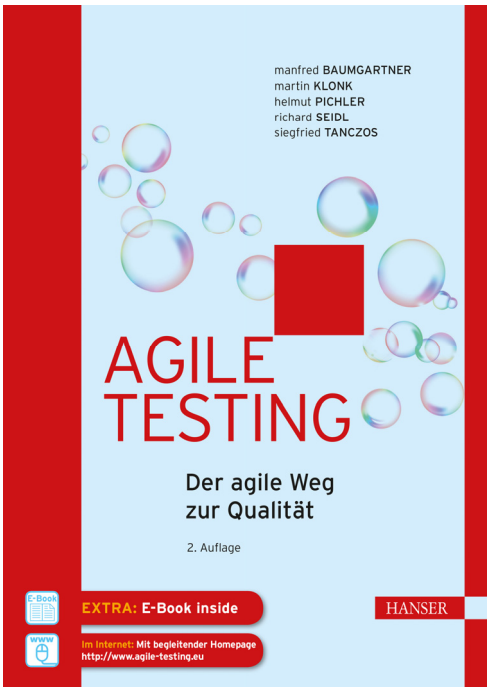
HANSER



Leseprobe

zu

„Agile Testing“ (2. Auflage)

von Manfred Baumgartner, Martin Klonk, Helmut Pichler, Richard Seidl, Siegfried Tanczos

ISBN (Buch): 978-3-446-45292-3
ISBN (E-Book): 978-3-446-45298-5

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/9783446452923
sowie im Buchhandel

Inhalt

Geleitwort

Im Winter 2001 wurde auf einer entlegenen Skihütte im Staate Utah von einer verschworenen kleinen Clique bekannter Software-Entwickler zu einer Revolution in der Software-Welt aufgerufen. Sie erschufen das Agile Manifest. Mit diesem Manifest legte die Gruppe fest, was sie ohnehin schon mit Extreme Programmierung praktizierten. Aber mit der schriftlichen Formulierung gelang ihnen ein publizistischer Coup, mit dem sie weltweit Aufmerksamkeit für ihr Anliegen gewannen. Die Entwicklungsexperten, die sich dort versammelten, hatten es satt, sich von starren Prozessregeln, unsinnigen bürokratischen Richtlinien und weltfremden Vorgehensweisen der damaligen Software-Engineering-Disziplin gängeln zu lassen. Sie erkannten, dass das monotone Arbeiten nach Vorschrift in der neuen schnelllebigen Zeit überholt war. Sie wollten sich von den Fesseln der Projektbürokratie befreien, um zusammen mit den Anwendern Software nach Bedarf zu entwickeln. An die Stelle der bisher schwerfälligen, phasenorientierten, dokumentengesteuerten Software-Entwicklung sollte eine flexible, menschengesteuerte Entwicklung mit kleinen, überschaubaren Schritten treten. Die agile Software-Entwicklung sollte die Vorgehensweise des neuen Jahrhunderts sein.

Im Vordergrund der agilen Entwicklung steht nicht das Projekt, sondern das Produkt. Da Software-Entwicklung immer mehr zu einer Expedition ins Ungewisse wurde, sollte das Produkt Stück für Stück in kleinen Inkrementen entstehen. Statt lange Absichtserklärungen bzw. Anforderungsdokumente zu schreiben, über Dinge, über die man zu dem Zeitpunkt gar nicht Bescheid wissen konnte, sollte man lieber gleich etwas programmieren, um eine schnelle Rückkopplung von dem künftigen Benutzer zu bekommen. Es soll nicht mehr Monate oder gar Jahre dauern, bis sich herausstellt, dass sich das Projekt auf einem Irrweg befindet oder das Projektteam überfordert ist. Dies sollte sich schon nach wenigen Wochen erweisen.

Das Grundprinzip der agilen Entwicklung ist also die inkrementelle Lieferung. Ein Software-System soll stückweise fertiggestellt werden. Damit hat der Benutzervertreter im Team die Möglichkeit mitzuwirken. Nach jeder neuen Auslieferung kann er das ausgelieferte Zwischenprodukt mit seinen Vorstellungen vergleichen. Der Test ist dadurch in das Verfahren eingebaut. Die Software wird von Anfang an dauernd getestet. Ob da ein Tester mit im Spiel ist, wurde zunächst offengelassen. Die Verfasser des agilen Manifests waren gegen eine strenge Arbeitsteilung. Die Aufteilung in Analytiker, Designer, Entwickler, Tester und Manager war ihnen zu künstlich und verursachte zu viele Reibungsverluste. Natürlich soll das Projektteam diese Fähigkeiten besitzen, aber die Rollen innerhalb des Teams

sollten austauschbar sein. Das Entwicklungsteam soll als Ganzes für alles verantwortlich sein. Erst durch die Beiträge von Crispin und Gregory hat sich die Rolle des Testers im Team herausgestellt. Die beiden haben sich dafür eingesetzt, dass sich jemand im Team um die Belange der Qualität kümmert.

Software-Entwicklung verlangt sowohl Kreativität als auch Disziplin. Gegen Ende des letzten Jahrhunderts haben die Befürworter von Ordnung und Disziplin die Oberhand gehabt und mit ihren strengen Prozessen und Qualitätssicherungsmaßnahmen die Kreativität der Entwickler vereitelt. Wenn übertrieben wird, kehrt sich alles ins Gegenteil um. Mit dem Qualitätsmanagement wurde zu viel des Guten getan. Die Gegenreaktion war die agile Bewegung, die darauf ausgerichtet war, mehr Spontaneität und Kreativität in die Software-Entwicklung zurückzubringen. Dies ist durchaus zu begrüßen, aber auch hiermit darf nicht übertrieben werden. Man braucht einen Gegenpol zu der sprudelnden Kreativität der Benutzer und Entwickler. Dieser Gegenpol ist der Tester im Team.

In jedes Entwicklungsteam gehört mindestens ein Tester, um die Belange der Qualität zu vertreten. Der Tester oder die Testerin sorgt dafür, dass das entstehende Produkt sauber bleibt und die vereinbarten Qualitätskriterien erfüllt. In dem Drang, schneller voranzukommen, geraten die nichtfunktionalen Qualitätsanforderungen gegenüber den funktionalen Anforderungen allzu leicht ins Hintertreffen. Es ist der Job des Testers, dafür zu sorgen, dass ein Gleichgewicht zwischen Produktivität und Qualität bewahrt wird. Der Tester ist sozusagen der gute Geist, der das Team davon abhält, Fortschritt auf Kosten der Qualität zu erringen. In jedem Release soll nicht nur mehr Funktionalität, sondern auch mehr Qualität angestrebt werden. Der Code soll regelmäßig bereinigt bzw. refaktoriert, nachdokumentiert und von Mängeln befreit werden. Dass dies tatsächlich geschieht, ist die Aufgabe des Testers.

Natürlich hat die agile Projektorganisation auch Folgen für den Test und die Qualitätssicherung. Die Verantwortlichen für die Qualitätssicherung sitzen nicht mehr in einer entfernten Dienststelle, von wo aus sie die Projekte überwachen, die Projektergebnisse zwischen den Phasen kontrollieren und in der letzten Phase das Produkt durchtesten. Sie sind in den Entwicklungsteams fest integriert, wo sie ständig prüfen und testen. Es obliegt ihnen, auf Mängel in der Architektur sowie im Code hinzuweisen und Fehler im Verhalten des Systems aufzudecken. Ihre Rolle ist jedoch nicht mehr die des lästigen Kontrolleurs, sondern vielmehr die des Freund und Helfers. Sie weisen auf die Probleme hin und helfen den Entwicklern, die Qualität ihrer Software auf den erforderlichen Stand zu bringen. Im Gegensatz zu dem, was manche behaupten – nämlich, dass Tester in agilen Projekten nicht mehr nötig sind –, ist ihre Rolle wichtiger denn je. Ohne ihren Beitrag wachsen die technischen Schulden und diese bringen das Projekt früher oder später zum Stillstand.

Das vorliegende Buch beschreibt den agilen Test in zehn Kapiteln. Das erste Kapitel schildert den kulturellen Wandel, den die agile Entwicklung mit sich gebracht hat. Mit dem agilen Manifest wurden die Weichen für eine Neuordnung der IT-Projektlandschaft gesetzt. Es soll nicht mehr starr nach Phasenkonzept, sondern flexibel in kleinen Iterationen entwickelt werden. Nach jeder Iteration soll ein lauffähiges Teilprodukt vorzuweisen sein. Damit werden Lösungswege erforscht und Probleme früh erkannt. Die Rolle der Qualitätssicherung wandelt sich. Statt als externe Instanz auf die Projekte von außen zu wirken, sind die Tester im Projekt eingebettet, um ihre Tests sofort vor Ort als Begleiter der Entwicklung durchzuführen. Natürlich müssen die Anwenderbetriebe ihre Managementstrukturen ent-

sprechend anpassen: Statt abseits auf ein Endergebnis zu warten, sind die Anwender aufgefordert, im Projekt aktiv mitzumachen und die Entwicklung über ihre Anforderungen, sprich „Stories“, zu steuern. Auf der Entwicklungsseite arbeiten sie mit den Entwicklern zusammen, um die gewünschte Funktionalität zu analysieren und zu spezifizieren. Auf der Testseite arbeiten sie mit den Testern zusammen, um zu sichern, dass das Produkt ihren Erwartungen entspricht.

Letztendlich müssen sich alle umstellen – Entwickler, Tester und Anwender –, um das gemeinsame Ziel zu erreichen. Manch traditionelle Rolle fällt dabei weg wie die des Projektleiters und des Testmanagers. Dafür gibt es neue Rollen wie die des Scrum Masters und des Teamtesters. Das Projektmanagement im klassischen Sinne findet nicht mehr statt. Jedes Team managt sich selbst. Die IT-Welt ändert sich und mit ihr die Art und Weise, wie Menschen Software entwickeln. Es gilt also, diesem neuen Zustand gerecht zu werden. Der Weg dazu wird hier im ersten Kapitel geschildert.

Im zweiten Kapitel über agile Vorgehensmodelle gehen die Autoren auf die Rolle der Qualitätssicherung in einem agilen Entwicklungsprojekt ein. Dabei scheuen sie sich nicht, die verschiedenen Zielkonflikte, z. B. zwischen Qualität und Termintreue, zwischen Qualität und Budget und zwischen Qualität und Funktionalität objektiv zu betrachten. Die Versöhnung dieser Zielkonflikte ist eine Herausforderung des agilen Tests.

Im Gegensatz zur landläufigen Meinung, dass in den agilen Projekten weniger getestet werden muss, wird hier gefordert, noch mehr zu testen. Test-Driven Development (TDD) soll nicht nur für den Unit Test, sondern auch für den Integrations- und Systemtest gelten, nach der Devise: erst die Testfälle, dann der Code. Hier heißt es: erst die Testspezifikation, dann die Implementierung. Dabei spielt die Testautomation eine entscheidende Rolle. Erst wenn der Test automatisiert ist, kann in der erforderlichen Geschwindigkeit die erforderliche Qualität erreicht werden. Das ganze Team soll sich an dem Automatisierungsprozess beteiligen, denn der Tester allein kann es nicht schaffen. Er braucht die Unterstützung der Entwickler, denn er hat auch andere Aufgaben zu erledigen. Neben dem Test wird auch die Durchführung von Audits zu bestimmten Zeitpunkten in der Entstehung des Software-Produkts gefordert. Die Audits zielen darauf hin, Schwachstellen und Missstände in der Software zu enthüllen. Der Zeitpunkt dafür ergibt sich nach jedem Sprint in einem Scrum-Projekt. Aufgrund der Ergebnisse der Audits können die Prioritäten für den nächsten Sprint gesetzt werden. Diese kurzen Audits bzw. Momentaufnahmen der Produktqualität können durch QS-Experten von außen in Zusammenarbeit mit dem Team durchgeführt werden. Der Zweck ist nicht zu sehr, das Projekt durch Kritik aufzuhalten, sondern dem Team zu helfen, Risiken rechtzeitig zu erkennen.

Zusätzlich zum Scrum-Prozess behandelt das zweite Kapitel auch Kanban und den schlanken Software-Entwicklungsprozess (Lean Software). Der Leser bekommt etliche Hinweise, wie Qualitätssicherung in diesen Verfahren einzubauen ist, und zwar mithilfe von Beispielen aus der Projektpraxis.

Das dritte Kapitel behandelt die agile Testorganisation bzw. den Standort der Tester in einem agilen Umfeld. Zu diesem Thema herrschen sehr unterschiedliche Ansichten. Die Autoren stellen die Frage, welcher Test wofür gut ist. Zur Beantwortung dieser Frage werden die vier Testquadranten von Crispin und Gregory aufgeführt. Zum einen wird gefragt, ob der Test fachlich oder technisch ist, zum anderen, ob er auf das Produkt oder die Umgebung bezogen ist. Daraus ergeben sich die vier Testarten:

1. Unit- und Komponententest = technisch/produktbezogen
2. Funktionaler Test = fachlich/produktbezogen
3. Explorativer Test = fachlich/umgebungsbezogen
4. Nichtfunktionaler Test = technisch/umgebungsbezogen

Für die Erläuterung dieser Testansätze werden wiederum Beispiele aus der Testpraxis angeführt, die zeigen, welche Testart welchem Zweck dient.

Zum Schluss des Kapitels gehen die Autoren auf das agile Ausbaumodell von Scott Ambler ein und betonen, wie wichtig es ist, den Testprozess beliebig ausbauen zu können. Es gibt Kernaktivitäten, die auf jeden Fall stattfinden müssen, und Randaktivitäten, die je nach Ausbaustufe hinzukommen. Somit gibt es nicht die eine Organisationsform, sondern viele mögliche Organisationsformen in Abhängigkeit von der Produktart und den Projektbedingungen.

Wesentlich für die Wahl der geeigneten Organisationsform sind die Umgebung, in der das Projekt stattfindet, sowie die Produkteigenschaften wie Größe, Komplexität und Qualität. Jedenfalls darf man das Hauptziel, nämlich die Unterstützung der Entwickler, nicht aus den Augen verlieren. Alle Testansätze haben dem Ziel zu dienen, Probleme so schnell und so gründlich wie möglich aufzudecken und den Entwicklern auf eine nicht aufdringliche Art und Weise mitzuteilen. Sollten mehrere agile Projekte nebeneinander laufen, empfehlen die Autoren, ein Test Competence Center einzurichten. Die Aufgabe dieser Instanz ist es, die Teams in Fragen des Tests zu betreuen, z. B. welche Methoden, Techniken und Werkzeuge sie nutzen sollten. Am Ende des Kapitels werden zwei Fallstudien in Testorganisation angeführt, eine aus dem Telekommunikationsbereich und eine aus dem Gesundheitsbereich. In beiden Studien richtet sich die Testorganisation nach der Projektstruktur und nach den jeweiligen Qualitätszielen.

In Kapitel 4, „Die Rolle des Testers in agilen Projekten“, stellt sich die Frage, ob der agile Tester Generalist oder Spezialist sein sollte. Die Antwort lautet, wie so oft in der Literatur zur agilen Entwicklung: sowohl als auch. Es hängt von der Situation ab. Es gibt Situationen wie zu Beginn des Releases, wenn der Tester mit dem Benutzer über die Akzeptanzkriterien verhandelt, in denen der Tester neben allgemeinen auch fachliche Kenntnisse braucht. Es gibt andere Situationen wie am Ende des Releases, wenn Tester mit automatisierten Testwerkzeugen umgehen müssen, in denen der Tester spezielle technische Kenntnisse braucht. Ein agiler Tester muss eben viele Rollen übernehmen können. Was Rollen betrifft, ist es am wichtigsten, dass der Tester sich in das Team als Teamplayer einfügt, egal, welche Rolle er im Moment zu übernehmen hat. Soft Skills sind gefragt. Auf jeden Fall ist der Tester der Anwalt der Qualität und er hat dafür zu sorgen, dass die Qualität bewahrt wird, auch wenn die Zeit knapp wird. Dazu muss er sich an allen Diskussionen rund um die Produktqualität beteiligen, während er gleichzeitig die Software prüft und testet. Er soll entscheiden, Probleme rechtzeitig aufdecken und dafür sorgen, dass sie frühestmöglich beseitigt werden. Natürlich kann er dies nicht alleine schaffen, er braucht die anderen Teammitglieder dazu. Darum muss er als eine Art Qualitätsberater agieren und seinen Teamkollegen dazu verhelfen, ihre eigenen Probleme zu erkennen und zu lösen. Die Qualität der Software ist schließlich eine Angelegenheit des Teams in seiner Gesamtheit, das Team haftet für die Qualität des Produkts.

Im Zusammenhang mit der Rolle des Testers in einem agilen Team geht das Kapitel auf das Erfahrungsprofil ein und stellt die Frage, ob agil nur etwas für junge Mitarbeiter ist. Wie

sehen die Karrierebilder in der agilen Welt aus? Tatsache ist, dass es in der agilen Entwicklung keine festen Rollen mehr gibt. Die Rollen wechseln je nach Situation, auch die des Testers. Mitarbeiter mit ausschließlicher Erfahrung in den traditionellen Entwicklungsmethoden können sich nicht mehr in traditionelle Rollenbilder zurückziehen. Es bleibt ihnen nur übrig, sich anzupassen. Das dürfte nicht jedem (älteren) Mitarbeiter leicht fallen. Der Autor Manfred Baumgartner schlägt ein Trainingsprogramm „Agilizing 40+“ vor, das sie auf die Tätigkeit als agiler Tester vorbereitet. Er verweist auf positive Erfahrungen damit und schließt mit einer zuversichtlichen Note ab, dass flexible Mitarbeiter, ob alt oder jung, in die Rolle eines agilen Testers hineinwachsen können. Ob sie sich diesem Stress wirklich aussetzen wollen, ist eine andere Frage.

In Kapitel 5 wenden sich die Autoren den Methoden und Techniken des agilen Testens zu. Hierbei stellen sie die Unterschiede zum konventionellen, phasenorientierten Testen in den Vordergrund. Das beginnt schon mit der Testplanung, wobei der Plan viel unverbindlicher ist. Er soll flexibel bleiben und sich leicht fortschreiben lassen. Der agile Test ist viel mehr mit der Entwicklung verflochten und darf nicht mehr getrennt als Projekt im Projekt betrachtet werden. Es soll zwar mindestens einen Tester in jedem Entwicklungsteam geben, aber er ist dort voll integriert. Er ist nur dem Team Rechenschaft schuldig. Möglicherweise gibt es irgendwo einen projektübergeordneten Testmanager, der als Bezugsperson für die Tester in mehreren Teams dient, aber er darf keinen Einfluss auf die Arbeit innerhalb des Teams haben. Er übt allenfalls eine Beraterfunktion aus. Die bisherige Planung, Organisation und Steuerung eines separaten Testteams unter der Leitung eines Teammanagers fällt weg. Sie passt nicht zur agilen Philosophie der Teamarbeit.

Was die Testmethoden anbetrifft, werden jene Methoden betont, die zur agilen Vorgehensweise am besten passen – risikobasiertes Testen, wertgetriebenes Testen, exploratives Testen, Session-basiertes Testen und abnahmetestgetriebene Entwicklung. Konventionelle Testtechniken wie Äquivalenzklassenbildung, Grenzwertanalyse, Zustandsanalyse und Entscheidungstabellen bzw. -bäume gelten nach wie vor, nur in einem anderen Zusammenhang. Sie sollten ohnehin in die Testwerkzeuge eingebaut werden. Hervorgehoben wird die Bedeutung der Testwiederverwendung und der Testwiederholung. Sämtliche Techniken müssen diese Kriterien erfüllen. Der Integrationstest ist eine nie endende Geschichte und der Abnahmetest wird ständig wiederholt. Die zyklische Natur eines agilen Projekts erzeugt eine Umdefinition der Testausgangskriterien. Eigentlich ist der Test nie zu Ende – solange das Produkt noch weiterwächst. Irgendwann wird die Entwicklung als beendet deklariert und das Produkt geht in die Wartung.

In Kapitel 6, „Agile Testdokumentation“, beschreiben die Autoren, welche Dokumente von den Testern in einem agilen Projekt noch zu erstellen sind. Dazu gehören eine testbare Anforderungsspezifikation aus den User Stories, ein Testdesign, eine Benutzerdokumentation und Testberichte. Die Testfälle gelten nicht als Dokumentation, sondern als Testware. Ein besonderes Anliegen der agilen Entwicklung ist, die Dokumentation auf ein Mindestmaß zu reduzieren. Früher übertrieb man es in der Tat mit der Dokumentation. In einem agilen Entwicklungsprojekt wird nur das dokumentiert, was unbedingt notwendig ist. Ob eine Teststrategie oder ein Testdesign absolut notwendig ist, bleibt dahingestellt. Testfälle sind unentbehrlich, aber sie gehören zum Software-Produkt ebenso wie der Code. Deshalb gelten sie nicht als Dokumentation.

Das wichtigste Dokument ist die Anforderungsspezifikation, die aus den User Stories hervorgeht. Sie dient als Basis für den Test, das sogenannte Testorakel. Aus ihr werden die Testfälle abgeleitet und gegen sie wird getestet. Sie enthält auch die Abnahmekriterien. Die einzigen wirklich erforderlichen Testberichte sind der Testüberdeckungsbericht und der Fehlerbericht. Der Testüberdeckungsbericht belegt, was getestet wurde und was nicht. Die Tester brauchen dieses Dokument als Nachweis dafür, dass sie ausreichend getestet haben. Der Benutzer braucht es, um Vertrauen in das Produkt zu gewinnen. Der Fehlerbericht hält fest, welche Abweichungen eingetreten sind und was mit ihnen geschieht. Diese beiden Berichte sind die besten Indikatoren für den Stand des Tests.

Schließlich sind die Tester prädestiniert, das Benutzerhandbuch zu schreiben, weil sie das System in seiner Gesamtheit am besten kennen und wissen, wie man damit umgeht. Es muss jemand die Bedienungsanleitung verfassen und der Tester ist der geeignete Kandidat dafür. Er sorgt dafür, dass dieses Dokument nach jedem Release fortgeschrieben wird. Ansonsten folgt das Buch dem agilen Prinzip, die Dokumentation auf das Wesentliche zu beschränken. Das, was noch an Testdokumentation bisher geliefert wurde, ist in einer Umgebung verzichtbar, in der die verbale Kommunikation dominiert. Hauptsache, es gibt immer eine solide Anforderungsspezifikation und eine verständliche Benutzerdokumentation. Eine strukturierte, semiformale Anforderungsspezifikation bildet die Basis für den Test und auf eine Benutzeranleitung möchte kein Benutzer verzichten.

Die Kapitel 7 und 8 befassen sich mit dem wichtigen Thema „Testautomation“. Testautomation ist bei der agilen Entwicklung besonders wichtig, weil sie das Hauptinstrument der Projektbeschleunigung ist. Nur durch Automation kann der Testaufwand auf ein vertretbares Maß bei gleichzeitiger Erhaltung der Produktqualität reduziert werden. Die Autoren unterscheiden hier zwischen Unit Test, Komponentenintegrationstest und Systemtest. Der Unit Test wird am Beispiel von JUnit ausführlich dargestellt. Darin wird gezeigt, wie der Entwickler testgetrieben zu arbeiten hat, wie er seine Testfälle aufbaut und wie er die Testüberdeckung misst. Der Komponentenintegrationstest wird anhand des Apache-Maven-Integrationsservers erläutert. Hier kommt es darauf an, die Schnittstellen der integrierten Komponenten zu den noch nicht vorhandenen Komponenten durch Platzhalter zu simulieren. Der Systemtest wird durch einen fachlichen Test mit FitNesse beschrieben. Das Wichtigste hier ist die Verfassung der Testfälle in Testskripten, die beliebig ausgebaut und wiederholt ausgeführt werden können. Die Autoren betonen außerdem, wie wichtig es ist, die Testware – Testfälle, Testskripte, Testdaten usw. – bequem und sicher verwalten zu können, damit der Test möglichst reibungslos abläuft. Dafür werden auch Werkzeuge gebraucht.

Kapitel 8 ergänzt die Behandlung der Testautomation mit konkreten Beispielen aus der Testautomationspraxis. Als Erstes wird das Tool CA Agile Central beschrieben, das den agilen Lebenszyklus von der Verwaltung der Stories bis hin zur Fehlerverwaltung unterstützt. Der agile Tester kann dieses Tool in seinem Test planen und steuern. Eine Alternative zu CA Agile Central ist das Tool Polarion, das für die Erfassung und Priorisierung der Testfälle sowie auch für die Verfolgung der Fehler besonders geeignet ist. Weitere Testplanungs- und Verfolgungswerkzeuge sind die Tools Bug Genie, das die Testaufwandsschätzung besonders unterstützt, und Atlassian JIRA, das eine umfangreiche Fehleranalyse anbietet, sowie Microsofts TSF Testmanager.

Für den Tester in einem agilen Projekt kommt es vor allem auf den fortwährenden Integrationstest an. Er muss die letzten Komponenten möglichst schnell mit den Komponenten des

letzten Releases integrieren und bestätigen, dass sie reibungslos zusammenwirken. Dazu muss er nicht nur über die Benutzerschnittstelle, sondern auch über die internen Systemschnittstellen testen. Mit Tosca lassen sich sowohl externe als auch interne Schnittstellen generieren, aktualisieren und validieren. Die Testnachrichten werden auf bequeme Weise über die Drag-und-Drop-Technik zusammengestellt. Die Autoren schildern aus ihrer eigenen Projekterfahrung, wie diese Werkzeuge eingesetzt werden und wo ihre Grenzen liegen. Der Tester bekommt viele nützliche Hinweise, die er beim Einsatz der Werkzeuge zu beachten hat.

Das neunte Kapitel des Buchs ist dem Thema „Ausbildung und deren Bedeutung" gewidmet. Die Autoren betonen die Rolle der Mitarbeiterschulung beim Einstieg in die agile Entwicklung. Eine qualifizierte Ausbildung ist für den Erfolg im Umgang mit der neuen Methode unerlässlich und dies gilt besonders für die Tester. Tester in einem agilen Team müssen genau wissen, worauf es ankommt, und das können sie nur über eine geeignete Schulung lernen. Dabei müssen sie sich vor falschen Propheten in Acht nehmen. Vieles, was unter der Bezeichnung „agil" verkauft wird, ist im Grunde genommen nicht agil. Es gibt zwar viele Interpretationen der agilen Vorgehensweise, aber die Qualität des Produkts muss gesichert werden, und dazu braucht man professionelle Tester, die geschult sind, in einem agilen Team mitzuarbeiten. Nützlich ist dabei die vom International Software Quality Institute (iSQI) entwickelte Ausbildung zum Certified Agile Tester. Dieses Ausbildungsprogramm ist speziell auf die Belange des agilen Tests ausgerichtet. Mit dem Erlangen des Certified-Agile-Tester-Zertifikats ist ein Tester gut darauf vorbereitet, in ein agiles Projekt einzusteigen und seinen Mann bzw. seine Frau als nützliches Teammitglied zu stehen.

Zusammenfassend ist zu sagen, dass dieses Buch die wesentlichen Aspekte des agilen Tests abdeckt und eine wertvolle Leitlinie für das Testen in einem agilen Test bietet. Der Leser bekommt viele Anregungen, wie er in einem agilen Projekt vorzugehen hat. Er erfährt, wie der agile Test vorzubereiten, durchzuführen und abzunehmen ist. Als Buch von Testpraktikern geschrieben, hilft es Testern, sich in einer oft verwirrenden agilen Welt zurechtzufinden. Es gibt ihnen eine klare, fundierte Anleitung für die Umsetzung der agilen Grundsätze in der Testpraxis. Es gehört damit in die Bibliothek jeder Organisation, die agile Projekte betreibt.

Harry M. Sneed

Vorwort

Als im Jahre 2001 von einer Gruppe von Software-Ingenieuren in Utah/USA das „Manifesto for Agile Software Development“ unterzeichnet wurde, leitete dies den wohl wesentlichsten Wandel in der Software-Entwicklung seit der Einführung der Objektorientierung Mitte der 80er-Jahre des vorigen Jahrhunderts ein. Das Agile Manifest, quasi die Zehn Gebote der agilen Welt, kann auch als Ausdruck einer Gegenbewegung zu den stark regulierenden Vorgehens- und Planungsmodellen gesehen werden, die ab den späten 80er-Jahren große Verbreitung fanden, wie z. B. PRINCE, das V-Modell oder auch ISO 9001. Diese Modelle versuchten, den bis dahin eher chaotischen und willkürlichen Entwicklungsprozessen durch Planung, Strukturierung der Prozesse und Dokumentation entgegenzuwirken. Das Agile Manifest positioniert sich in den zentralen vier Werthaltungen bewusst zu diesen Aspekten und räumt den agilen Werten – Interaktion, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderungen und letztlich funktionierende Software – eine höhere Relevanz für eine erfolgreiche Software-Entwicklung ein.

Nicht zuletzt durch die Art der Formulierung in Werten und Prinzipien ist der Siegeszug der agilen Software-Entwicklung in den Jahren seit der Veröffentlichung des Agilen Manifests geprägt von vielen Glaubenssätzen, wenn nicht sogar Glaubenskriegen. Wir, die Autoren dieses Buchs, erleben dies nicht das erste Mal. In den Jahrzehnten unserer beruflichen Erfahrung waren wir schon oft mit immer wieder neuen Lösungen für das „Software-Problem“ konfrontiert: strukturierte Programmierung, objektorientierte Programmierung, CASE (Computer-Aided Software Engineering), RUP (Rational Unified Process), V-Modell, ISO 9001, SOA (Service-Oriented Architecture), … – eine lange Liste an Heilsversprechen, immer begleitet von selbsternannten Gurus, manche nennen sich sogar Evangelisten. Und viele dieser Innovationen liefen nach sehr ähnlichen Mustern ab. Während sie sich selbst als *die* Lösung präsentierten oder von deren Verfechtern als rettende Idee verkauft wurden, wurden bisherige Ansätze als falsch oder veraltet abgetan. Es fanden sich auch immer rasch viele Anhänger, die radikalen Ideen oft unreflektiert und fast willenlos folgten, denn die Zahl der Unzufriedenen war groß und ist es noch immer. Hier haben Prediger und Berater, die jeden Hype zur Profilierung nutzen, leichtes Spiel – eine große Gefahr für gute Ideen.

Der letzte Gedanke war auch die zentrale Motivation für das vorliegende Buch. Wir, die Autoren, waren in der Vergangenheit stets unglücklich mit der Art und Weise, wie versucht wurde, neue Ansätze in der Software-Entwicklung dogmatisch umzusetzen. Oft wurde das Kind mit dem Bade ausgeschüttet. Im Gegensatz dazu sehen wir die Veränderungen als Chance für einen Prozess stetiger Verbesserung und Optimierung. Aber gerade als Tester

waren wir Autoren in den letzten Jahren in agilen Projekten immer wieder damit konfrontiert, dass nun all das, was wir uns an Methoden, Techniken, Selbstverständnis als Tester oder Standards (wie etwa die Testprozesse nach ISTQB) angeeignet und erarbeitet haben, nicht mehr gelten sollte. Das mag auch daran liegen, dass es in der Vergangenheit insbesondere Software-Entwickler waren, die die agile Community vorangetrieben haben. Diese Tatsache ist mit ein Grund dafür, dass die Aufgaben und die Rolle des Software-Testers in den agilen Methoden und Projekten oftmals nicht oder nur unklar definiert sind. Dazu tragen auch unterschiedlich interpretierbare Terminologien bei: Spricht Scrum zum Beispiel von einem interdisziplinären Entwicklungsteam, meinen manche, das Team besteht nur mehr aus Entwicklern (im Sinne von Programmierern), die alles machen. Andere wiederum glauben, dass im Test-Driven Development mit der Entwicklung eines automatisierten Unit Test Sets die Testaufgaben in der Entwicklung hinlänglich erfüllt sind und der Rest in der Verantwortung des Anwenders im User Acceptance Test liegt. Wo finden sich also die für uns so gewohnten Testphasen und Teststufen? Wo und wie finden wir uns in agilen Projekten als Tester wieder? Der agile Ansatz stellt uns Tester offenkundig vor mehr Fragen, als er Antworten auf bisherige Problemstellungen liefert.

Genau hier wollen wir mit unserem Buch, das von Testern für Tester geschrieben wurde, ansetzen. In den einzelnen Kapiteln bieten wir Antworten auf zentrale Fragestellungen, die wir in unseren Projekten erlebt haben. Dabei geht es um allgemeine bzw. als geradezu kulturell zu bezeichnende Veränderungsprozesse, um Fragen des Vorgehens und der Organisation im Software-Test, um den Einsatz von Methoden, Techniken und Werkzeugen, im Speziellen um die Testautomatisierung, sowie um die neu zu definierende Rolle des Testers in agilen Projekten und deren Ausbildung. Ein breites Spektrum also, das im Rahmen dieses Buchs sicherlich nicht final und umfassend, aber dennoch, so hoffen wir, für den Leser ideen- und antwortgebend behandelt wird.

Um die beschriebenen Aspekte noch greifbarer zu gestalten, werden die einzelnen Themen dieses Buchs von den Erfahrungen aus konkreten Software-Entwicklungsprojekten verschiedener Unternehmen begleitet.

Die Beispiele sollen illustrieren, dass durchaus unterschiedliche Herangehensweisen zu guten, zu den konkreten Herausforderungen agiler Vorhaben passenden, Lösungen führen können.

In diesem Sinne wünschen wir dem Leser auch viel Erfolg in der Umsetzung hier dargestellter Inhalte in den eigenen Projekten und laden ihn gleichzeitig ein, uns, die Autoren, auf unserer Internetplattform *www.agile-testing.eu* zu besuchen.

Manfred Baumgartner, Wien 2017

Martin Klonk, Wien 2017

Helmut Pichler, Wien 2017

Richard Seidl, Potsdam 2017

Siegfried Tanczos, Wien 2017

Praxisbeispiele

Das Praxisbeispiel EMIL in diesem Buch stammt aus einem Unternehmen der Gesundheitsbranche, das auf 25 Jahre erfolgreiche Produkt- und Software-Entwicklung zurückblickt. Doch mit dem Wachstum der Organisation, den neuen Wünschen der Kunden und den strengeren regulatorischen Anforderungen wurde auch der Bedarf größer, die Entwicklungs- und Testprozesse zu optimieren und effizienter zu gestalten. Die Idee des Wechsels vom traditionellen zum agilen Entwicklungsprozess tauchte hier und da im Unternehmen bereits auf. Mit dem Software-Entwicklungsprojekt EMIL wurde er in Angriff genommen. Ziel des Projekts ist die Neuimplementierung einer Analyse-Software, die zwar weltweit erfolgreich im Einsatz war, aber ebenfalls bereits auf zehn Jahre Historie und wechselnde Entwickler zurückblickte. Insbesondere technologisch und architektonisch ließen sich aktuelle Anforderungen nicht mehr ohne Probleme umsetzen, viele Funktionen wurden im Laufe der Zeit auch nur als „provisorische Balkone“ angebaut – aber nie mehr abgebaut oder integriert. Als grober Zeitrahmen für die Re-Implementierung aller Funktionen der bestehenden Software wurden ca. zweieinhalb Jahre geschätzt. Als die größten Herausforderungen auf dem Weg zur agilen Entwicklung wurden die fehlende Erfahrung in der Zieltechnologie sowie die regulatorischen Anforderungen, die die Gesundheitsbranche mit sich bringt, identifiziert. Die positiven und negativen Erfahrungen, aufgetretene Probleme und die versuchten Lösungsansätze aus den ersten eineinhalb Jahren des Projekts finden sich in diesem Buch und sind in den jeweiligen Kapiteln entsprechend markiert.

Weitere Praxisbeispiele stammen von OTTO. Als Onlinehändler bewegt sich OTTO in einem höchst agilen Marktumfeld und sorgt mit innovativen Technologien für ein positives Einkaufserlebnis auf *otto.de* und in den Spezialshops. Als Teil der Otto Group gehört OTTO zu den erfolgreichsten E-Commerce-Unternehmen Europas und ist Deutschlands größter Onlinehändler für Fashion und Lifestyle im B2C-Bereich. Über 90 Prozent des Gesamtumsatzes wird dabei online erwirtschaftet.

Die Autoren

Manfred Baumgartner

Manfred Baumgartner ist seit 2015 Mitglied der Geschäftsleitung bei ANECON Software Design und Beratung GmbH. Nach Abschluss des Studiums der Informatik an der Technischen Universität Wien war er als Software-Engineer in einem großen Software-Haus im Bankenumfeld und später als Quality Director eines CRM-Lösungsanbieters tätig. Seit 2001 hat er das Beratungs- und Trainingsangebot von ANECON, heute eines der führenden Dienstleistungsunternehmen im Bereich Software-Test in Österreich, auf- und ausgebaut. Seine umfassenden Erfahrungen sowohl in der klassischen als auch in der agilen Software-Entwicklung bringt er als beliebter Sprecher auf renommierten Konferenzen sowie als Autor und Mitautor einschlägiger Fachbücher ein: „Der Systemtest – Von den Anforderungen zum Qualitätsnachweis“ (2008), „Software in Zahlen“ (2010), „Basiswissen Testautomatisierung“ (2012).

Martin Klonk

Martin Klonk ist Kompetenzfeldleiter für Testprozesse und Teststrategie bei der ANECON Software Design und Beratung GmbH. Als ausgebildeter Wirtschaftsingenieur an der Technischen Universität Berlin (und der Université Libre de Bruxelles) startete er 1996 seine Karriere als Softwaretest-Spezialist bei der SQS Software Quality Systems in Köln und München. Martin Klonk arbeitete in verschiedensten Branchen und hat schon in fast allen Bereichen des Software-Tests aktiv mitgearbeitet. Als Mitglied im Austrian Testing Board des ISTQB arbeitete er an Lehrplänen und ihren deutschen Übersetzungen mit und hält auch selbst Trainings. Seit er 2007 in einem agilen Projekt erfolgreiche Teststrategien umsetzen konnte, ist Martin Klonk überzeugter Verfechter agiler Praktiken auch im Test und hat schon mehrfach verschiedene agile Projekte als Testspezialist betreut. Er ist zertifizierter Scrum Master und Trainer für den Certified Agile Tester des iSQI.

Helmut Pichler

Helmut Pichler leitet in der ANECON den Trainingsbereich und ist als Berater für Test und Qualitätsmanagement in beiden Welten (traditionell und agil) unterwegs. Er beobachtete bereits seit Anbeginn auf Konferenzen und in der Community das „Heranwachsen“ der Agilität und ist einer der ersten Trainer zu dem von iSQI entwickelten Ausbildungsprogramm Certified Agile Tester. In der internationalen Community ist oder war er als Country Ambassador nationaler und internationaler Konferenzen wie Agile Testing Days und EuroSTAR tätig. Neben seinen Aufgaben in der ANECON ist Helmut Pichler seit über zwölf Jahren Präsident des Austrian Testing Boards, dem regionalen Vertreter des ISTQB in Österreich, und aktives Mitglied der internationalen Tester Community, wo er gemeinsam mit

Experten aus Österreich und in enger Zusammenarbeit mit dem Swiss- sowie dem German Testing Board maßgeblich an der Aktualisierung und Weiterentwicklung Internationaler (Testing) Standards mitwirkt.

Richard Seidl

Richard Seidl ist Agile Quality Coach und Software Test Experte. Er hat in seiner vielseitigen beruflichen Laufbahn schon viel Software gesehen und getestet: gute und schlechte. Große und kleine. Alte und neue. Seine Erfahrungen verbindet er nun zu der Erkenntnis, dass die Entwicklungs- und Testprozesse nur dann erfolgreich sein können, wenn die verschiedensten Kräfte, sowie Stärken und Schwächen, ausgewogen sind. So wie ein Ökosystem nur harmonisch mit allen Aspekten in seiner ganzen Qualität bestehen kann, so müssen auch Prozesse im Testing-Umfeld als ein Netzwerk der verschiedenen Akteure betrachtet werden. Qualität wird dann zur Haltung, die man wirklich leben kann, anstatt sie nur abzuarbeiten. Als Autor und Mitautor hat er verschiedene Fachbücher und Artikel veröffentlicht, unter anderem „Der Systemtest – Von den Anforderungen zum Qualitätsnachweis“ (2008), „Der Integrationstest – Von Entwurf und Architektur zur Komponenten- und Systemintegration“ (2012) und „Basiswissen Testautomatisierung“ (2012).

Siegfried Tanczos

Siegfried Tanczos ist seit 2004 bei ANECON Software Design und Beratung GmbH beschäftigt und seit vielen Jahren auch Teamleiter in der Solution Test Consulting & Organisation. Neben seiner Leitungsfunktion ist Siegfried Tanczos von Anbeginn seiner Tätigkeiten bei ANECON in etlichen Software-Testprojekten im Einsatz gewesen. Seine Erfahrungen im

Bereich Software-Test baute er bereits während seiner beruflichen Laufbahn im Bankenumfeld auf und er ist seit 1998 als Software-Tester tätig. Durch seine Arbeit in diversen Kundenprojekten bei ANECON konnte Siegfried Tanczos weitreichende Erfahrungen im Umgang mit klassischen und agilen Vorgehensmodellen sammeln.

Danksagungen

Wir danken den Unternehmen ANECON Software Design und Beratung GmbH, GETEMED Medizin- und Informationstechnik AG und Otto (GmbH & Co KG) für die Unterstützung bei diesem Buch.

Ebenso danken wir unseren Kolleginnen und Kollegen für deren eifrige Unterstützung und unseren Reviewern, die uns mit kritischen Hinweisen geerdet haben und so einen wertvollen Beitrag zu diesem Buch geleistet haben:

Sonja Baumgartner (Grafik), Stefan Gwihs, Diana Kruse (Praxisbeispiele *Otto.de*), Anett Prochnow, Petra Scherzer, Michael Schlimbach, Silvia Seidl und Harry Sneed. Unser Dank geht auch an Petra Kienle, die im Copy-Editing noch viele Fehler ausbügelte, die uns entgangen waren.

5 Agiles Testmanagement, -methoden und -techniken

Das Management umfasst (lt. Wikipedia) die Aufgaben Planung, Organisation, Führung und Kontrolle einer Aktivität. Ein Projektmanager ist demnach jemand, der ein Projekt plant, organisiert, überwacht und steuert. Dazu braucht er Wissen, Erfahrung, Information und Kompetenz. Um das Management eines agilen Tests zu verstehen, müssen wir uns erst bewusst ansehen, wie solche Projekte funktionieren. Dazu ist schon vieles geschrieben worden. In den meisten agilen Vorgehensmodellen wie z. B. Scrum, Kanban, Extreme Programming und Lean Management, (Details dazu finden Sie in Kapitel 2, „Agile Vorgehensmodelle und deren Sicht auf Qualitätssicherung“) wird der Test, wie wir ihn von traditionellen Modellen kennen, nicht explizit angesprochen – es wird implizit vorausgesetzt, dass dies durch die Teams selbstverständlich abgedeckt wird.

Der Test ist also untrennbar mit der Entwicklung verwoben. Es mag zwar Tester geben, also Spezialisten mit dem Schwerpunkt Qualitätssicherung und Akzeptanztest, aber eine separate Testschiene kommt in den meisten Vorgehensmodellen nicht vor (Kanban lässt dies bewusst offen). Teammitglieder mit Testfokus decken somit mehrere Rollen ab: zum einen die des Testers, zum anderen die des Testmanagers und schließlich auch die des Qualitätsmanagers. Welche Aufgaben sie dabei wahrnehmen, haben wir bereits in Kapitel 4, „Die Rolle des Testers in agilen Teams“, beschrieben. Da dies essenziell ist, sei hier nochmals erwähnt, dass sich Tester in der agilen Welt praktisch selbst managen. Falls es mehrere Tester in einem Team gibt, teilen diese sich die koordinierenden Aufgaben meist untereinander auf (Beck, 2000).

Unabhängig von der Vorgehensweise, also ob traditionell oder agil, gibt es eine Vielzahl an Tätigkeiten, die Tester in Projekten wahrnehmen.

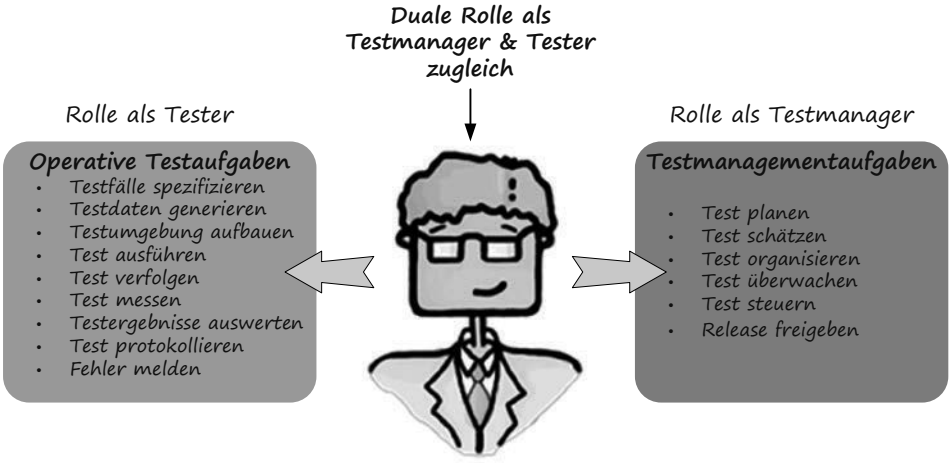


Bild 5.1 Tester als Manager

5.1 Testmanagement

Der Begriff Testmanagement dient dazu als Sammelbegriff, den wir, um ihn hier besser beschreiben zu können, grob in folgende Einzelthemen zerlegen:

- Testplanung
- Testschätzung
- Testorganisation
- Testerstellung, Durchführung, Freigabe
- Testüberwachung
- Teststeuerung

5.1.1 Testplanung im traditionellen Umfeld

Bevor wir die Testplanung im agilen Umfeld betrachten, ist es hilfreich, sich dazu grob das bisherige Vorgehen im traditionellen Projektumfeld nochmals vor Augen zu führen. In konventionellen Software-Entwicklungsprojekten wird der Test als eigenständige Aktivität geplant. Das Testkonzept bzw. die Prüfspezifikation gilt als Teil der Pflichtlieferung. Aber wie lief die Testplanung in der Praxis in diesen Projekten ab?

Laut einer Umfrage zum Software-Test in der Praxis unter öffentlichen Organisationen und Unternehmen im deutschsprachigen Raum (Haberl, Spillner, Vosseberg, & Winter, 2011) wird früher oder später ein Testmanager nominiert, der umgehend mit der Testplanung beginnt.

„Moment!“, werden Sie zu Recht sagen, „da fehlt ja noch der oft sehr zeitaufwendige, mühsame Teil der Unterlagensichtung!“ Richtig.

Also macht sich der nominierte Testmanager am Projektordner auf die Suche nach:

- Architekturschaubilder
- Anforderungsdokumente
- Spezifikationen zu Funktionalität, Services, Masken
- GUI-Prototypen
- Styleguides
- Lasten- und Pflichtenhefte
- Use-Case-Beschreibungen (verbal, grafisch)
- Big Picture
- Projektplanung
- Projekt- und Qualitätsmanagementhandbuch
- Testdatenstrategie
- Coding-Guidelines
- Risk-Management-Plan
- u. v. m.

Kurzum: alles, was sich zum vorliegenden Projekt an Dokumenten auftreiben lässt.

Mit diesen Unterlagen als Grundlage bekommt das Testkonzept, meist basierend auf dem von ISTQB empfohlenen Template der IEEE 29119 (ehem. IEEE 829), nach und nach Inhalte.

Parallel dazu ist der Testmanager auch oft gefordert, bereits konkret den Testaufwand zu schätzen, der in die Gesamtprojektplanung einfließt. Viele haben dabei immer wieder Probleme, da die Anforderungen oft unvollständig oder so vage beschrieben sind, dass man daraus keine seriöse Schätzung ableiten kann. Ein weiterer Aspekt, der die Schätzung sehr fehleranfällig macht, ist die Tatsache, dass zu diesem Zeitpunkt noch nicht durchgängig absehbar ist, ob „trivial scheinende Anforderungen“ plötzlich zu „Aufwandsfressern“ mutieren.

Oft behelfen sich erfahrene Testmanager damit, Erfahrungswerte von ähnlichen Projekten aus ihrer Vergangenheit als Schätzgrundlage zu nehmen. Ein anderer, auch oft verwendeter Ansatz ist die 20 bis 30 % Daumenschätzung. Das heißt, vom geschätzten Gesamtprojektbudget kalkuliert man rund ein Viertel für die Testaktivitäten.

Die Erfahrung zeigt, dass solche Ansätze zwar Zeit und Budget für Test vorsehen, die Realität am Ende aber dann doch ganz anders aussieht und der Test nur als billiger Puffer für die Verzögerungen der Vorphase herhalten muss. Diese Praktik hält sich bis heute, obwohl man schon x-fach erfahren musste, dass man zu diesem Zeitpunkt noch nicht abschätzen kann, was letztlich alles gebraucht wird.

Hier passt das etwas abgewandelte Zitat von Sokrates:

„Ich weiß nicht, was ich nicht weiß.“

Diese Problematik konnte man bislang im traditionellen Vorgehensmodell nicht wirklich in den Griff bekommen. Warum soll das durch das agile Vorgehen anders werden?

5.1.2 Testplanung im agilen Umfeld

Wie wir bereits in Kapitel 1 und 2 ausführlich beschrieben haben, ist dieses Dilemma der Initialfunke der agilen Bewegung. Diese radikale Umstellung der Vorgehensweise bei der Abwicklung von Projekten hat auch gravierende Auswirkungen auf alle Phasen des Tests, von der Planung, dem Design, Entwurf über die Durchführung bis hin zur Steuerung des Tests an sich.

Auch wenn die Rolle der Tester und speziell des Testmanagers umstritten ist, so liegt es auf der Hand, dass in einem professionellen Projektumfeld – egal ob traditionell oder agil – viele Aufgaben zur Sicherstellung der Qualität anfallen und auch durchgeführt werden müssen. Wie heißt es so schön: „Vertrauen ist gut, Kontrolle ist besser."

Für den agilen Ansatz würden wir diese Aussage folgendermaßen adaptieren: *Vertrauen ist gut, Kontrolle mit laufendem und konstruktivem Feedback zur Verbesserung ist das Beste.*

Diese Chance bietet sich in agilen Teams und Projekten. Die Aktivitäten zur Sicherstellung der Qualität ziehen sich über die gesamte Projektlaufzeit und sind vom gesamten Team wahrzunehmen. In vielen von uns aktiv begleiteten Projekten zeigt sich, dass nur dann der Aspekt der Qualität akzeptabel mitberücksichtigt wird, wenn sich jemand aktiv darum kümmert.

Wie die Rolle bezeichnet wird, die all diese Aufgaben wahrnimmt, ist letztlich egal. Nennen wir sie der Einfachheit halber Testmanager und/oder Tester bzw. die Aktivität „Testen".

Hier gilt dasselbe, das uns schon aus der traditionellen Welt bekannt ist:

„Je früher man als Tester bzw. Testmanager in ein Projekt eingebunden wird, desto besser."

Im agilen Umfeld bedeutet dies, dass lange bevor ein Projekt gestartet wird, bereits mit der Sammlung von Produktideen begonnen werden kann und diese dann in das Product Backlog eingetragen werden.

Um nun rasch zu geeigneten User Stories zu kommen, kann man bewusst das Pferd von hinten aufzäumen und erst einmal konkrete Beispiele zu einer Anforderung sammeln und dann in ein bis zwei erklärenden Sätzen zusammenfassen (Adzic, 2011). Damit ist dann die User Story entstanden – mehr braucht es oft nicht. Wir ergänzen das mit „… und wirf bereits dabei ein Auge drauf, auch wenn sie noch so grob sind, ob diese in sich testbar sind".

Das heißt: Bereits hier ist Sorgfalt wichtig. Es sollen nur Stories aufgenommen werden, die als Deliverable wirklich testbar sind und vor allem dem Endkunden auch wirklich Nutzen bringen. Also primär Funktionalität und Abläufe.

Dann ist des Öfteren zu hören: „Wenn wir über Planung sprechen, was ist mit den technischen Anforderungen wie Antwortzeit, Stabilität o. Ä.? Wo definiere ich diese Qualitätsmerkmale?“ Diese sind ja auch wichtiger Bestandteil eines Produkts und mitunter aufwendiger umzusetzen – denken Sie nur an eine eventuell benötigte Datenmigration oder den erforderlichen Umstieg auf ein neues Framework.

Folgt man den Aussagen aus der Agile Community, so sollten diese Aspekte – formuliert in Form von Akzeptanzkriterien – den User Stories zugeordnet werden, da nur den Stories auch ein echter Nutzen zuzurechnen ist. Kurz: „Technische Stories“ gehören nicht in ein Backlog. Mehr davon später.

Eine Vorgabe, wie die Qualität im Projekt sichergestellt werden soll, ist abhängig von der Teamstruktur (Hinweis: Beispiele nach Scrum-Notation):

a) **Ein Team an einem Projekt**
   Hier gilt es primär sicherzustellen, in welchem Umfang das Team die Qualität gewährleisten soll. Es hängt von den mit dem Endkunden/Product Owner vereinbarten Deliverables hinsichtlich des Qualitätsnachweises ab, ob extra Reports oder Dokumente anzufertigen sind oder ob z. B. der Korrektheitsnachweis im Rahmen der Sprint Reviews und der teamintern definierten seriösen Definition of Done (DoD) ausreichend ist.

b) **Mehrere Teams an einem Projekt (Scrum of Scrum)**
   Grundsätzlich gilt dasselbe wie bei Punkt a), jedoch legen Unternehmen, die solche Scrum of Scrum-Teams bereits länger im Einsatz haben, darauf Wert, dass die Art und Weise, wie der Qualitätsnachweis zu erfolgen hat, teamübergreifend einheitlich erfolgen soll. Als einzige Ausnahme gilt hier: Der Kundenvertreter/Product Owner bestellt andere Qualitätsnachweise. Ist das Projekt einmal abgeschlossen, legen die Teams die Ergebnisse, Dokumentationen, Qualitätsnachweise etc. zusammen und archivieren diese. Steht in Folge eine Weiterentwicklung an, ist die Vorgeschichte, also wie das Produkt initial erstellt wurde, unerheblich – da muss es „wie aus einem Guss“ sein.

c) **Ein oder mehrere Teams in einer Organisation**
   In vielen, gerade größeren Unternehmen gibt es einheitliche Qualitätsvorgaben, sei es aufgrund von ISO-Vorgaben (900x) oder einem Regelwerk wie z. B. nach CMMi, das bestimmte Ergebnisse verlangt. Daher ist bei der Einführung, beim Aufbau und Etablieren von agilem Vorgehen unbedingt darauf zu achten, diese Rahmenbedingungen einzuhalten und zu erfüllen. Hier ist der Qualitätssicherungsbeauftragte oder Testmanager gefordert, den Rahmen für die Teams vorzugeben.

d) **Teams, die an besondere Rahmenbedingungen gebunden sind**
   Unternehmen/Teams, die Software produzieren, die gesetzlichen Regulatoren unterliegen wie z. B. im Medizin-, Pharma- oder safety-kritischen Umfeld, sind es gewohnt, ausführlichere Test- und Qualitätsnachweise erstellen zu müssen. Diese werden nicht obsolet, nur weil man nun agil vorgeht. Somit wird die Erstellung dieser Nachweise vom Team von Beginn an fix in ihren Ablauf integriert. Wie Teams dies handhaben, ist individuell verschieden. Manche sehen diese Nachweise als fixes Deliverable jedes Sprints, andere integrieren die Erstellung von Qualitätsnachweisen fix in ihre Definition of Done.

In den uns bekannten Unternehmen wird darauf Wert gelegt, dass unabhängig von den gerade beschriebenen Varianten einheitlich gearbeitet wird.

Das bringt den Unternehmen und Teams folgende Vorteile:

- **Teamübergreifender Ressourcenausgleich**
  Identifiziert das Team – aus welchen Gründen auch immer – einen Ressourcenengpass und adressiert diesen als Impediment im Daily Stand Up Meeting, ist der Scrum Master am Zug. Dieser kümmert sich darum, dass sein Team Verstärkung bekommt. Arbeiten nun alle Teams im Projekt bzw. Unternehmen nach ähnlichen (Test-) Vorgaben, können sich Mitglieder anderer Teams mit überschaubarem Einarbeitungsaufwand in das neue Team integrieren.
- **Synergien in Technologien und Methoden nutzen**
  Ähnlich verhält sich das auch in Bezug auf Technologien und Methoden.

Alles in allem: Je einheitlicher Teams arbeiten, desto flexibler kann teamübergreifend agiert werden. So können z. B. bei Bedarf Ressourcen in anderen Teams auch ohne größere Reibungsverluste aushelfen, wenn gerade „Not am Mann" ist. Wobei hier ganz klar festzuhalten ist: So eine Vereinheitlichung darf natürlich nicht so weit gehen, dass Teams nicht mehr individuell arbeiten können.

5.1.3 Testkonzept

Wie könnte nun so ein generelles Testkonzept in einem nach agilen Grundsätzen arbeitenden Team aussehen?

Bewährt hat sich hier, das generelle Testkonzept, ähnlich der Unternehmenstestrichtlinie/Test Policy, also auf hohem Abstraktionsniveau zu gestalten. Darin definiert der Testmanager die Prinzipien, Vorgehensweisen und wichtigsten Ziele einer Organisation in Bezug auf das Testen, die für jedes Projekt einzuhalten sind.

Dieses könnte folgendermaßen aussehen:

Testrichtlinie der Fa. QualityIsOurSuccess GmbH[1]

1. Definition des Testens im Unternehmen

Unsere Produkte stehen für Qualität. Wir stimmen jede Kundenanforderung mit unseren Kunden ab, identifizieren gemeinsam jene, die unseren Kunden am wichtigsten sind bzw. den größten Nutzen bringen, und liefern diese zügig und mit nachweislicher Qualität.

Diese Qualität wird nachweislich im gesamten Entwicklungsprozess gelebt.

Kurz: *„Ensuring the software fulfills its requirements"*[2]

[1] Fiktives, frei erfundenes Beispiel, in Anlehnung an konkrete Testrichtlinien

[2] Quelle für diese und alle weiteren Kurzdarstellungen ist *TestingExcellence.com*.

2. Definition des Testprozesses

Von der Erhebung der Anforderungen über den Ansatz des Test-Driven Development begleitende Sicherstellung der Qualität durch Tester in den Teams, die die Qualität jeder einzelnen Anforderung aus Enduser-Sicht sicherstellen, Automatisierung der Akzeptanzkriterien bis hin zu Regressionstests der bestehenden Funktionalität.

Im Detail ist das folgendermaßen sicherzustellen:

- **Backlog Item Quality**
  Jeder Eintrag, der in das Product Backlog aufgenommen wird, ist dahingehend zu prüfen, dass dieser das richtige Story-Format hat und der vereinbarten Beschreibungstiefe entspricht.
- **Testbare Akzeptanzkriterien**
  Sicherstellen, dass jedes Backlog Item bzw. jede User Story testbare Akzeptanzkriterien aufweist – hier sind statische Testtechniken wie Reviews anzuwenden und so lange mit Product Owner, Entwicklern und Business-Analysten abzuklären, bis alle Kriterien klar sind.
- **Testaufwand berücksichtigen**
  Einbringen der Testsicht bei allen Planungsmeetings, d. h., bei den Schätzungen ist sichergestellt, dass auch der Testaspekt berücksichtigt wird.
- **Testentwurf und Durchführung**
  Während der Entwicklungszyklen (Sprints) sind für jedes Akzeptanzkriterium je nach Kritikalität Testfälle zu entwerfen, zu dokumentieren und durchzuführen.
- **Abnahmetest**
  Sicherstellen, dass zu jedem Akzeptanzkriterium zumindest ein Testfall erstellt wird.
- **Traceability**
  Jeder Testfall ist eindeutig einer Story zugeordnet.
- **Sprint Review**
  Am Ende jedes Sprints ist die Funktionstauglichkeit jeder Story, die als „Fertig“ gemeldet wurde, dem Product Owner bzw. Endkunden im Rahmen eines Sprint Reviews zu beweisen. Der Beweis gilt als erfolgreich, wenn die Akzeptanzkriterien fehlerfrei vorgestellt werden konnten.

Kurz: *„All test plans are written in accordance with company policy.“*

3. Sicherstellung der Qualität (Testevaluation)

- Vorgaben für Teams bzgl. Entwicklung:
- Jede Funktion ist mittels TDD (Test-Driven Development-/Test First-Ansatz) zu entwickeln. Die so entstandenen Unit- und Integrationstests sind im Rahmen der Daily/Nightly Builds laufend durchzuführen. Sollten dabei Fehler auftreten („Broken build“), sind diese umgehend zu beheben.

- Fertigkriterium:
- Generell gilt für die Umsetzung jeder Anforderung (Story):
- Fertig ist eine Story erst dann, wenn alle Kriterien der Definition of Done erfüllt sind
- (z. B. vom Team sämtliche Tests (Unit Tests und funktionale Tests) erfolgreich durchgeführt wurden („Green Bar“).
- Qualitätsnachweis:
- Der Nachweis ist durch ein Testprotokoll zu erbringen.
- Teamzusammensetzung:
- Das Team besteht aus Mitgliedern mit cross-funktionalen Skills. Um die Enduser-Sicht in den Teams bereits zu berücksichtigen, ist in jedem Team mindestens ein Tester fix einzubinden.

Kurz: *„Effect on business of finding a fault after its release.“*

4. Zu erreichende Qualitätskriterien

Durch „Fertig“-Meldung von Stories ist sichergestellt, dass keine Fehler der Klasse „Blocker“ und „Schwere Fehler“ vorliegen. Somit ist sichergestellt, dass diese Funktion im Einsatz funktionstauglich ist.

Kurz: *„No outstanding high severity faults prior to products release.“*

5. Verbesserungsprozess

Unser Ziel:

Lernen aus Erfahrungen, laufende Verbesserung in allen Bereichen des Entwicklungszyklus.

Jeder Entwicklungszyklus (Sprint) ist mit einer Retrospektive abzuschließen, worin Rückblick und Potenzial für Verbesserungen entwickelt werden.

Kurz: *„Project review meetings to be held after project completion.“*

■

Begriffserklärung

Broken Build: Unit Tests, die nach Erstellung eines Build automatisch durchgeführt werden, liefern Fehler. Die Ursache ist, dass die zuletzt eingecheckte Komponente mit den anderen Komponenten noch nicht kompatibel ist.

Green Bar: Im Continuous Integration System (z. B. via Tools Jenkins oder Hudson) werden die Ergebnisse der automatisch laufenden Unit- und Integrationstests in einem Report dargestellt. Ziel ist es, dass der Ergebnisbalken für den Test jedes Build fehlerfrei, also grün ist.

Die Testrichtlinie bildet den generellen Rahmen über alle im Unternehmen abgewickelten Projekte, ist für alle verbindlich und entsteht im Normalfall bereits lange, bevor ein Projekt initiiert bzw. ein Projektteam aufgestellt ist. Unternehmen, die bereits länger nach diesem Vorgehensmodell arbeiten, sehen diese Richtlinie als Essenz der bisherigen Erfahrungen, die auch laufend aktualisiert wird. D. h., keiner der angeführten Punkte ist in Stein gemeißelt, sondern ganz im Gegenteil, hier wird ganz stark das Prinzip „inspect and adapt“ angewandt.

Nun geht's endlich mit einem Projekt los. Das Team wird zusammengestellt, nimmt Formen an und geht in die Projektstartphase.

5.1.4 Testaktivitäten in Iteration Zero – Initialisierungs-Sprint

Genauso wie in der Chirurgie alle Geräte hergerichtet werden, damit sie bei der Operation griffbereit sind, bereitet man in unserem Fall das gesamte Umfeld vor, damit sich das Team in den Sprints auf das Ziel konzentrieren kann: nämlich Deliverables zu produzieren.

„Jeder Weg beginnt mit dem ersten Schritt."

Dieser erste Schritt nennt sich Iteration Zero und inkludiert gleich auch einiges an Testaktivitäten.

Diese gilt es, in allen Phasen gleich mit zu berücksichtigen und auch explizit einzuplanen. Dazu haben wir die folgende „Checkliste Iteration Zero" mit den Testaktivitäten ergänzt (die Liste erhebt keinen Anspruch auf Vollständigkeit):

Allgemeine Teamaufgabe	Spezielle Testaufgaben dabei
▪ Aufbau/Ausbau des Product Backlog – Priorisieren der Backlog Items	▪ Review der Items auf Testbarkeit ▪ Betrachtung aus Enduser-Sicht ▪ Sind Akzeptanzkriterien vorhanden, die klar, eindeutig, widerspruchsfrei formuliert sind? Wichtig dabei: Sind diese auch testbar? ▪ So die Einträge im Product Backlog bereits mit groben Aufwänden oder Story Points (Aufwandskennzahl) versehen werden, ist sicherzustellen, dass hier auch Testaspekte berücksichtigt sind.
▪ Aufsetzen der Umgebung/en ▪ Entwicklung ▪ Continuous Integration (CI) ▪ Test ▪ Pre-Production	▪ Ist der CI-Prozess aufgesetzt? ▪ Ist sichergestellt, dass die Unit Tests (für TDD) eingebunden und die Ergebnisse transparent sind (Reports, Dashboard u. Ä.)? ▪ Ist der Prozess CI auf Testumgebung klar definiert? ▪ Ist der Prozess Test - Pre-Production definiert?
▪ Auswahl/Definition des Entwicklungsvorgehens	▪ Dazu passendes Testkonzept erstellen.
▪ Rekrutieren/Formen eines Teams bzw. Finalisieren des Teamaufbaus (inkl. aller Rollen)	▪ Sind Tester mit an Bord? ▪ Haben diese ausreichend Know-how? Wenn nicht, dann bereits grobes Ausbildungskonzept skizzieren.
▪ Definieren der Arbeitsumgebung (Räumlichkeiten, Equipment …)	Dasselbe gilt auch für Testumgebungen.
▪ ToolauswahlErfolgt idealerweise via Proof of Concept – Checkliste geeigneter Tools (Software) für Entwicklung inkl. GUI, Unit Tests, Codeanalyse, CI, Collaboration, Testautomatisierung finden	Vorgehen/Konzept bei Testautomatisierung auf System- und Akzeptanztest-Ebene erstellen Definieren der Testverwaltung
▪ AutomatisierungSind Unit-Test-Frameworks vorhanden und einsetzbar?	▪ Ist Testautomatisierungs-Framework vorhanden und einsetzbar

Allgemeine Teamaufgabe	Spezielle Testaufgaben dabei
▪ Wenn das Team schon verfügbar ist: Erarbeiten, wie Zusammenarbeit sichergestellt ist: Wo steht das Taskboard, wie schaut dieses aus? etc.	Klären, wie Task-Handling erfolgen wird: ▪ Gibt es eigene Entwicklungs- und Tester-Tasks oder werden Tasks jeweils ähnlich wie bei einem Staffellauf von der Entwicklung direkt an den Test übergeben? Ist Transparenz gegeben, wann die Entwicklung mit den Tasks fertig ist und wann Tester diese übernehmen und zu Ende führen können? Wie organisieren Tester eigene Tester-Tasks? Nutzt man dasselbe Taskboard wie das Team oder ein eigenes? Festlegen, wie Defects kommuniziert werden Taskboard/Tool
▪ Gemeinsames Erarbeiten der Definition of Done	Fokus auf Qualität, Testbarkeit und Messbarkeit legen ▪ Sind vorgegebene Rahmenbedingungen in DoD abgedeckt (wie z. B. gesetzliche Regulatorien)?

5.1.5 Externe Unterstützung der Testplanung

In agilen Teams sind die Planungsaktivitäten grundsätzlich direkt ins Team integriert und in den einzelnen Sprints durchzuführen.

In manchen Fällen ist es erforderlich, besondere Spezifikationen auf- bzw. umzusetzen, die innerhalb des Teams schwierig bis unmöglich zum Umsetzen sind. Dazu gehören z. B.:

- Partnersysteme (Drittsysteme, die für den übergreifenden Ablauftest benötigt werden; Zeitfenster für Nutzung solcher Systeme etc.)
- Testdaten (so z. B. auch von Drittsystemen zuzuliefernde spezifische Daten)
- Definition von Detailkomponenten, die besonderen Rahmenbedingungen unterliegen (Safety- und Security-Kriterien, Compliance-Richtlinien, Oberflächen, wo z. B. Barrierefreiheit gefordert ist)
- Besondere Architekturvorgaben (bzgl. Performance, Ausfallssicherheit, …)

Also überall dort, wo eine langfristigere Planung bzw. Vorbereitung erforderlich ist, hat es sich bewährt, einen Verantwortlichen auch außerhalb des Teams zu finden, der diese Aktivitäten für das Team verfolgt und erledigt. Hier ist es von Unternehmen zu Unternehmen unterschiedlich, ob es dafür jemanden in der Organisation gibt oder ob das jemand aus dem Team zusätzlich übernehmen muss.

5.1.6 Testschätzung

Im Unterschied zu traditionellen Projekten, in welchen es meist nur in Unternehmen mit einem hohen Reifegrad umgesetzt war, dass die Qualitätssicherung und die Einbindung des Tests bereits in der Anforderungsphase beginnt, ist dies in agilen Projekten von Haus aus, zumindest dem Prozess nach, sichergestellt.

In agilen Projekten steht als eine der ersten Aufgaben in der Initialisierungsphase die Befüllung des Product Backlog mit Stories durch den Product Owner an. Diese Stories beinhalten neben der Story-Beschreibung bereits rudimentäre Ansätze der Akzeptanzkriterien, den Business Value[3] sowie eine grobe Ersteinschätzung der Komplexität der Story. Diese Komplexität wird in Story Points[4] angegeben. Oft ist es hier schon der Fall, dass vom Tester neben einem Review der User Stories (auf Testbarkeit hin) auch schon eine grobe Schätzung der Komplexität gefordert wird, d. h. wie viele Story Points im Vergleich zu den anderen Stories für die Umsetzung benötigt werden.

Dabei hat sich – unabhängig, ob man das aus Sicht der Entwicklung oder des Tests betrachtet – folgende Herangehensweise bewährt:

1. Man wählt aus den vorhandenen Stories eine möglichst einfache – aus Testsicht gut abschätzbare – Story als Referenz-Story aus.

   Diese könnte z. B. aus Testsicht folgende Eckdaten haben:

   - Fünf bis zehn Testdatensätze (Typ einfach) müssen erstellt werden.
   - Ergebniswerte sind nur gegen Referenzwerte zu verifizieren.
   - Es braucht keine Daten aus Fremdsystemen.

   Setzt man hier z. B. das Planungs-Poker ein, bekäme diese Story nun die Komplexität „1 Story Point".

   Ist Automatisierung auf Akzeptanz-/Systemtestebene vorgegeben, nimmt man den nächst höheren Wert.
2. Bei den weiteren Stories vergleicht man nun gegen diese Referenz-Story und erhält so eine Ersteinschätzung – aus Testsicht.
3. Die so erhaltenen Werte sind als ein erster Richtwert zu verstehen und auch ein Maß dafür, ob die Story angemessen „geschnitten" ist, also ob sie realistischerweise in einem Sprint umgesetzt werden kann.
4. Die eigentliche Schätzung kann jedoch erst im Rahmen der Sprint Planning Meetings erfolgen. Denn erst, wenn alle Teammitglieder ihre Sichtweisen zu den einzelnen Stories einbringen, kann eine Story wirklich vernünftig erfasst und bewertet werden.

   Hier nehmen die Tester durchwegs auch eine aktive Rolle ein. Denn, Sie erinnern sich: Ziel ist es, jede Story im Rahmen eines Sprints „potential shippable" (also fertig und auslieferbar) zur Verfügung zu haben, und da gehört der Test nun einmal untrennbar dazu.

   Manchmal scheint eine Story nur eines minimalen Entwicklungsaufwands zu bedürfen, der jedoch umfangreiche (Regressions-)Testaktivitäten nach sich zieht. Hier muss sich der Tester ggf. gegen das gesamte Team behaupten, falls das Bewusstsein im Team noch nicht sensibilisiert ist.

---

[3] Unter Business Value versteht man einen Richtwert, mit dem der Product Owner eine Gewichtung der Stories vornehmen kann. Je höher dieser Wert ist, desto mehr Nutzen bringt die Funktionalität dem Endanwender.

[4] Story Points ist eine beliebte Schätzmethode, mit der die Größe einer Story angegeben wird. Die häufig verwendete Bezeichnung Komplexitätsmaß ist unserer Erfahrung nach zu kurz gedacht: Für uns ist das Maß abhängig von Umfang, Risiko, Klarheit der Story, Erfahrung des Teams und auch Komplexität. Als Messgröße hat sich die von Mike Cohn angepasste Fibonacci-Reihe etabliert, die folgende Punktewerte umfasst: 1, 2, 3, 5, 8, 13, 20, 40, 100 Je höher der Wert, desto komplexer, umfangreicher, risikoreicher,… ist die Story.

5. Auch beim laufenden Nachschätzen der noch im Product Backlog befindlichen Stories („Backlog Grooming“) sind die Tester aufgefordert, ihre Expertise einzubringen.

5.1.7 Testorganisation

In Kapitel 4 lernten wir schon verschiedene Teamkonstellationen kennen – auch die, die sich in einigen unserer Projekte gut bewährt haben.

Jedes agile Team besteht u. a. aus zumindest ein bis zwei ausgebildeten Testern. Weiterhin stehen dem Team – so erforderlich – ergänzend noch einige Support-Teams oder auch einzelne Fachexperten zur Verfügung, die für Spezialthemen herangezogen werden können.

Ein Security-, Recovery- oder Last- und Performance-Experte bzw. Team wird – als eine Möglichkeit – immer dann vom Team „adoptiert“, wenn Stories diese (meist nichtfunktionalen) Qualitätsmerkmale einfordern. Konkret heißt das, dass das Team temporär erweitert wird. Die temporären Teammitarbeiter haben in diesem Fall dieselben Rechte und Pflichten wie alle anderen Teammitglieder – d. h., sie committen sich ebenfalls zum Sprint-Ergebnis.

Eine Alternative bietet sich dazu an und zwar, diese Stories oder Testaufgaben außerhalb des Teams durch andere Organisationen bzw. Teams durchführen zu lassen.

Ähnlich verhält es sich mit anderen Supporting-Rollen wie DB-Modellierer, Systemarchitekten etc. Werden solche Spezial-Skills im agilen Team benötigt, werden diese Know-how-Träger temporär ins Team engagiert und erledigen dann diese Aufgabenstellungen entweder als externer Zulieferer oder sogar als temporäres Teammitglied während des aktuellen Sprints.

Vereinfacht kann die Teamzusammenstellung so dargestellt werden:

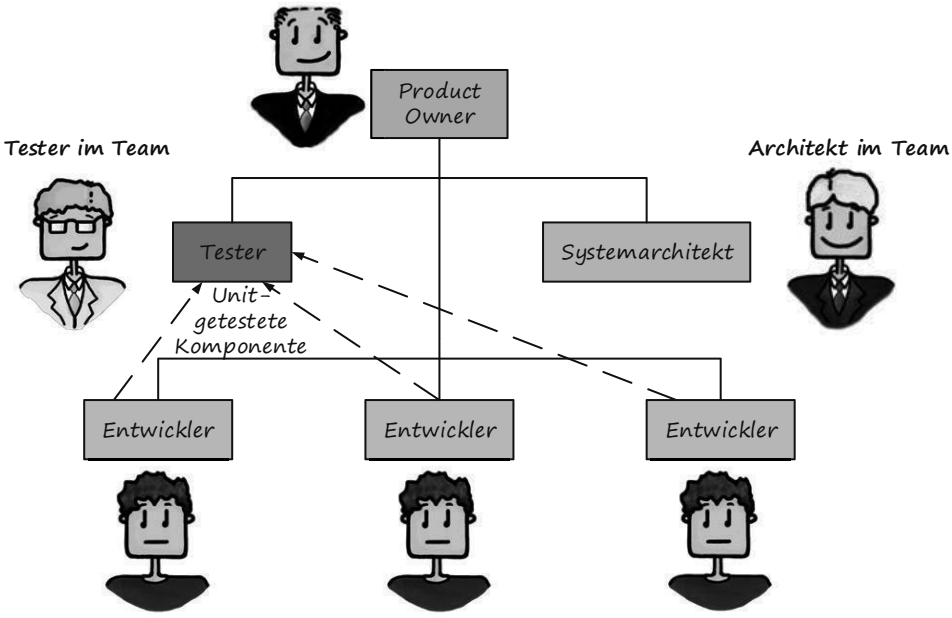


Bild 5.2 Teamzusammenstellung und Position der Tester

5.1.8 Testerstellung, Durchführung und Release

Sind agile Teams co-located aufgesetzt, können sie einen wesentlichen Vorteil generieren, und zwar jenen der direkten Kommunikation. Dieser Mehrwert wird oft massiv unterschätzt, zeigt sich jedoch dann ganz deutlich, wenn man bei Teams den Mehraufwand transparent macht, den sie durch auf mehrere Standorte verteilte Teams Tag für Tag in Kauf nehmen müssen. Das beginnt bereits beim Review und bei der Schätzung von Stories bei der initialen Befüllung des Backlogs, setzt sich über die Release- und Sprint-Planung fort, geht über die Vorbereitung zum Sprint Review bis hin zur Abnahme der Story und letztlich dem Release. Je mehr direkte Kommunikation hier möglich ist, desto effizienter wird ein Team arbeiten. Auch wenn man sich heutzutage ein verteiltes Team quasi auf den Bildschirm bringen kann, geht nichts über einen Erfahrungsaustausch von Angesicht zu Angesicht.

Wenn wir nun den Test betrachten, sehen wir, dass in all den oben genannten Phasen das gesamte Team gefordert ist, am gemeinsamen Ziel zu arbeiten. Je mehr es das Team verstanden hat, die „Hand in Hand"-Technik zu praktizieren, desto effizienter und rascher wird es ans Ziel, der Erfüllung der Akzeptanzkriterien jeder Story, kommen. Tester führen ihr Team zum Erfolg, indem sie gezielt die Schwächen und Probleme identifizieren und das Team dahingehend steuern, dass diese Schwachstellen beseitigt werden.

Was den eigentlichen Test betrifft, so kommt einem ein im Test altbewährtes Prinzip zugute: dass nämlich die Anforderungen bereits dadurch auf Umsetzbarkeit und Klarheit geprüft bzw. reviewt werden, dass sich Tester dazu Testfälle überlegen und entwickeln.

Was den Test selbst betrifft, hat sich nicht viel geändert. Es gilt, in möglichst kurzer Zeit möglichst viele Fehler zu finden. Dazu gehört, neben dem in Folge noch genauer beschriebenen explorativen Testansatz nach wie vor auch, gute Testfälle zu definieren, Testausführungen (mit Vor- und Nachbereitung) zu meistern, Testdaten zu finden und zu verwalten. Einziger Unterschied: Teams erstellen primär dafür ausführlichere Testfälle, wenn es für sie von Nutzen ist bzw. wenn es das Projektumfeld verlangt – z. B. wenn gesetzliche Regularien einzuhalten sind.

Bei **Last- & Performance-Tests** gibt es unterschiedliche Ansätze.

In-Team Abwicklung: Diese finden wir vielfach in frühen Phasen, in welchen es primär darum geht, den gewählten Lösungsansatz der System- oder Datenbankarchitektur zu verifizieren, ob z. B. ein Request durch die gewählte Layer-Struktur auch wirklich die gewünschte Performance aufweist. Diese Tests können mit einigen wenigen parallelen Usern erfolgen und auch z. B. auf der Entwicklungsumgebung durchgeführt werden. Hier steht also die Performance der Einzeltransaktionen im Fokus.

Support-Team Abwicklung: Geht es dann darum, die prognostizierte Produktivleistungsfähigkeit des Systems zu verifizieren, sieht das dafür benötigte Setting wesentlich anders aus. Es geht darum, den Nachweis zu erbringen, dass das erstellte System dem Einsatz in der „realen Welt" gewachsen ist, d. h., ob Anwender das System auch bei Volllast noch innerhalb der akzeptablen Reaktionszeiten bedienen können. Die dabei sehr beliebten Open-Source-Tools sind, sofern man dazu Entwickler einsetzen kann, zwar sehr gut geeignet, um Last zu erzeugen und minimale Messwerte auszuwerten. Aber benötigt man seriöse, tiefgreifende Analysen, um hier die Auswirkungen auf das gesamte Systemumfeld zu berück-

sichtigen, ist der Einsatz professioneller Last- und Performance-Testwerkzeuge unabdingbar. Hersteller solcher kommerziellen Tools haben mittlerweile der Kritik des Markts Rechnung getragen und bieten leistungsfähige Tools schon mit attraktiven Kauf- und Mietmodellen an.

Die Durchführung von Last- und Performance-Tests samt Auswertung und Analyse des Systemverhaltens folgt eigenen Regeln. Diese lassen sich, wenn man es wirklich seriös und professionell abwickeln möchte, schwer in die fixe Taktung von Iterationen/Sprints zwängen, weshalb diese Tests oft auch an Support-Teams im Testcenter oder Betrieb ausgelagert werden. Hier stehen dann auch die benötigten Systeme und Netzwerke zur Simulation einer produktionsnahen Umgebung sowie die Methoden- und Tool-Profis zur Verfügung. Nach Übermittlung der Anforderungen wickeln diese Teams die Tests ab und liefern die Ergebnisse sowie Detailanalysen am Ende dem Team zurück. Die Teams bewerten die Ergebnisse und leiten daraus, sofern erforderlich, Maßnahmen ab, die sie entweder direkt umsetzen oder als „Ticket“ in das Backlog aufnehmen.

Die Testplanung ist das eine, das „moderner“ geworden ist, doch wie sieht es mit den bekannten Testtechniken aus? Hat sich auch da ein Wandel eingestellt?

5.2 Testmethoden im agilen Umfeld

Zehn Jahre nach dem Erscheinen des ersten Artikels von Crispin über agiles Testen in Amerika beschreibt Markus Gärtner im Fachjournal „Objektspektrum“ eine andere Art, agiles Testen zu praktizieren. Er suchte nach einer Antwort auf die berechtigte Frage, wie Tester bei so schnellen Entwicklungszyklen – zwei bis sechs Wochen – mithalten sollen, wenn jeder weiß, dass es länger dauert, Software zu testen als Software zu schreiben (Budd & Majoros, 1978). Bei der Suche nach einer Antwort auf diese Frage ist er auf einige vielversprechende Ansätze gestoßen, darunter

- risiko- und value-basiertes Testen
- exploratives Testen (ET)
- session-basiertes (exploratives) Testen (SBT)
- abnahmegetriebene Entwicklung (ATTD) und
- Testautomatisierung.

Es obliegt den Testern, den geeigneten Ansatz bzw. die geeignete Kombination von Ansätzen für ihre jeweilige Projektsituation zu finden (siehe Bild 5.3).

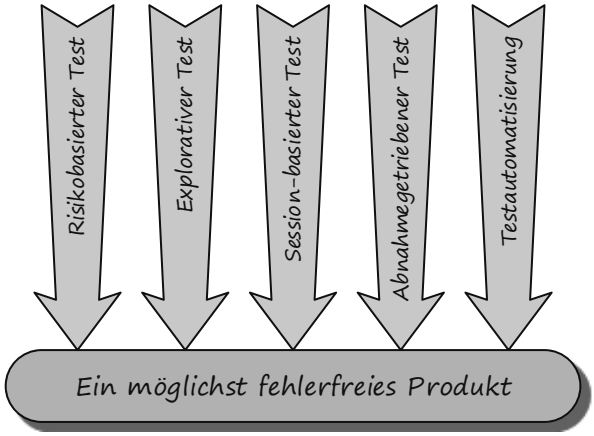


Bild 5.3 Pfade zur Teamunterstützung für das agile Team

5.2.1 Risikobasiertes und valuebasiertes Testen

Gärtner stellte damals schon fest, dass agile Entwicklung mit ihren kurzen Release-Intervallen die bisherigen Testmethoden vor ein schwer lösbares, wenn nicht gar unlösbares Problem stellt. Zeit, um Testfälle für sämtliche Funktionen zu finden, zu dokumentieren und auszuführen, gibt es wohl kaum. Für Tester gilt es daher, einen Kompromiss einzugehen. Der Kompromiss sieht vor, aufgrund der verkürzten Testzeit weniger zu testen – dabei also gerade mal so viel, dass die gravierendsten Fehler doch noch aufgedeckt werden (Menzies & Cukic, 2000).

Der risikobasierte Test bietet einen Weg dazu. Demnach soll der Tester die risikoreichsten Stellen eines Systems heraussuchen und diese gezielt angehen.

Soweit die Idee, aber wie weiß nun das Team, welche Komponenten, Funktionen, Stories oder Abläufe für den Endkunden welches Risiko bedeuten?

Eine Erweiterung des genannten Ansatzes ist, neben dem Risiko auch den Nutzen (Value) der jeweiligen Funktionalität mit einzubeziehen. Das heißt, je geschäftskritischer der Ausfall einer Funktionalität wäre, desto intensiver ist diese im Test zu berücksichtigen.

Hier ist der Product Owner gefragt, die wichtigen Infos bereits bei der Befüllung des Backlogs zu vermerken oder spätestens den Teams beim Planungs-Meeting bekanntzugeben. Darauf aufsetzend führen die Tester zusammen mit dem Rest des Teams eine Risikoanalyse der als kritisch eingestuften Komponenten durch. Jede neue Funktion dieser Komponente wird auf ihre Risiken überprüft. Zum einen wird die potenzielle Auswirkung eines Fehlers in der Funktion eingestuft und zum anderen die Wahrscheinlichkeit eines Fehlers berechnet. Die Auswirkung mal die Wahrscheinlichkeit ergibt die Priorität des Tests (Bach J., 1999).

Ein kreativer „Spaßvogel“ hatte eine noch effizientere Idee:

„Einfach nur für die Teile Tests vorbereiten und durchführen, die Fehler enthalten.

Den Test der anderen, fehlerfreien Teile, kann man getrost einsparen.“

Demjenigen, der das zur „Serienreife“ bringt, wäre der Titel „Rising Star“ am Testerhimmel und auf Fachkonferenzen sicher. Wir arbeiten dran … ;o)

Eine andere Art von Risiko kann sich aus der Umsetzung ergeben: Werden z. B. zentrale Module entwickelt, die das Herzstück für das gesamte System darstellen, sind diese natürlich wesentlich kritischer und somit intensiver zu testen als andere „normale“ Funktionalität.

Durch die enge Zusammenarbeit im Team ist die Identifizierung der risikoreichsten Stellen eines Systems meist einfacher als früher, als Tester erst das fertige Produkt bekamen.

Die Messskala für die oben genannten Faktoren (Auswirkung und Wahrscheinlichkeit) ist dieselbe:

sehr hoch = 4

hoch = 3

mittel = 2

niedrig = 1

Hat also eine Funktion ein hohes Gefährdungspotenzial = 3 und eine hohe Wahrscheinlichkeit, fehlerhaft zu werden = 3, ist die Priorität 9. Hat sie ein mittleres Gefährdungspotenzial = 2 und eine niedrige Wahrscheinlichkeit, fehlerhaft zu werden = 1, ist die Priorität 2. Wichtig ist dabei, dass das ganze Team über die Prioritäten mitentscheidet.

Liegt nun aktuell ein Testdurchlauf an, wirkt in diesem Fall als Testendekriterium: „Testzeit zu Ende“.

Das heißt, es werden nur so viele Testfälle aus der priorisierten Testausführungsliste abgearbeitet, bis der Zeitrahmen ausgeschöpft ist. Punkt. Nicht mehr und nicht weniger. Je länger die Testzeit ist, desto weniger Risiko verbleibt im „nichtgetesteten“ Teil des Systems. So kann es z. B. durchaus auch vorkommen, dass in einem neuen Release nur die Hälfte der Funktionen getestet wird. Durch den gewählten risiko- bzw. value-basierten Testansatz ist dann jedoch zumindest gewährleistet, dass die Funktionen getestet wurden, die am geschäftskritischsten sind bzw. worin im Fehlerfall die größten Gefahren für die Durchführung des Kerngeschäfts liegen.

Sollte es Fehler in den restlichen Funktionen geben, werden diese schlimmstenfalls irgendwann einmal in der Produktion auftauchen. Da sie jedoch – so man die Einschätzung des Risikos korrekt getroffen hat – kaum bis nicht geschäftskritisch sind, kommen sie ins übliche Fehlermanagement-Tool und das Team behebt diesen Fehler im Rahmen der Wartung.

Wichtiger als die Fehlerfreiheit ist die Tatsache, dass der Endanwender die Software rechtzeitig einsetzen kann.

In der agilen Entwicklung hat Funktionalität meist Vorrang vor Qualität.

Prof. Mario Winter beschreibt das Thema Risiken und Testen in seinem Vortrag (Winter, 2009) folgendermaßen:

Was versteht man unter Risiken?

Risiken sind Probleme, die sowohl während der Entwicklung als auch beim Einsatz des Produkts möglicherweise eintreten können und unerwünschte Folgen haben könnten.

- Projektrisiken beziehen sich auf das Management und die Steuerung eines (Test-)Projekts, z. B. Mangel an personellen Ressourcen, zu enge Zeitrahmen, sich ändernde Anforderungen usw.
- Produktrisiken sind direkt auf das Software-Produkt bezogen und resultieren oft aus Qualitätsmängeln.

Projekt- und Produktrisiken werden dabei häufig nach der Software-Qualität nach DIN/ISO/IEC 9126 (Anm: diese Norm ist von ISO/IEC 25000 abgelöst worden) betrachtet.

Die Einflussfaktoren sind dabei sehr breit gestreut und können teilweise kaum beeinflusst werden:

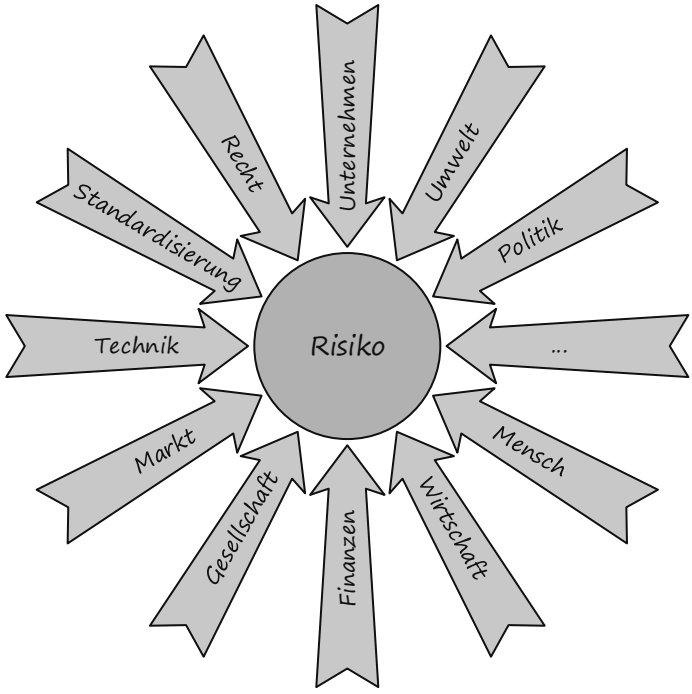


Bild 5.4 Risikoeinflussfaktoren

Unter value- bzw. risikobasiertem Testen versteht man also:

- **Zielgerichtet testen**, indem Systemfunktionen je nach Nutzen/Value, die diese Funktionen für das Unternehmen darstellen, betrachtet werden. Außerdem muss ergänzend die Risikostufe berücksichtigt werden, also die Wahrscheinlichkeit, dass dieses Risiko eintritt. Dabei sind sowohl die Wahl des Testverfahrens als auch die Testtiefe maßgeblich.
- **Priorisiert testen**, wobei Bereiche mit höherem Risiko bei der Testplanung eine höhere Priorität erhalten und entsprechend frühzeitig und intensiv getestet werden.
- In den Testberichten die **Restrisiken beziffern**, die bei einer Auslieferung der Software trotz Verkürzung des Tests oder Verzicht auf die Ausführung geplanter Tests verbleiben.

5.2.2 Explorativer Test

Exploratives Testen wird schon länger als Alternative bzw. als Ergänzung zum mechanischen Testen stur nach einer Methode gepriesen. Erfahrene Testexperten wie Cem Kaner und James Bach plädieren für einen kreativen Testansatz, nach dem der Tester von seiner Intuition und nicht von einer Methode oder von einem Werkzeug gesteuert wird (Kaner, Bach, & Pettichord, 2002). Ein guter, kreativer Tester wird ahnen, wo die Fehler liegen. Dieser Ansatz läuft auf eine Erforschung der Software hinaus. Der Tester stößt in das System hinein und verfolgt Pfade, die eventuell zu Fehlern führen können. Wenn er keine findet, kehrt er zurück und verfolgt einen anderen Pfad. Aufgrund seiner langjährigen Erfahrung wird der erfahrene Tester wissen, wo er zu suchen hat. Ähnlich wie es im Entwicklungsumfeld den Begriff „code smell“ gibt, den Entwickler riechen, wenn sie suboptimalen Code sehen, könnte man sagen, Tester nehmen den „bug smell“ wahr.

Dieser erfahrungsbasierte Ansatz wurde inzwischen bereits klarer strukturiert: Tester sind sensibilisiert auf Begriffe wie „Features“, „Complexity“, „Configuration“ und „Interoperability“; diese bieten dem Tester Anhaltspunkte, von wo aus er seine Touren durch das System starten kann.

James Whittaker vergleicht Exploratory Testing mit Touristen in einem fremden Land, die die Möglichkeit haben, die Gegend auf unterschiedlichen Touren zu entdecken (Whittaker, 2009), z. B.:

Landmark Tour: Hier nimmt man sich die Sehenswürdigkeiten, also die wichtigsten Features vor und testet diese auf vielen verschiedenen Wegen und Reihenfolgen – ganz wie Touristen die Sehenswürdigkeiten nach Interesse und Belieben besuchen.

Money Tour: Diese Tour hat in erster Linie die Funktionen im Visier, die für den Kunden den größten Nutzen bringen, quasi wie ein Kunstliebhaber, den in jeder Stadt primär die Museen interessieren.

Intellectual Tour: Das ist die Tour der Tüftler, die die Grenzen des Systems herausfinden wollen, also im Grunde der „traditionelle Testeransatz“, wo es primär darum geht, möglichst viele Fehler zu finden, und sei es auch mit besonders kniffligen Eingabekombinationen.

Saboteur Tour: Hier fühlen sich möglichst destruktive Tester wohl, die jede Situation nutzen, um das System gezielt zu Fall zu bringen.

Fedex Tour: Ähnlich wie bei der Paketverfolgung folgt man den Daten durch das System und achtet besonders darauf, dass sie an den richtigen Stellen ankommen.

Guidebook Tour: Diese Tour ist vor allem dann angebracht, wenn es zum System eine Dokumentation gibt. Man folgt rein den in der Doku angegebenen Wegen und findet so heraus, ob Beschreibung und System zusammenpassen und auch korrekt funktionieren.

James Whittaker hat in seinem Buch noch eine Reihe anderer Touren entdeckt, die Sie als Software-Touristen sicher interessieren können.

Dennoch bleibt das explorative Testen im Wesentlichen, was es ist: ein manueller Suchprozess nach Erfahrung und Instinkt. Entweder hat der Tester die richtige Erfahrung und den nötigen Instinkt oder eben nicht. Wenn er sie hat, kann er die Testzeit relativ zum Testergebnis wesentlich kürzen. Bringt er das nicht mit, ist dieser Testansatz nicht gerechtfertigt (Wallmüller, 2004).

Exploratives Testen, vor allem wenn es als session-basiertes Testen praktiziert wird, ist eine hoch effiziente Methode, die auch die Rollentrennung Tester von (Fach-)Experte ermöglicht.

5.2.3 Session-basiertes Testen

Damit exploratives Testen nachhaltiger und vor allem auch messbar wird, haben Jonathan & James Bach im Jahre 2000 eine eigene, erweiterte Testmethode entwickelt: das session-basierte Testen (Session Based Testing, SBT).

J. & J. Bach beschreiben ihre Beweggründe folgendermaßen:

„From a distance, exploratory testing can look like one big amorphous task. But it's actually an aggregate of sub-tasks that appear and disappear like bubbles in a Jacuzzi. We'd like to know what tasks happen during a test session, but we don't want the reporting to be too much of a burden. Collecting data about testing takes energy away from doing testing." (Bach J. , 2000)

Um die „Bubbles" nachvollziehbarer verfolgen zu können, geht man also prinzipiell nach dem explorativen Ansatz vor, folgt jedoch bei der Herangehensweise und Abwicklung einer vorab definierten Struktur, sogenannter Session Sheets. Das hat gleich mehrere Vorteile:

- Die Kreativität bei der explorativen Testdurchführung ist nicht eingeschränkt.
- Die Testläufe werden nachvollziehbar, da sie einer rudimentären Struktur folgen.
- Testergebnisse sind dadurch auch messbar.
- Wiederholbarkeit, also eine Art Regressionstests, ist möglich.

Ein weiterer großer Vorteil ist die flexible Anpassbarkeit. Erkennt das Team Optimierungspotenzial, adaptieren sie das Basis-Sessions-Sheet – fertig. Schon ist die Änderung vollzogen.

Dieser Testansatz findet sich oft auch in Unternehmen wieder, deren Testprozess noch nicht so ausgereift ist, aber deren Testteams dennoch eine mess- und kontrollierbare Abwicklung benötigen.

Wie läuft SBT nun ab?

Der session-basierte Testansatz (SBT) verlangt vom Tester, dass er kurze Arbeitsperioden plant, in denen er intensiv testet.

Zum Start der Session erhalten die Tester ein Debriefing vom Experten bzw. Team, in welchem die Eckdaten der durchzuführenden Testsession definiert werden – dazu hat sich das Session Sheet (lt. Bach) bewährt, welches folgende Kategorien beinhaltet:

- **Session Charter**: Diese inkludiert ein Mission Statement, eine Kurzdefinition, welches Ziel mit dem Test der aktuellen Session verfolgt werden soll. Das kann beispielsweise die Strategie/Tour [Entdeckung, Bug-Findung, Bug Retest, „Einfach drauflos"] sowie die Sessiondauer: [kurz: 60 Minuten; normal: 90 Minuten; lang: 120 Minuten] beinhalten.
- **Tester Name(s)**: Zuordnung, wer die Tests durchführt
- **Date and time**: Wann die Testsession startet
- **Task Breakdown**: Dahinter verbergen sich die Kennzahlen des Tests:
  - T(ime – Duration, also Dauer der Testsession)
  - B(ugs – Anzahl, die während der Session gefunden wurde)
  - S(ession Setup – also welche Rahmenbedingungen für diese Session wirken)
- **Datenfiles**: die benötigt werden bzw. angelegt wurden
- **Testnotizen**: Alles wichtig Erscheinende wird dokumentiert (Schritt für Schritt oder stichwortartig, welches Verhalten zeigte das System, welches hätte man erwartet? etc.).

  Erfahrene Tester können hier z. B. einzelne Testideen und Ansätze auch mit klassischen Testtechniken ergänzen: Gerade wenn beispielsweise Akzeptanzkriterien getestet werden, könnte das so aussehen:
- AC01: Schlüsselwerteingabe muss Abhängigkeiten zur Eingabe Feld xyz (lt. Anhang XY) erfüllen
  - Verifikation des Eingabefelds mittels Entscheidungstabelle (lt. Beiblatt)
  - Verifikation der unterschiedlichen Bereiche (lt. Grenzwertanalyse)
- **Issues**: Fragen, Auffälligkeiten ….
- **Bugs**: Fehler, die während der Testdurchführung/Session aufgefallen sind. Das kann entweder die Fehler-ID sein, wenn Fehler in einem Fehlermanagement-Tool erfasst wurden, oder man hängt dem Session Sheet alle erforderlichen Logfiles oder Screenshots an, die zur Analyse und Behebung benötigt werden.

Im Weiteren ein beispielhaftes Session Sheet aus einem unserer Übungsprojekte:

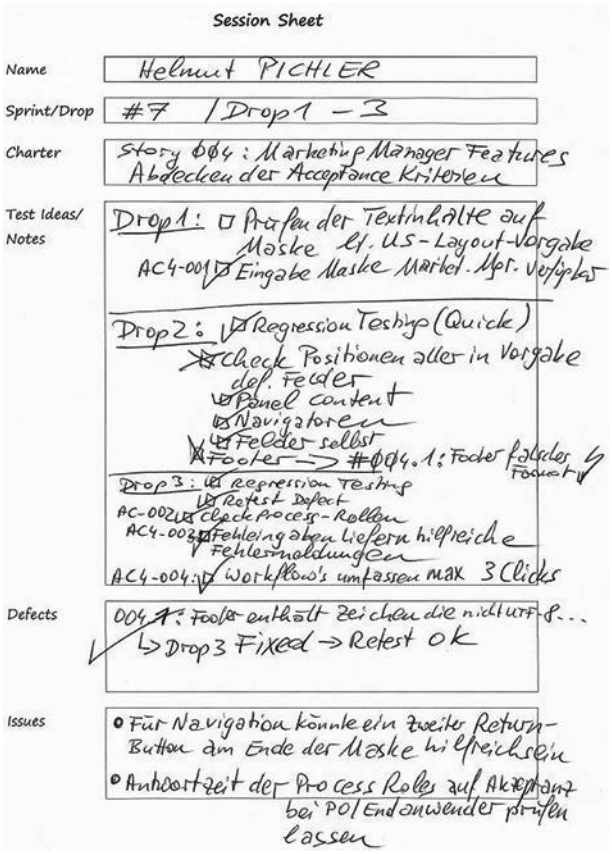


Bild 5.5 Übungsbeispiel für ein Session Sheet

Am Ende der Session berichten die Tester den Experten bzw. dem Team ihre Erfahrungen, was aufgrund der geordneten, wenngleich auch meist stichwortartigen Dokumentation der Ergebnisse effizient möglich ist.

Die daraus gewonnenen Erkenntnisse fließen dann an das Entwicklungsteam zurück. Sollte sich dabei herausstellen, dass Tester z. B. falsche Annahmen getroffen oder sich „verrannt" haben, lässt sich das hier auch rasch lösen und die Charter entsprechend anpassen.

Auf Basis der Testergebnisse entscheiden Experte bzw. Team über weitere Testbereiche und geben den Testern weitere Testaufträge (für weitere Sessions).

In dieser Methode sind risikobasiertes Testen, strukturiertes Testen, gute Dokumentation und sinnvolle Rollentrennung (Experten müssen nur relativ wenig Zeit investieren) vereint – also schlichtweg alles, was einem zur Erreichung höherer Qualität verhelfen kann.

In der Community ist SBT weit verbreitet, die SBT als Basis für weiterführende Techniken nutzen bzw. darauf aufbauen und eigene Kennzahlen darauf aufsetzen. Einen sehr interessanten Ansatz entwickelte z. B. James Lindsay (Lindsay, 2003), der basierend auf SBT ein Verfahren entwickelte, bei dem er mittels Ermittlung und Anwendung von Test Points Verbesserungsmodelle ableitet und somit mit einfachen Mitteln den Test gezielt steuern kann.

SBT ist also eine relativ einfache und sehr wirkungsvolle Methode, die den Test von agilen Projekten noch professioneller werden lässt.

Zusammenfassung

Session-basiertes Testen (SBT) ist:

- gemanagtes und kontrolliertes, exploratives Testen,
- limitierte Dauer (Time Boxed),
- explorative Ausrichtung,
- Mitprotokollieren der Aktivitäten.

5.2.4 Abnahmetestgetriebene Entwicklung

In einer abnahmetestgetriebenen Entwicklung, auch Acceptance Test Driven Development genannt, nimmt sich das gesamte Team bei jeder Iteration bzw. jedem Sprint im Sprint Planning Meeting oder bereits im laufend stattfindenden Grooming Story für Story vor und diskutiert dazu die vom Product Owner vordefinierten Akzeptanzkriterien. Anhand von konkreten Interpretationsbeispielen werden rudimentäre Beispiele – wie z. B. „Wenn der Kunde Sneed ein Buch mit dem Titel ‚Lange war es her' bestellt, bekommt er die Meldung, dass dieses Buch nicht mehr auf Lager ist" und weitere – besprochen.

Treten hier, also bereits ganz am Anfang des Sprints, unterschiedliche Sichten auf, dienen diese Beispiele dem Team als Diskussionsgrundlage bei der Abstimmung mit dem Product Owner.

Sobald das Team eine einheitliche Sicht und ein einheitliches Verständnis hat, wie die vorgegebenen Akzeptanzkriterien zu verstehen sind, formulieren die Tester aus dem Gesamt-Set an Beispielen heraus relevante Testfälle, die ab sofort als „konkrete Akzeptanzkriterien" gelten. Diese dienen künftig für die Abnahme, z. B. beim Sprint Review, als einzig gültige Akzeptanzkriterien.

Somit hat das gesamte Team von Anfang an diese Akzeptanzkriterien, um das gemeinsame Verständnis der zu betrachtenden Stories zu schärfen.

Elisabeth Hendrickson, eine der Master Minds zum agilen Testen, sieht es als eine der wichtigsten Praktiken im agilen Umfeld an (Hendrickson, 2008).

Gojko Adzic hat diesem Thema mit seinem Buch „Specification By Example" (Adzic, 2011) zu neuem Aufschwung verholfen. Adzic zeigt darin sehr anschaulich, welche Herangehensweise zum Erfolg führt, und würzt seine Empfehlungen mit sehr verständlichen Beispielen.

5.2.5 Testautomatisierung

Die Testautomatisierung ist eine unverzichtbare Begleiterscheinung des agilen Tests. Sie muss daher schon vor Projektbeginn eingeführt und geschult werden. Wenn die erste Iteration stattfindet, müssen die Tester schon damit umgehen können. Natürlich können nicht alle Testaufgaben automatisiert werden, aber ein großer Teil davon, darunter sämtliche wiederholbaren Aufgaben wie die Testdatengenerierung, die Testergebnisprüfung, die Testab-

laufprotokollierung und die Testlaufwiederholung. Der Tester sollte von diesen zeitaufwendigen Aktivitäten befreit werden, damit er sich auf die kreativen Aufgaben konzentrieren kann.

Die Testautomatisierung ist für Gärtner die Krönung des agilen Tests. Es mag sein, dass sie in jedem Projekt etwas anders ausfällt, aber jedes Projektteam muss dafür eine geeignete Lösung finden. Nur mittels der Automatisierung wird es den Testern gelingen, mit den Entwicklern Schritt zu halten, und dies sei die „Conditio sine qua non" für den Test in agilen Projekten. Es muss in der Kürze der Zeit einer Iteration das Maximum an Testüberdeckung erreicht werden.

„To automate or not to automate that's the question."

frei nach Shakespeare

Will man Projekte im agilen Umfeld erfolgreich abwickeln, würde das Zitat ergänzend lauten:

„… but if you want to be successful with agile projects there's no option: Automation is a MUST have"

Da dieses Thema gerade in agilen Projekten essenziell ist, haben wir diesem ein ganzes Kapitel (siehe Kapitel 7) gewidmet.

5.3 Wesentliche Einflussfaktoren auf den Test

Damit agile Teams überhaupt arbeiten können, sind einige Praktiken und Prozesse sicherzustellen. Permanente Integration sowie laufendes Deployment der zu entwickelnden Applikation, kurze Lieferzyklen, gemeinsame Codebasis, um nur einige zu nennen, müssen aufgesetzt sein und „wie geschmiert laufen". Denn erst, wenn dies „rund und stabil läuft", kann ein Team überhaupt an eine erfolgreiche Umsetzung von Stories denken.

Ich höre bereits Stimmen, die da sagen: „Das ist doch primär für die Entwickler relevant, also ein reines Entwicklungsthema!"

Stimmt, das hat zwar nicht unmittelbar etwas mit dem Testen zu tun, stellt aber die Grundlage für den gesamten Projektablauf dar. Vom Funktionieren dieser Werkzeuge bzw. Prozesse hängt es ab, ob das Team und somit auch die Tester regelmäßig die neuesten Versionen bekommen.

Schauen wir uns diese Prozesse und deren Geschichte etwas genauer an:

5.3.1 Continuous Integration (CI)

Der Begriff Continuous Integration ist ein Begriff aus der Software-Entwicklung und in der agilen Welt ein „Must have". Continuous Integration ist eine Praxis aus XP (Extreme Programming) und die Basis, um hoch qualitative Software zu liefern.

Wird in Projekten, die den klassischen Entwicklungsmodellen folgen, der Code-Stand zu einem späten Zeitpunkt dem Test bereitgestellt, um die notwendigen Testabläufe durchzuführen, verfolgen agile Methoden den Ansatz, die Software laufend zu testen, um auf diese Weise sehr rasch Fehler festzustellen und diese gleich zu beheben.

Erfahrungen aus diversen Software-Projekten zeigen, dass die späte Übergabe der Software an die Testteams viele Problemstellungen mit sich bringt. Es zeigt sich sehr häufig, dass die Software im Vorfeld nicht ausreichend durch die Entwicklung getestet worden ist. Die so übergebene Software an das Testteam enthält sehr oft noch Kinderkrankheiten, die Ursache für oftmalige, unnötige Zeitverzögerungen sind, bis die Software einen Stand aufweist, um einen reibungslosen Test zu ermöglichen. Folgewirkungen daraus sind höhere Kosten in der Testphase, erhöhter Personalaufwand, Verzögerungen bei der Auslieferung der Software an den Kunden sowie Qualitätsmängel im Betrieb der Software.

Continuous Integration ist ein Weg, um hier Abhilfe zu schaffen.

Der Prozess sieht relativ einfach aus:

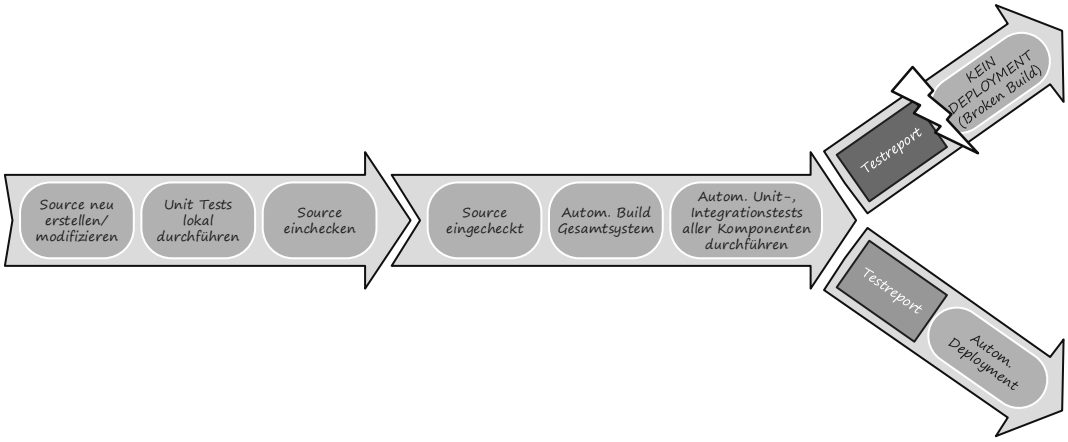


Bild 5.6 Continuous Integration – Build-Prozess (vereinfachte Darstellung)

Continuous Integration ist ein wesentlicher Bestandteil einer agilen Teststrategie: Angenommen, die Entwicklung arbeitet nach dem Test-First-Ansatz, dann passen die Entwickler ihre Codeteile so lange an, bis die lokal erstellten Unit Tests fehlerfrei ablaufen. Danach checken sie ihre neuen oder geänderten lauffähigen Source-Bausteine in das Konfigurationsmanagement-Tool (Versionsverwaltung) ein.

Der automatische Build

Nach jeder Änderung in der Source-Bibliothek startet üblicherweise sofort nach dem Einchecken ein neuer Build-Lauf, der die neuen Software-Builds automatisch nach einem vorgegebenen Schema versioniert und im Software-Repository ablegt. Danach startet das Tool ebenfalls automatisch alle im zentralen Repository enthaltenen Unit- und Integrationstest, die bisher erstellt und eingecheckt wurden.

Hier beginnen „Bauchkribbeln und Bangen“ zu wachsen: Laufen alle Testfälle noch erfolgreich durch – ist der Ergebnisbalken grün oder verursacht (m)eine Codeänderung Fehler in anderen Komponenten? Dies wird auch als Broken Build bezeichnet.

Treten Fehler im Build auf, herrscht im Team höchste Alarmstufe! Oberstes Ziel ist es nun, die Fehler so rasch wie möglich zu beseitigen.

„Unser oberstes Ziel nach jedem Build – das ist ganz klar –, ist die Erhaltung des Green Bar."
Schlachtruf aus einem unserer Projekte

Build-Prozess
Für die Administration des gesamten Build-Prozesses ist üblicherweise ein eigener Build-Manager verantwortlich. Dieser passt die Build-Routinen an, steuert, wann und wie viele Testfälle sowie ggf. auch ausgesuchte (kurze) fachliche Tests automatisiert gestartet werden, und definiert auch, wann komplexere funktionale und nichtfunktionale Tests im Rahmen des Build-Prozesses automatisch gestartet werden. In vielen Teams erfolgt dies meist nur einmal am Tag, und das bevorzugt über Nacht (Nightly Build). Das heißt, hier wird ein komplett neuer Build erstellt, es laufen alle Unit- und Integrationstests durch – wenn diese eine Green Bar produziert haben, laufen im Anschluss die aufwendigeren funktionalen Tests automatisiert durch. Am Ende stehen dem Team dann am frühen Morgen sowohl ein Ergebnisreport als auch eine komplett neue Version zur Verfügung.

Zum Thema Continuous Integration und Build-Prozess gibt es auch hervorragende, vertiefende Literatur wie z. B. (Pichler, Roock, & Havenstein, 2011), wo Andreas Havenstein das Thema Continuous Integration sehr detailliert und mit hohem Praxisbezug aufschlüsselt. Nach einer Einführung ins Thema geht er auf die Rahmenbedingungen ein, die für das Betreiben dieser Technik erforderlich sind. Weiterhin widmet er dem Thema Feedback ein umfangreiches Kapitel, da Feedback genaugenommen als Grundprinzip hinter der CI steht. Außerdem beschreibt er im Detail, welche Rahmenbedingungen bei der Einführung und beim „Leben" der CI beachtet werden sollten, z. B. was es mit der „Broken Windows Theory" bzw. dem „Stop the line"-Prinzip auf sich hat und warum das gerade bei agilen Teams ein Thema ist.

5.3.2 Automatisiertes Konfigurationsmanagement

Unternehmen, die ihre Software intern entwickeln, brauchen auf jeden Fall ein automatisiertes Konfigurationsmanagement mit Versionisierungsfähigkeit. Das Check-in- und Check-out-Verfahren sollte dafür sorgen, dass immer die aktuelle Version verfügbar ist und dass frühere Versionen archiviert werden. Ein weiterer Punkt, auf den der Anwenderbetrieb achten sollte, ist die Schnittstelle, an der die Software in die IT-Systeme des Unternehmens gelangt. Der Weg der Sourcen bis hin zur ausführbaren Komponente muss eindeutig festgelegt sein. Sowohl der Kunde als auch die agilen Lieferanten profitieren von definierten Prozessen und Infrastrukturstandards. Sie ermöglichen es, neue Releases in leicht integrierbaren Installationspaketen auszuliefern.

Wer seine Software selbst in installierbare Pakete umwandelt, sollte die Daten streng strukturieren und vereinheitlichen. Erst dann lassen sich die Applikationen mühelos in verschiedene Systemtest-, Integrationstest- sowie Produktionsumgebungen einspielen. Die erforderlichen Anpassungen erfolgen über Umgebungs- und Software-Parameter, ohne den Code selbst zu verändern. Die Information dazu soll bei der Auslieferung dokumentiert und, wenn möglich, automatisch bereitgestellt werden.

Die meisten Build-Tools geben Workflow und Integrationsstandards vor, die für alle Releases einheitlich sein sollten. In der Regel reicht hier ein einziges Werkzeug aus, um die wesentlichen Anforderungen zu erfüllen. Selbst wenn ein Team sich täglich abspricht, wie es bei Scrum- oder Kanban-Teams der Fall ist, sollten die Build-Skripte grundsätzlich versioniert werden. Nicht zuletzt, weil dieses Vorgehen Revisionssicherheit verleiht und weil es durch die Versionisierung möglich ist, Fixes für die Produktion automatisiert an neue Releases weiterzugeben. Ein ähnliches Verfahren wie beim Build lässt sich für das Deployment anwenden. Eine einzige übergreifende Installationsroutine fasst alle zentralen Schritte zusammen, die für das Deployment notwendig sind. Diese Overlay-Komponente ist für alle Projekte gleich und wird bereits im Build-Prozess in jedes Software-Paket integriert. Ein solches Vorgehen ist nicht nur weniger aufwendig und fehleranfällig als eine manuelle Anpassung für jede einzelne Applikation, sondern auch wesentlich schneller. Anwendungen werden jeweils am zentralen Overlay vorgenommen und gelangen automatisch in jedes einzelne Paket. Sollten Entwickler den Code während eines agilen Entwicklungsprozesses ändern, kann das Deployment-Tool sofort darauf reagieren.

Als weitere kosten- und zeitsparende Maßnahme bietet das Deployment-Tool eine sofortige Prüfung der Verfügbarkeit einer Applikation – Server, Datenbanken, Zielschema und die geforderten Queues. Ergeben sich bei der Auslieferung trotz erfolgreichen Tests noch Fehler, so ist deren wahrscheinliche Quelle bereits gut eingegrenzt, nämlich in der Infrastruktur.

5.4 Die besonderen Herausforderungen beim Test von IoT

Grundsätzlich hat das Thema Internet of Things (meist IoT abgekürzt) nicht unbedingt etwas mit agiler Vorgehensweise zu tun. Doch zum einen werden viele IoT-Entwicklungen agil durchgeführt und zum anderen stellt IoT den Test vor ganz besondere Herausforderungen, mit denen wir uns hier beschäftigen wollen.

5.4.1 Was ist das Internet of Things?

Mit Internet of Things (IoT) wird eine Vielzahl von sehr unterschiedlichen Anwendungsgebieten von Software zusammengefasst, bei denen viele einzelne Geräte (z. B. Sensoren, Steuergeräte, Haushaltsgegenstände, medizinische Geräte) über ein gemeinsames Netzwerk miteinander und mit einem zentralen Server kommunizieren und dadurch eine zentrale Erfassung und Steuerung ermöglichen. Die sicher bekanntesten Anwendungsgebiete sind zum Beispiel die Steuerung der Beleuchtung, der elektrischen Geräte, des Backofens etc. vom Mobiltelefon oder Tablet-PC aus. Die Palette reicht aber auch bis zu Sensoren, die die Beanspruchung und Defekte von ausgeliehenen Baugeräten an den Verleiher oder Hersteller weitergeben, so dass gezielt gewartet oder nach Nutzung abgerechnet werden kann.

Die Vernetzung von Implantaten bei Menschen (sog. Cyborgs) ist vergleichsweise noch in den Anfängen, aber auch schon Realität.

Laut Beecham Research (Beecham Research, 2017) gibt es neun verschiedene Branchengebiete, in denen ganz unterschiedliche Lösungen angeboten werden: Gebäudeverwaltung, Energieversorgung, Konsumartikel und Heim, Gesundheitswesen und Life Science, Industrie, Transport, Handel, persönliche und öffentliche Sicherheit, IT & Netzwerke. Das allein macht schon deutlich, wie breit das Anwendungsgebiet von IoT ist. Das Potenzial, das in vernetzten Einzelgeräten für die Wirtschaft und letztendlich für unsere Gesellschaft liegt, kann nicht hoch genug eingeschätzt werden. Dementsprechend wird IoT einer der neuen Megatrends der IT – zumal es mit anderen Megatrends wie Big Data und Cloud Services oft gekoppelt eingesetzt wird.

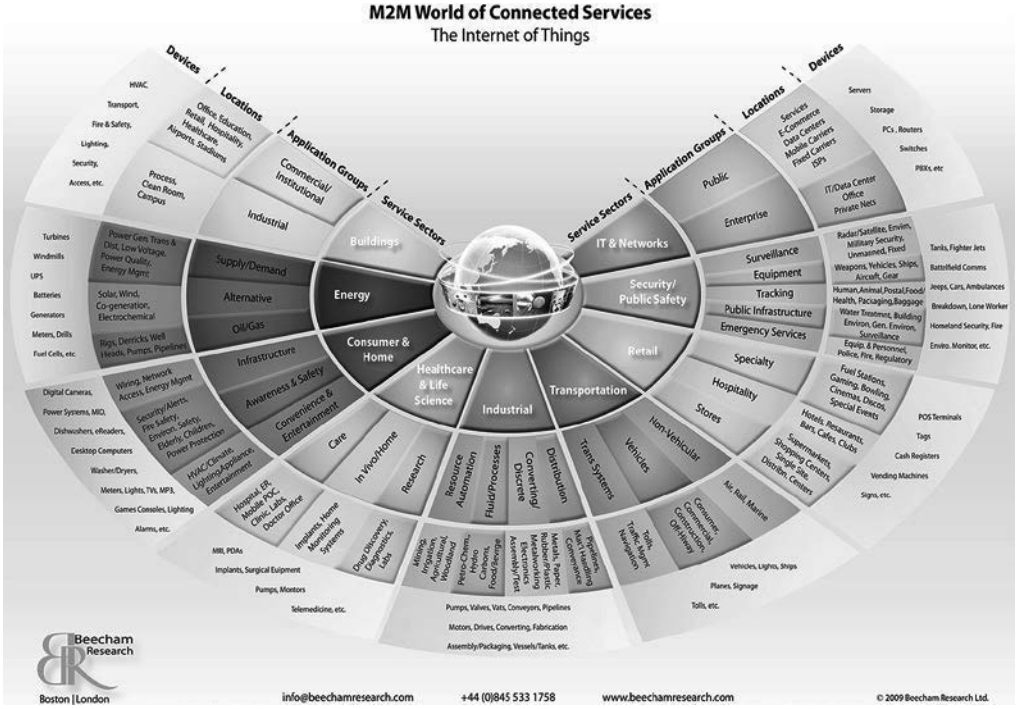


Bild 5.7 Die verschiedensten Anwendungsgebiete von IoT nach(Beecham Research, 2017)

Die IT-Lösung bei IoT besteht meist nicht nur aus der Software in den vernetzten Geräten oder Bauteilen, sondern auch in der Verwaltung des erforderlichen Netzwerks, zentralen Server-Applikationen zur Verarbeitung der Daten, Applikationen zur Auswertung der Informationen, Steuerungs-Applikationen und schließlich Cloud-Services für die flexible Anbindung. Somit sind IoT-Anwendungen häufig komplexe IT-Systeme, in denen die neuesten Technologien wie u. a. Mobile-Apps und Mikro-Services zur Anwendung kommen.

5.4.2 Die Herausforderung für agile Teams im Test

IoT benötigt extrem interdisziplinäre Teams, bei denen nicht nur in Bezug auf IT-Spezialisten (Netzwerkspezialisten, Sicherheitsspezialisten, Datenbankspezialisten, Mobile-App-Entwickler, …) interdisziplinär gedacht werden muss, sondern meist auch Spezialisten aus Elektrik und Maschinenbau wichtig sind – wenn es nicht sogar Mediziner braucht (wie bei Cyborgs oder Medizintechnik). Dies muss man nun bei der Organisation der agilen Teams stark beachten.

Naheliegend wäre es nun, der Teamgröße zuliebe das Testen ganz im klassischen Sinne von Scrum jedem Teammitglied als Aufgabe weitgehend selbst zu überlassen und auf den Testspezialisten im Team zu verzichten. Das Gegenteil ist aber ratsam: Es braucht gerade bei IoT jemanden, der sich der Testaufgabe ganzheitlich annimmt. Denn die Herausforderungen für den Test sind enorm: Folgende Herausforderungen für die Qualität der IoT Anwendungen sind typisch:

- Sicherheit und passende Auslegung der Datenverbindung (richtige und sichere Verschlüsselung, zuverlässige Kopplung, ausreichende Bandbreite)
- Richtige Nutzung von Big-Data-Lösungen für viele kleine Devices (Last- & Performance-Eigenschaften)
- Die User Experience für die Anwender richtig einschätzen und die Lösung darauf ausrichten
- Interoperabilität der Geräte und Austauschbarkeit/Erweiterung von Geräten bzw. Teilsystemen, kurz Kompatibilität
- Zuverlässigkeit, Stabilität und Robustheit der Lösung, sicherstellen; also der Umgang mit (Teil-)Ausfällen, Funklöchern etc.
- Gesteigerten regulatorischen Ansprüchen genügen, wie z. B. zum Schutz persönlicher Daten, zu Ausfallsicherheits- und Notfallplänen

Ein agiles Team ist gut beraten, wenn es im Team jemanden gibt, der sich des Qualitätsproblems von IoT-Lösungen umfassend annimmt, damit nicht im Eifer des Gefechts wichtige Qualitätsrisiken unberücksichtigt bleiben. Viele Anbieter von aktuellen IoT-Lösungen machen gerade hier sehr leidvolle Erfahrung, weil häufig dieser umfassende Blick vernachlässigt wurde.

Gut bewähren dürften sich auch Testspezialisten, die teamübergreifend beraten und zum Teil auch mitarbeiten. Eine Community of Practice zum Test, in der teamübergreifend zwischen den Testern der Teams ein Austausch zu den nötigen QS-Maßnahmen geführt wird, kann ebenso wichtig wie nützlich sein.

Wirklich neue Methodiken im Test braucht es bei IoT erstaunlicherweise nicht. Man muss nur wissen, wie man die Ansätze im richtigen Mix anwendet:

- Einsatz nichtfunktionaler Tests, wie
  - gezielte Last- und Performancetests, die systemübergreifend bzw. End-to-End Schwachstellen nachspüren,
  - Sicherheitstests, insbesondere Penetrationstests von gut qualifizierten Spezialisten,
  - Benutzbarkeitstests, um z. B. mit typischen Anwenderprofilen bestimmte Szenarien durchzuspielen,

- Zuverlässigkeitstests, die vor allem auch auf Bandbreitenschwankungen und Misskonfiguration beim Kunden abzielen,
- Robustheitstests, wo vor allem auch die Resilienz geprüft wird; so z. B. stabiles Verhalten auch bei Teilausfall des Gesamtsystems oder einzelner Sensoren

- Shift-left beim Testen:
  - Review auf Anforderungen und Entwurfsspezifikationen ernst nehmen bzw. das Sprint-Planning sehr effektiv (möglichst in mehreren Schritten) gestalten,
  - Tests von Teilsystemen so früh wie möglich ansetzen und stark mit Mockups oder Simulationsumgebungen arbeiten,
- kreative explorative Testsessions planen, bei denen man bewusst viele (ungünstige) Einflussfaktoren auf einmal zusammenbringt (z. B. Ausfall, Funkloch, Fehlbedienung, Misskonfiguration oder auch bewusste Störangriffe von außen) – eventuell auch bewusst Fehlerattacken als Testmethode anwenden

Zuletzt sei noch ein aus unserer Sicht sehr wichtiger Aspekt erwähnt: die Ethik. Diese findet sich zwar nicht als extra Qualitätsmerkmal und lässt sich auch nicht wirklich testen. Dennoch sollten Tester sie im Hinterkopf haben. So gibt es auch noch keinen allgemeinen „Code of Ethik“, wie z. B. in der Medizin. Aber man hat als Tester die Möglichkeit, sich an den z. B. im ISTQB-Training gelernten Ethik-Kodex von ACM und IEEE zu halten (ISTQB - International Software Testing Qualifications Board, 2011).

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z