein. Auch hier gilt also nicht, dass ISTQB® ein Standard ist, dem sich alles und jeder unterzuordnen hat.)

Außerdem stellen wir fest, dass „agiles Testen“, wie bereits ausgeführt, keine neuen Testmethoden etabliert hat. Exploratives Testen etwa war als Methode schon vielen nichtagilen Projekten ein Begriff, bevor Scrum & Co ihren Siegeszug antraten. (Dass die Testmethode in agilen Projekten so populär wurde, liegt schlicht daran, dass sie gut zu agilen Zielen passt: Sie ist schlank und liefert schnelles Feedback, in dem sie sich ausschließlich auf das Ziel „Fehler finden“ konzentriert. Sie liefert aber meistens keinerlei systematische Qualitätsmetriken wie etwa Anforderungsabdeckung!) Ergänzend zur oben genannten Definition des agilen Testens beschreiben wir in unseren Kundenprojekten die Veränderung für die Tester, die die Agilität mit sich bringt, gerne mit dem Slogan: „Change your mindset – keep your method set!“ Das Wissen um geeignete Testentwurfsverfahren entfaltet unzweifelhaft auch in agilen Projekten seinen Nutzen.

Die Veränderungen, vor denen Tester und Qualitätsmanager in agilen Projekten stehen, sind aber ebenso unzweifelhaft zahlreich und signifikant. Darauf hat das ISTQB®-Ausbildungsschema reagiert und im Jahr 2014 einen Lehrplan zum sogenannten „Agile Tester – Foundation Level Extension“ vorgelegt, aus dem wir in diesem Artikel bereits zitiert haben. Dieser setzt auf der Grundausbildung, dem „Certified Tester Foundation Level“, auf und vermittelt Spezialthemen rund um das agile Vorgehen. Offenbar wird das Schema vom Markt gut angenommen, aber wie immer gilt: Schulung ist das eine, Erfahrung ist das andere. Für Tester, Testmanager und Qualitätsmanager, die zum ersten Mal mit agilem Vorgehen in Berührung kommen, ist diese Ausbildung vielleicht notwendig, aber sicher nicht hinreichend. Umgekehrt richtet sie aber auch keinen Schaden an, sondern vermittelt wertvolle Anregungen, wie man Test- und QS-Themen erfolgreich in agilen Teams integriert.

Fazit

Spätestens mit den Fragestellungen rund um „agile Unternehmen“ ist Agilität auch im Qualitätsmanagement als Thema angekommen. Freiheitsgrade, die agile Teams einfordern, kollidieren möglicherweise mit QM-Vorgaben, aber diese Konflikte lassen sich nach unserer Erfahrung meistens befriedigend auflösen. Wie dies jetzt und künftig gelingen kann, haben wir in diesem Beitrag dargelegt. Letztlich verfolgen beide Parteien die gleichen Ziele: reife Prozesse und hohe Produktqualität. Nur die Wege dorthin unterscheiden sich, denn sie kommen aus unterschiedlichen Richtungen - agile Teams „bottom up“ und QM-Stellen „top down“. Unverzichtbar ist auf beiden Seiten die Bereitschaft, die Sichtweisen und Methoden der jeweils anderen Fraktion undogmatisch zu betrachten. Letztlich hilft auch hier einer der vier agilen Werte des zu Beginn zitierten Manifests: „collaboration“ ist demnach ein höherer Wert als die sture „contract negotiation“. Damit einher geht der generelle Appell an klassische Management-Rollen, nicht nur zu fordern, sondern auch zu fördern: In agilen Unternehmen ist zunehmend der „Manager Teacher“ gefragt, der konstruktiv hilft, auch in einem flexibler gestalteten Prozess die Qualitätsziele zu erreichen.

Referenzen

[1] Agiles Manifest: agilemanifesto.org

[2] Offizieller SCRUM-Guide: www.scrumguides.org

[3] Spillner, A.; Vosseberg, K.; Winter, M.; Haberl, P.: „Wie agil ist die Praxis? Auswertung der Umfrage Softwaretest in der Praxis“, in: iX Developer 1/2012 - Programmieren heute

[4] Brandes, C.; Roßner, T.: „Geht doch! Wie Agilität und Prozessreifegrad-Modelle zusammenpassen“, in: iX Developer 3/2013 - Bessere Software

[5] Schooenderwoert, N.v.: „No Bugs“, Keynote auf dem QS-Tag 2011 in Nürnberg. (http://www.qs-tag.de/fileadmin/Resources/QSTag/Archive/files/2011/abstracts/abstract-vanschooenderwoert/index.html#c12417)

[6] Wikipedia-Seite zu „Agiles Testen“: de.wikipedia.org/wiki/Agiles_Testen

[7] ISTQB® Certfied Tester – Ausbildungsschema: www.istqb.org

[8] Portal zur ISO 29119: www.softwaretestingstandard.org

[9] Definition of Test: www.agile-softwaretesting.de

[10] Agile Testquadranten: lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/

[11] Adzic, G.: „Specification By Example“. Manning, 2011.

[12] Martin, R.C.: „Clean Code“. mitp, 2009

[13] Lang, M.; Scherber, S. (Hrsg.): „Perfekte Softwareentwicklung“. Düsseldorf, Symposion Publishing, 2013

[14] SaFE: www.scaledagileframework.com

[15] LeSS: less.works

[16] Nexus: www.scrum.org/Resources/The-Nexus-Guide

[17] Wikipedia-Seite zu „Servant Leadership“: de.wikipedia.org/wiki/Servant_Leadership

[18] Mathis, C.: „SAFE – Lean und Agile in großen Unternehmen skalieren”, dpunkt.verlag, 2016

[19] ISTQB®-Lehrplan zur “Agile Tester Extension”: www.istqb.org/downloads/syllabi/agile-tester-extension-syllabus.html

[20] Discover to Deliver: www.discovertodeliver.com

[21] Brandes, C.: “Modellbasiertes Testen: Testkosten senken, Qualität steigern”, in: it-management Juli-August 2015

[22] Brandes, C.; Okujava, S.; Baier, J.: „Architektur und Testbarkeit: Eine Checkliste (nicht nur) für Softwarearchitekten“, in: OBJEKTspektrum 01/2016 und 02/2016

[23] Studie „The Future of Testing“: www.imbus.de/unternehmen/forschung/the-future-of-testing