

Eine der wichtigsten Erkenntnisse der letzten Jahre aus agilen Kundenprojekten ist, dass die meisten agil arbeitenden Projekte entwicklergetrieben sind (so sind Scrum-Master häufig 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.

2.1 Qualitätschancen, 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. Nancy van Schooenderwoert (2011) 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.“ 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:

Agiles Testen

„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.“ (Wikipedia)

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.

Als Best Practices für Test und Qualitätssicherung in agilen Projekten können angeführt werden:

2.2 Agile Teststrategie

Im ISTQB-Standard (ISTQB) 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]).

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.

Trotzdem: Auch agile Teams benötigen eine Teststrategie! Diese muss aber NICHT zwingend aussehen wie ein Testkonzept nach IEEE 829, das Gefahr läuft, als „Schranksware“ ungelesen in einem Regal zu verstauben. Es gibt Formate, die zu agilen Teams kompatibler sind – etwa eine „Definition of Test“ auf Grundlage der agilen Testquadranten nach (Marick) (siehe Abb. 2.1). 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.

Anders gesagt: Alles, was etwa der ISTQB-Standard an guter und bewährter Testpraxis empfiehlt (Einsatz von Testmethoden, risikobasiertes Testen, usw.), lässt sich problemlos auch in ein agiles Set-up 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,

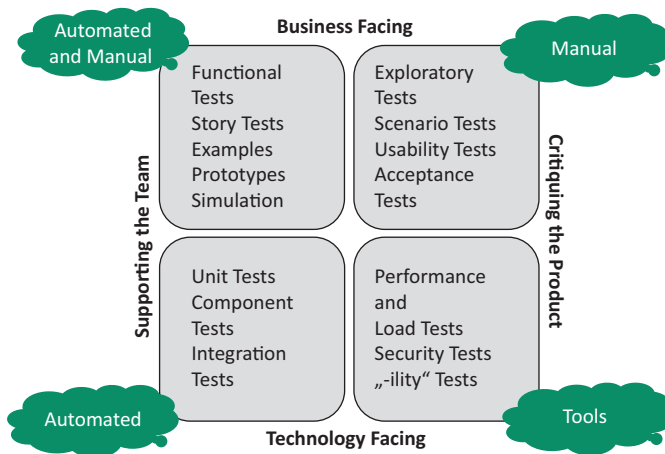


Abb. 2.1 Agile Testquadranten. (Nach B. Marick)

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. Man sollte sich aber dem Umstand, dass QM-Vorgaben von agilen Teams auch mal kritisch hinterfragt werden, ebenso unvoreingenommen stellen.

2.3 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“ (Adzic 2011) 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 Anforderungsautor.

2.4 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 and Replay“ oder „Record and Playback“), wird man mit hoher Wahrscheinlichkeit mittelfristig vor massiven Wartungs- und Pflegeproblemen 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“ (Martin 2009) oder „Collective Code Ownership“ (gemeinsame Code-Verantwortung) sollten genauso für den Testcode wie den Programmcode gelten.

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 Abb. 2.2). 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.

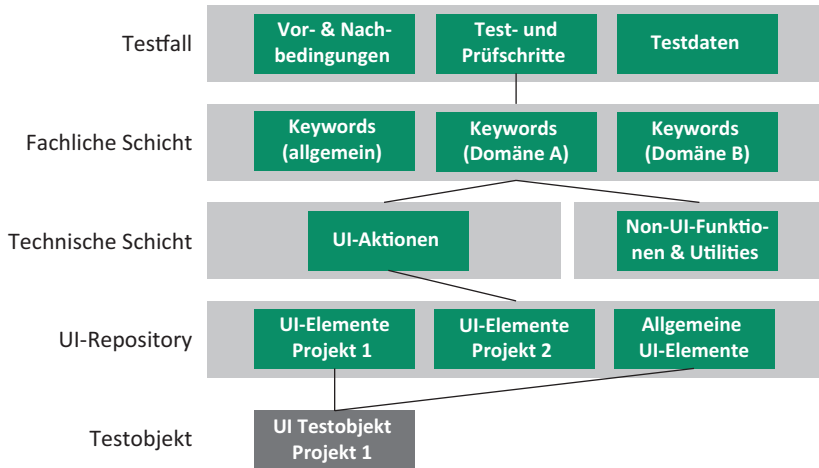


Abb. 2.2 Reife Testautomatisierung trennt modular zwischen Testlogik und Implementierung

2.5 Continuous Integration

Wenn Continuous Integration (CI) zur Sprache kommt, gilt der erste Gedanke im Regelfall den Werkzeugen. Die meisten agilen Teams und Projekte haben entweder etablierte Toolketten für CI oder sind gerade dabei, sie einzuführen. Grundlage ist ein Tool zum Konfigurationsmanagement und zur Versionsverwaltung. Neben einem CI-Server umfasst die Minimalmenge für CI typischerweise Tools für Whitebox-Tests und Unittests des Entwicklungscodes, den Compiler, und natürlich Tools für den automatischen Build-Prozess. Beim Aufbau eines solchen CI-Toolings kann man verschiedene Hersteller- oder Open-Source-Produkte zu einer individualisierten/passgenauen Lösung zusammensetzen („Best of Breed“-Ansatz) oder alternativ die Produkte aus der Hand eines einzigen ALM-Herstellers („Application Lifecycle Management“) für sich zu nutzen versuchen.

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.

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“ (Lang, M. und Scherber 2013, 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, sodass 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 Kümmerner findet, wird auch der Aufwand für die Fehlersuche in den automatischen Tests leichter handhabbar.

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.

Die Frage, wie viel 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

Feature-Druck zum Opfer zu fallen. Theoretisch schätzt ja das Team seine Aufwände inklusive aller nötigen qualitätssichernden Maßnahmen ab, sodass 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, obwohl sie wissen, dass man diesem nicht nachgeben und dabei Qualitätsziele vernachlässigen sollte. 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. Die Empfehlung lautet hier: 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“, wie wir später noch erläutern werden.

Am Ende jeder CI-Toolkette (siehe Abb. 2.3) findet sich ein Berichtswesen (Reporting). Oft gibt es große Bildschirme in Teamräumen, auf denen zu sehen

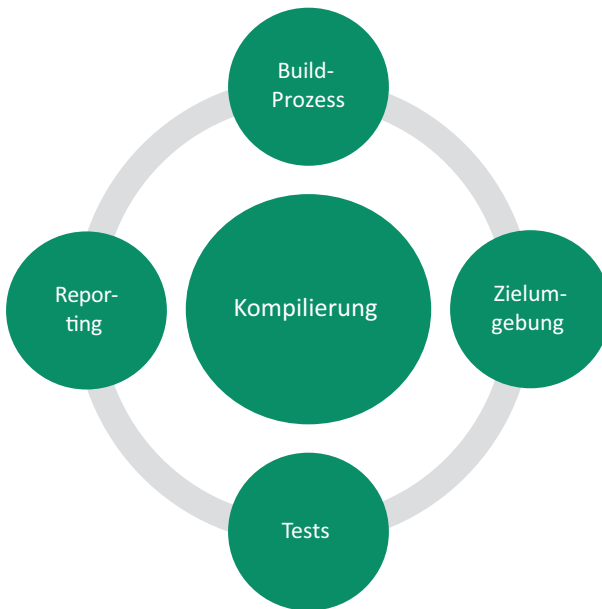


Abb. 2.3 Adressierte Themen in einer Werkzeugkette zur kontinuierlichen Integration

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.

Qualitätsmanagement in agilen IT-Projekten – quo
vadis?

HMD Best Paper Award 2016

Brandes, C.; Heller, M.

2017, IX, 25 S. 5 Abb., Softcover

ISBN: 978-3-658-18084-3