HANSER

Leseprobe

zu

Agile Testing

von Manfred Baumgartner, Martin Klonk, Christian Mastnak and Richard Seidl

Print-ISBN: 978-3-446-47767-4
E-Book-ISBN: 978-3-446-47841-1
E-Pub-ISBN: 978-3-446-48030-8

Weitere Informationen und Bestellungen unter
https://www.hanser-kundencenter.de/fachbuch/artikel/9783446477674
sowie im Buchhandel

Inhalt

Geleitwort

Im Winter 2001 rief eine verschworene Clique bekannter Softwareentwickler in einer abgelegenen Skihütte im Bundesstaat Utah zu einer Revolution in der Softwarewelt auf. Sie schufen das Agile Manifest. Mit diesem Manifest definierte die Gruppe, was sie mit Extreme Programming bereits praktizierte. Mit ihrer schriftlichen Formulierung gelang ihnen ein publizistischer Coup, der weltweit Aufmerksamkeit erregte. Die auf dieser Skihütte versammelten Entwicklungsexperten hatten genug von starren Prozessregeln, unsinnigen bürokratischen Richtlinien und weltfremden Vorgehensweisen der damaligen Software-Engineering-Disziplin. Sie erkannten, dass monotones „Arbeiten nach Vorschrift“ in der neuen, schnelllebigen Zeit nicht mehr zeitgemäß war. Sie wollten sich von den Fesseln der Projektbürokratie befreien, um Software gemeinsam mit den Anwendern nach Bedarf zu entwickeln. Die bisher schwerfällige, phasenorientierte, dokumentengetriebene Softwareentwicklung sollte durch eine flexible, menschengesteuerte Entwicklung in kleinen, überschaubaren Schritten ersetzt werden. Agile Softwareentwicklung sollte der Ansatz des neuen Jahrhunderts sein!

Bei der agilen Entwicklung steht nicht das Projekt, sondern das Produkt im Mittelpunkt. Da die Softwareentwicklung mehr und mehr zu einer Expedition ins Unbekannte wurde, sollte das Produkt nach und nach in kleinen, inkrementellen Schritten erstellt werden. Anstatt ausufernde Anforderungsdokumente über Dinge zu schreiben, die man zu diesem Zeitpunkt noch nicht wissen konnte, sollte man lieber etwas programmieren, das einem zukünftigen Benutzer schnelles Feedback entlocken kann. Es sollte nicht Monate oder gar Jahre dauern, bis man feststellt, dass das Projekt auf dem falschen Weg oder das Projektteam mit der Aufgabe überfordert ist. Dies sollte innerhalb weniger Wochen erkannt werden.

Das Grundprinzip der agilen Entwicklung ist also die inkrementelle Bereitstellung. Ein Softwaresystem soll Stück für Stück fertiggestellt werden. Dies gibt dem Benutzervertreter im Team die Gelegenheit, sich an jedem Schritt der Entwicklung zu beteiligen. Nach jeder neuen Lieferung kann er das gelieferte Zwischenprodukt mit seinen Vorstellungen abgleichen. Das Testen ist somit in den Prozess integriert. Von Anfang an wird die Software kontinuierlich getestet. Ursprünglich wurde offengelassen, ob ein Tester auch am agilen Prozess teilnehmen sollte. Die Autoren des Agilen Manifests sprachen sich gegen eine strikte Arbeitsteilung aus. Die Aufteilung in Analysten, Designer, Entwickler, Tester und Manager erschien ihnen zu künstlich und sie befürchteten, dass dadurch zu viele Reibungsverluste entstehen. Natürlich sollte das Projektteam über all diese Fähigkeiten verfügen, aber die Rollen innerhalb des Teams sollten austauschbar sein. Das Entwicklungsteam sollte für alles verantwortlich sein. Die Rolle eines speziellen Testers im Team wurde erst durch die

Beiträge von Lisa Crispin und Janet Gregory eingeführt. Sie setzten sich dafür ein, dass sich jemand im Team um Qualitätsfragen kümmert.

Die Entwicklung von Software erfordert sowohl Kreativität als auch Disziplin. Gegen Ende des letzten Jahrhunderts hatten die Verfechter von Ordnung und Disziplin die Oberhand und bremsten mitunter die Kreativität der Entwickler durch starre Prozesse und Qualitätssicherungsmaßnahmen aus. Doch wenn etwas übertrieben wird, schlägt das Pendel zurück. Zu viel Disziplin wurde den traditionellen Entwicklungsprojekten aufgezwungen. Die Reaktion darauf ist die agile Bewegung, die Spontaneität und Kreativität in die Softwareentwicklung zurückbringen soll. Das ist sicherlich zu begrüßen, aber auch dieser Aspekt darf nicht übertrieben werden. Die Kreativität von Anwendern und Entwicklern sollte geerdet bleiben – und ein erfahrener, unparteiischer Tester kann dies unterstützen.

Jedes Entwicklungsteam sollte mindestens einen Tester haben, der die Interessen der Qualität vertritt. Dieser stellt sicher, dass der resultierende Code und das Produkt sauber bleiben und die vereinbarten Qualitäts- oder Abnahmekriterien erfüllen. In dem Drang, schneller voranzukommen, können nichtfunktionale Qualitätsanforderungen hinter den funktionalen Anforderungen zurückbleiben. Es ist die Aufgabe eines Testers, dem Team zu helfen, ein Gleichgewicht zwischen Produktivität und Qualität zu wahren. Der Tester ist quasi der gute Geist, der das Team davor bewahrt, Fortschritte auf Kosten der Qualität zu machen. Mit jedem neuen Release soll nicht nur Funktionalität hinzugefügt, sondern auch die Qualität erhöht werden. Der Code soll regelmäßig bereinigt bzw. refaktoriert, dokumentiert und von allen Mängeln befreit werden. Es ist die Aufgabe des Testers, das ganze Team dazu zu befähigen und dafür zu sorgen, dass dies geschieht.

Natürlich hat die agile Projektorganisation auch Auswirkungen auf das Testen und die Qualitätssicherung. Die für die Qualitätssicherung zuständigen Personen sitzen nicht mehr in einem entfernten Büro, von wo aus sie die Projekte überwachen, die Projektergebnisse zwischen den Phasen prüfen und das Produkt in der letzten Phase testen, wie es bei traditionellen Projekten oft der Fall war. Sie sind jetzt fest in die Entwicklungsteams integriert, wo sie ständig die neuesten Ergebnisse überprüfen und validieren. Es ist ihre Aufgabe, auf Mängel in der Architektur und im Code hinzuweisen und eventuelle Fehler im Verhalten des Systems zu erkennen. Sie weisen auf Probleme hin und helfen dem Team, die Qualität der Software zu verbessern. Die Rolle des Testers entwickelt sich zu einem agilen Qualitätscoach, der das Team in allen qualitätsrelevanten Fragen unterstützt. Im Gegensatz zu dem, was in den Anfängen der agilen Bewegung von einigen behauptet wurde („Tester sind in agilen Projekten nicht mehr notwendig“), ist ihre Rolle heute wichtiger denn je. Ohne ihren Beitrag wachsen die technischen Schulden und diese bringen das Projekt früher oder später zum Stillstand.

Dieses Buch beschreibt das agile Testen in zehn Kapiteln. Das erste Kapitel schildert den Kulturwandel, den die agile Entwicklung mit sich gebracht hat. Mit dem Agilen Manifest wurden die Weichen für eine Neugestaltung der IT-Projektlandschaft gestellt. Die Entwicklung sollte nicht mehr starr nach dem Phasenkonzept erfolgen, sondern flexibel und in kleinen Iterationen. Nach jeder Iteration ist vom Team ein lauffähiges Teilprodukt zu präsentieren. Auf diese Weise werden Lösungen erforscht und Probleme frühzeitig erkannt. Die Rolle der Qualitätssicherung ändert sich. Anstatt als externe Instanz für die Projekte zu agieren, werden die Tester in das Projekt eingebettet, so dass sie ihre Tests unmittelbar vor Ort als Teilnehmer am Entwicklungsprozess durchführen können. Natürlich müssen die Unternehmen ihre Managementstrukturen entsprechend anpassen: Anstatt abseits auf ein

Endergebnis zu warten, sind die Fachanwender gefordert, sich aktiv am Projekt zu beteiligen und die Entwicklung über ihre Anforderungen, die „Storys“, 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 Testern zusammen, um sicherzustellen, dass das Produkt ihren Erwartungen entspricht.

Letztlich müssen sich alle – Entwickler, Tester und Benutzer – anpassen, um das gemeinsame Ziel zu erreichen. Viele traditionelle Rollen, wie die des Projektleiters und des Testmanagers, sind im Schwinden oder zumindest im Wandel begriffen. Dafür gibt es neue Rollen, wie Scrum Master und Lead Tester oder Agile Quality Coach. Projektmanagement im traditionellen Sinne findet nicht mehr auf Teamebene statt. Jedes Team verwaltet sich selbst. Die IT-Welt verändert sich und damit auch die Art und Weise, wie Menschen Software entwickeln.

Im zweiten Kapitel, das sich mit agilen Vorgehensmodellen befasst, konzentrieren sich die Autoren auf die Rolle der Qualitätssicherung in agilen Entwicklungsprojekten. Sie scheuen sich nicht, die verschiedenen Zielkonflikte objektiv zu betrachten, zum Beispiel zwischen Qualität und Termintreue, zwischen Qualität und Budget und zwischen Qualität und Funktionalität. Die Vereinbarkeit dieser Zielkonflikte ist eine der Herausforderungen für das agile Testen.

Entgegen der immer noch weit verbreiteten Meinung, dass in agilen Projekten nicht viel getestet werden muss, ist in Wirklichkeit sehr viel Testarbeit erforderlich. Testgetriebene Entwicklung (TDD) sollte nicht nur für den Unittest, sondern auch für den Integrationstest und den Systemtest gelten, nach dem Motto: erst die Testfälle, dann der Code. In diesem Fall heißt das: erst die Testspezifikation, dann die Implementierung. Die Testautomatisierung spielt dabei eine entscheidende Rolle. Nur wenn ein Test automatisiert wird, kann die geforderte Qualität in der nötigen Geschwindigkeit erreicht werden. An der Automatisierung sollte das gesamte Team beteiligt sein, denn ein Tester allein wird es nicht bewältigen können. Neben dem Test sind zu bestimmten Zeiten während der Entwicklung des Softwareprodukts auch Audits erforderlich. Ziel der Audits ist es, Schwachstellen und Mängel in der Software aufzudecken. Ein guter Zeitpunkt dafür ist am Ende jedes Sprints in einem Scrum-Projekt (Retrospektive). Das Team legt dann anhand der Ergebnisse der Audits die Prioritäten für den nächsten Sprint fest. Diese kurzen Audits, oder Momentaufnahmen der Produktqualität, können von externen QS-Experten in Zusammenarbeit mit dem Team durchgeführt werden. Ziel ist es, dem Team zu helfen, Risiken rechtzeitig zu erkennen.

Neben dem Scrum-Prozess wird im zweiten Kapitel auch auf Kanban und den schlanken Softwareentwicklungsprozess (Lean Software Development) eingegangen. Anhand von Beispielen aus der Projekterfahrung der Autoren erhält der Leser zahlreiche Tipps, wie er die Qualitätssicherung in diese Prozesse einbinden kann.

Das dritte Kapitel beschäftigt sich mit der agilen Testorganisation und der Positionierung des Testens in einem agilen Team. Zu diesem Thema gibt es recht unterschiedliche Ansichten. Mit Hilfe der vier Testquadranten von Crispin und Gregory untersuchen die Autoren, welcher Test zu welchem Zweck passt. Dabei wird zum einen die Frage gestellt, ob der Test geschäfts- oder technologieorientiert ist, und zum anderen, ob er sich auf das Produkt oder das Team bezieht. Daraus lassen sich vier Testarten ableiten:

1. Unit- und Komponententest: technologieorientiert/teamunterstützend
2. Funktionstest: geschäftsorientiert/teamunterstützend
3. Explorativer Test: geschäftsorientiert/produktbezogen
4. Nicht-funktionaler Test:.technologieorientiert/produktbezogen

Zur Verdeutlichung dieser Testansätze wird anhand von Beispielen aus der Testpraxis gezeigt, welche Art von Test welchem Zweck dient.

In diesem Kapitel gehen die Autoren auch auf das Thema Test in skalierenden, agilen Modellen und Frameworks, wie SAFe oder LeSS ein. Sie betonen dabei, wie wichtig es ist, den Testprozess nach Belieben erweitern zu können. Es gibt Kernaktivitäten, die stattfinden müssen, und Randaktivitäten, die je nach Ausbaustufe hinzugefügt werden. Es gibt also nicht nur eine, sondern viele mögliche Organisationsformen, die von der Art des Produkts und den Projektbedingungen abhängen.

Das Umfeld, in dem das Projekt stattfindet, und die Produkteigenschaften wie Größe, Komplexität und Qualität sind entscheidend für die Wahl der geeigneten Organisationsform. In jedem Fall darf das Hauptziel, nämlich die Unterstützung der Entwickler, nicht aus den Augen verloren werden. Das Ziel aller Testansätze ist es, Probleme möglichst schnell und gründlich aufzudecken und die Entwickler auf unaufdringliche Weise zu informieren. Wenn mehrere agile Projekte nebeneinander laufen, empfehlen die Autoren die Einrichtung eines Test Competence Center. Aufgabe dieser Instanz ist es, die Teams in allen Fragen der Qualitätssicherung zu unterstützen, zum Beispiel welche Methoden, Techniken und Werkzeuge sie einsetzen sollten. Am Ende des Kapitels werden zwei Fallstudien zur Testorganisation vorgestellt, eine aus dem Telekommunikations- und eine aus dem Gesundheitsbereich. In beiden Fällen orientiert sich die Testorganisation an der Projektstruktur und den jeweiligen Qualitätszielen.

In Kapitel 4, „Die Rolle des Testers in agilen Projekten“, wird die Frage aufgeworfen, ob ein agiler Tester ein Generalist oder ein Spezialist sein sollte. Die Antwort lautet, wie so oft in der Literatur zur agilen Entwicklung: sowohl als auch. Es kommt auf die jeweilige Situation an. Es gibt Situationen, wie z. B. zu Beginn einer Iteration, in denen der Tester mit dem Benutzer oder dem Product-Owner über Akzeptanzkriterien verhandelt, in denen der Tester sowohl technisches als auch allgemeines Wissen benötigt. Es gibt andere Situationen, in denen der Tester mit automatisierten Testwerkzeugen umgehen muss, für die er spezielle technische Kenntnisse benötigt. Ein agiler Tester muss in der Lage sein, viele Rollen zu übernehmen, doch das Wichtigste ist, dass der Tester sich als Teamplayer in das Team einfügt, ganz gleich, welche Rolle er gerade übernehmen muss. Soft Skills sind eine unabdingbare Voraussetzung. In jedem Fall sind die Tester die Verfechter der Qualität und müssen dafür sorgen, dass die Qualität erhalten bleibt, auch wenn die Zeit drängt. Dazu müssen sie an allen Diskussionen über die Produktqualität teilnehmen und gleichzeitig die Software prüfen und testen. Sie sollten Probleme rechtzeitig aufdecken und dafür sorgen, dass sie so früh wie möglich behoben werden. Natürlich können die Tester dies nicht allein tun; sie sind auf den Beitrag der anderen Teammitglieder angewiesen. Deshalb müssen Tester als eine Art Qualitätscoach fungieren und ihren Teamkollegen helfen, Probleme zu erkennen und zu lösen. Schließlich liegt die Qualität der Software in der Verantwortung des gesamten Teams.

Im Zusammenhang mit der Rolle des Testers in einem agilen Team befasst sich das Kapitel mit dem Erfahrungsprofil der Mitarbeiter. Wie sehen die Karrieremodelle in der agilen Welt

aus? Tatsache ist, dass es in der agilen Entwicklung keine festen Rollen mehr gibt. Die Rollen ändern sich je nach Situation, auch die des Testers. Mitarbeiter, die ausschließlich Erfahrungen mit traditionellen Entwicklungsmethoden haben, können sich nicht mehr auf traditionelle Rollen zurückziehen, sondern müssen sich anpassen. Dies wird nicht für jeden Mitarbeiter einfach sein. Die Autoren schlagen ein Trainingsprogramm vor, das auf die Rolle des agilen Testers vorbereitet. In dem Programm betonen sie positive Erfahrungen und schließen mit der Zuversicht, dass flexible Mitarbeiter, ob alt oder jung, in die Rolle des agilen Testers hineinwachsen können.

In Kapitel 5 wenden sich die Autoren den Methoden und Techniken des agilen Testens zu. Dabei betonen sie die Unterschiede zum herkömmlichen, phasenorientierten Testen. Der Prozess beginnt mit der Testplanung, wobei der Plan viel unverbindlicher ist als in traditionellen Projekten. Er sollte flexibel bleiben und leicht zu aktualisieren sein. Agiles Testen ist viel stärker mit der Entwicklung verwoben und kann nicht mehr separat als Teilprojekt im Projekt betrachtet werden. In jedem Entwicklungsteam sollte es mindestens einen Tester geben, der dort voll integriert ist. Die Tester sollten nur dem Team gegenüber verantwortlich sein. Es kann einen projektübergeordneten Testmanager außerhalb des Teams geben, der als Bezugsperson für die Tester in mehreren Teams dient; aber dieser darf keinen Einfluss auf die Arbeit innerhalb des Teams haben und hat allenfalls eine beratende Funktion. Die bisherige Planung, Organisation und Steuerung eines separaten Testteams unter der Leitung eines Teammanagers sind nicht mehr notwendig. Sie passen nicht zur agilen Philosophie der Teamarbeit.

Bei den Testmethoden werden die Methoden hervorgehoben, die sich am besten für den agilen Ansatz eignen: risikobasiertes Testen, wertgetriebenes Testen, exploratives Testen, sitzungsbasiertes Testen und abnahmetestgetriebene Entwicklung. Konventionelle Testtechniken wie Äquivalenzklassenbildung, Grenzwertanalyse, Zustandsanalyse und Entscheidungstabellen oder -bäume sind nach wie vor wichtig und werden durch agile Testmethoden nicht abgelöst, sondern in diesen angewendet. Die Bedeutung der Wiederverwendung von Tests und der Wiederholung von Tests wird betont. Alle Techniken müssen diese Kriterien erfüllen. Der Integrationstest ist eine fortlaufende Geschichte und der Abnahmetest wird immer wieder wiederholt. Die zyklische Natur eines agilen Projekts führt zu einer Neudefinition der Testendekriterien. Der Test endet nie – solange das Produkt weiterwächst.

In Kapitel 6, „Agile Testdokumentation“, beschreiben die Autoren, welche Dokumente die Tester in einem agilen Projekt erstellen müssen, sollen oder können. Dazu gehören eine testbare Anforderungsspezifikation aus den User-Storys, ein Testdesign, eine Benutzerdokumentation und Testberichte. Die Testfälle gelten nicht als Dokumentation, sondern als Testware. Ein Anliegen der agilen Entwicklung ist es, die Dokumentation auf ein Minimum zu reduzieren. In der Vergangenheit wurde es mit der Dokumentation tatsächlich übertrieben. In einem agilen Entwicklungsprojekt wird nur noch das dokumentiert, was unbedingt notwendig ist. Ob eine Teststrategie oder ein Testdesign notwendig ist, sei dahingestellt. Testfälle sind unerlässlich, aber sie sind ebenso Teil des Softwareprodukts wie der Code. Daher werden sie nicht als Dokumentation betrachtet.

Das wichtigste Dokument ist die Anforderungsspezifikation, die sich aus den User-Storys ergibt. Sie dient als Grundlage für den Test, das sogenannte Testorakel. Aus ihr werden die Testfälle abgeleitet und gegen sie wird getestet. Sie enthält auch die Abnahmekriterien. Die einzigen Testberichte, die wirklich benötigt werden, sind der Testüberdeckungsbericht und

der Fehlerbericht. Der Testüberdeckungsbericht zeigt, was getestet wurde und was nicht. Die Tester benötigen dieses Dokument als Nachweis, dass die Tests ausreichend durchgeführt wurden. Der Benutzer braucht ihn, um Vertrauen in das Produkt zu gewinnen. Im Fehlerbericht wird festgehalten, welche Abweichungen aufgetreten sind und wie sie behandelt werden. Diese beiden Berichte sind die besten Indikatoren für den Stand des Tests.

Tester wären prädestiniert, das Benutzerhandbuch zu schreiben, weil sie das System am besten kennen und wissen, wie man es benutzt. Jemand muss das Handbuch schreiben und die Tester sind die richtigen Kandidaten dafür. Sie sorgen dafür, dass dieses Dokument nach jeder Iteration oder jedem Release aktualisiert wird. Ansonsten folgt das Buch dem agilen Prinzip, die Dokumentation auf das Wesentliche zu beschränken. Der wichtigste Aspekt ist, dass es immer eine solide Anforderungsspezifikation und eine verständliche Benutzerdokumentation gibt. Eine strukturierte, semiformale Anforderungsspezifikation bildet die Basis für den Test und kein Anwender möchte auf eine Benutzeranleitung verzichten.

Kapitel 7 beschäftigt sich mit dem wichtigen Thema „Testautomatisierung“. Testautomatisierung ist in der agilen Entwicklung besonders wichtig, da sie das wichtigste Werkzeug zur Projektbeschleunigung darstellt und zur Unterstützung moderner DevOps-Ansätze unerlässlich ist. Nur durch Automatisierung kann der Testaufwand auf ein akzeptables Maß reduziert werden, während bei kontinuierlicher Integration (Continuous Integration) die Produktqualität erhalten bleibt. Die Autoren unterscheiden dabei zwischen Unittest, Komponenten-Integrationstest und Systemtest. Der Unittest wird am Beispiel von JUnit detailliert dargestellt. Es wird gezeigt, wie Entwickler testgetrieben arbeiten müssen, wie man Testfälle aufbaut und wie man die Testüberdeckung misst. Der Komponenten-Integrationstest wird am Beispiel des Apache Maven Integrationsservers erläutert. Dabei ist es wichtig, die Schnittstellen der integrierten Komponenten zu den noch nicht vorhandenen Komponenten über Platzhalter zu simulieren. Der Systemtest wird durch einen fachlichen Test mit FitNesse beschrieben. Das Entscheidende dabei ist die Umsetzung der Testfälle in Testskripte, die beliebig erweitert und wiederholt werden können. Die Autoren betonen auch, wie wichtig es ist, die Testware – Testfälle, Testskripte, Testdaten etc. – bequem und sicher verwalten zu können, damit der Test möglichst reibungslos abläuft. Auch dafür werden Werkzeuge benötigt.

Kapitel 8 fügt Beispiele aus der Testwerkzeugpraxis hinzu und erweitert die Testautomatisierung um Testmanagementfunktionalität. Zunächst wird das Werkzeug Broadcom Rally beschrieben, das den agilen Lebenszyklus von der Verwaltung der Storys bis zum Fehlermanagement unterstützt. Der agile Tester kann dieses Werkzeug zur Planung und Steuerung seiner Tests verwenden. Eine Alternative zu Broadcom Rally ist Polarion QA/ALM, das sich besonders für die Erfassung und Priorisierung von Testfällen und für die Fehlerverfolgung eignet. Weitere Testplanungs- und Tracking-Tools sind die Tools Pachno, das insbesondere die Testaufwandsschätzung unterstützt, Atlassian JIRA, das eine umfangreiche Fehleranalyse bietet, und Azure Test Plans.

Für Tester in einem agilen Projekt ist der kontinuierliche Integrationstest entscheidend. Er muss die letzten Komponenten so schnell wie möglich mit den Komponenten des letzten Release integrieren und bestätigen, dass sie reibungslos zusammenarbeiten. Dazu muss er die Tests nicht nur über die Benutzeroberfläche, sondern auch über die internen Systemschnittstellen durchführen. Mit Tricentis Tosca können sowohl externe als auch interne Schnittstellen generiert, aktualisiert und validiert werden.

Die Autoren beschreiben anhand ihrer eigenen Projekterfahrungen, wie diese Werkzeuge eingesetzt werden und wo ihre Grenzen liegen.

Das neunte Kapitel des Buchs ist dem Thema „Ausbildung und ihre Bedeutung“ gewidmet. Die Autoren betonen die Rolle der Mitarbeiterschulung für den Einstieg in die agile Entwicklung. Schulungen sind für den Erfolg bei der Anwendung der neuen Methoden unerlässlich und dies gilt insbesondere für die Tester. Die Tester in einem agilen Team müssen genau wissen, worauf sie sich konzentrieren müssen, und das können sie nur durch entsprechende Schulungen lernen. Es gibt viele Interpretationen von agilen Ansätzen, dennoch muss die Qualität des Produkts sichergestellt werden – und dazu braucht es professionelle Tester, die für die Arbeit in einem agilen Team ausgebildet sind. Ausbildungsprogramme, die auf die Bedürfnisse des agilen Testens ausgerichtet sind, werden in diesem Kapitel diskutiert.

Das zehnte Kapitel „Retrospektive“ blickt noch einmal auf die Gründe und die Motivation hinter der agilen Revolution zurück und weist nachdrücklich darauf hin, dass die agile Transformation ein Veränderungsprozess ist, der nicht nur einzelne Projektteams, sondern die gesamte Organisation betrifft.

Zusammenfassend deckt dieses Buch die wesentlichen Aspekte des agilen Testens ab und bietet einen wertvollen Leitfaden für das Testen in einer agilen Umgebung. Der Leser erhält viele Anregungen, wie er in agilen Projekten vorgehen kann. Er lernt, wie man das agile Testen vorbereitet und durchführt. Als Buch, das von Testpraktikern geschrieben wurde, hilft es Testern, sich in der oft verwirrenden agilen Welt zurechtzufinden. Es gibt ihnen klare, fundierte Anleitungen für die Umsetzung agiler Prinzipien in der Testpraxis. Es gehört damit in die Bibliothek jeder Organisation, die agile Projekte durchführt.

Harry M. Sneed

Vorwort

Im Jahr 2013 erschien die erste Auflage von „Agile Testing – Der agile Weg zur Qualität“, 2018 folgte die zweite Auflage. Jetzt, im Jahr 2024, also sechs Jahre später, freut es uns, die bereits dritte Auflage präsentieren zu können. Es überrascht nicht, dass auch in diesem Zeitraum unglaublich rasante Entwicklungen im Bereich der Softwareentwicklung stattgefunden haben. Agilität ist nicht mehr nur ein Thema für einzelne Entwicklungsprojekte und -teams, sondern hat die Unternehmen als Ganzes erfasst. Das Mindset in der Softwareentwicklung, insbesondere in Bezug auf das Testen und die Rolle der Tester, hat sich – aus unserer Sicht – sehr positiv verändert. Den Herausforderungen großer, skalierender Projekte und Programme wird mit entsprechenden Modellen begegnet. Automatisierung in allen Phasen des Softwarelebenszyklus und DevOps sind integrale Bestandteile vieler Vorgehensmodelle in den Unternehmen. Und es gibt auch noch eine Vielzahl von Unternehmen, die eher in traditionellen oder überwiegend hybriden Ansätzen arbeiten und damit erfolgreich sind. Auch für sie bietet dieses Buch Ansätze für die agile Transformation der Softwareentwicklung im Hinblick auf die Qualitätssicherung.

Deshalb haben wir in dieser Ausgabe von Agile Testing die Aspekte Mindset, Skalierung und DevOps erweitert und wollen auch betonen, dass agiles Testen keineswegs eine Abkehr von altbewährten Testtechniken bedeutet, sondern vielmehr eine Anwendung dieser Handwerkszeuge in einem veränderten Vorgehen.

Als das „Manifest für agile Softwareentwicklung“ im Jahr 2001 von einer Gruppe von Softwareingenieuren in Utah/USA unterzeichnet wurde, leitete es den wohl bedeutendsten Wandel in der Softwareentwicklung seit der Einführung der Objektorientierung Mitte der 1980er-Jahre ein. Das „Agile Manifest“, quasi die Zehn Gebote der agilen Welt, kann durchaus als Ausdruck einer Gegenbewegung zu den sich ab Ende der 1980er-Jahre verbreitenden, stark regulierenden Prozess- und Planungsmodellen wie PRINCE, dem V-Modell oder auch der ISO9001 gesehen werden. Diese Modelle versuchten, den bis dahin chaotischen und willkürlichen Entwicklungsprozessen durch Planung, Strukturierung der Prozesse und Dokumentation entgegenzuwirken. Das Agile Manifest positioniert sich bewusst zu diesen Aspekten und gibt seinen zentralen vier agilen Werten – Interaktion, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderungen und schließlich funktionierende Software – eine höhere Relevanz für eine erfolgreiche Softwareentwicklung.

Die Art und Weise, wie das Agile Manifest in Werten und Prinzipien formuliert ist, war ein Grund dafür, dass der Siegeszug der agilen Softwareentwicklung in den Jahren seit der Veröffentlichung des Agilen Manifests von vielen Glaubenssätzen geprägt war. Wir, die Autoren dieses Buchs, erlebten dies nicht zum ersten Mal. In den Jahrzehnten unserer Be-

rufserfahrung sind wir immer wieder mit neuen Lösungen für das „Softwareproblem“ konfrontiert worden: strukturierte Programmierung, objektorientierte Programmierung, CASE (Computerunterstützte Softwareentwicklung), RUP (Rational Unified Process), V-Modell, ISO9001, SOA (Service-Orientierte Architektur) … – eine lange Liste sogenannter „Silver Bullets“, die die Probleme lösen sollten. Allerdings wuchsen die Herausforderungen und ihre Komplexität schneller, als sich neue Ansätze etablieren konnten. So können wir uns bereits jetzt auf künftige Ansätze freuen, die beispielsweise auf den Konzepten der künstlichen Intelligenz und maschinellem Lernen basieren.

Auf dem Weg dieser ständigen Weiterentwicklung ist man jedoch gut beraten, nicht immer das Kind mit dem Bade auszuschütten. Vor zehn Jahren war eine Motivation für dieses Buch, dass einige Unternehmen der Meinung waren, dass Tester in agilen Projekten nicht mehr benötigt würden, weil das Testen nun von den Programmierern und dem Product-Owner erledigt würde. Sie trennten sich sogar von Testern und mussten später schmerzlich zur Kenntnis nehmen, dass sich die Programmierer nicht dem detaillierten funktionalen oder auch nicht-funktionalen Test der Storys und deren Integration auf Applikationsebene widmen konnten oder wollten und auch die Product-Owner und Anwender eher die Validierung der Epics und End-to-End-Tests im Auge hatten als eine Verifikation der Applikation im Sinne eines Systemtests.

Wir, die Autoren, waren schon immer unglücklich mit der dogmatischen Umsetzung neuer Ansätze in der Softwareentwicklung. Im Gegensatz dazu sehen wir die Veränderungen als Chance für einen Prozess der kontinuierlichen Verbesserung und Optimierung.

In den letzten Jahren wurden wir Autoren in agilen Projekten jedoch auch immer wieder damit konfrontiert, dass vieles, was wir als Tester an Selbstverständnis, Methoden, Techniken und Standards erarbeitet und gelernt haben, nicht mehr gelten soll. Das mag auch daran liegen, dass die agile Community in der Vergangenheit vor allem von Softwareentwicklern geprägt war. Diese Tatsache ist auch einer der Gründe, warum die Aufgaben und die Rolle des Softwaretesters in agilen Methoden und Projekten oft nicht oder nur sehr vage definiert sind. Dazu tragen auch unterschiedlich interpretierte Terminologien bei: Wenn z. B. bei Scrum von einem interdisziplinären Entwicklungsteam die Rede ist, dachten oder denken manche, dass das Team nur aus Entwicklern (im Sinne von Programmierern) besteht, die für alles verantwortlich sind, und dass durch testgetriebene Entwicklung mit der Implementierung eines automatisierten Unittest-Sets die Testaufgaben in der Entwicklung ausreichend erfüllt werden. Der Rest liegt in der Verantwortung des Anwenders im User Acceptance Test. Wo sind also die bekannten Testphasen und Teststufen, insbesondere der System- und Systemintegrationstest? Wo und wie finden wir uns als Tester in agilen Projekten wieder? Der agile Ansatz warf und wirft für uns Tester offensichtlich mehr Fragen auf, als er Antworten auf bisherige Probleme lieferte.

Lösungsansätze für diese Probleme wollen wir mit unserem Buch präsentieren, das von Testern für Tester geschrieben wurde. In den einzelnen Kapiteln geben wir Antworten auf die zentralen Fragen, denen wir in unseren Projekten immer wieder begegnet sind. Es geht um generelle und gleichsam kulturelle Veränderungsprozesse, um Fragen der Vorgehensweise und Organisation im Softwaretest, um den Einsatz von Methoden, Techniken und Werkzeugen, insbesondere der Testautomatisierung, und um die neu definierte Rolle des Testers in agilen Projekten. Ein breites Spektrum, das im Rahmen dieses Buchs sicher nicht abschließend und umfassend behandelt werden kann, von dem wir uns aber dennoch erhoffen, es mit Ideen und Anregungen für den Leser abzudecken.

Um die beschriebenen Aspekte noch greifbarer zu machen, werden die einzelnen Themen des Buchs von Erfahrungsberichten aus konkreten Softwareentwicklungsprojekten verschiedener Unternehmen begleitet.

Die Beispiele sollen zeigen, dass unterschiedliche Ansätze zu guten Lösungen führen können, die den spezifischen Herausforderungen agiler Projekte gerecht werden.

In diesem Sinne wünschen wir der Leserin bzw. dem Leser viel Erfolg bei der Umsetzung der hier vorgestellten Inhalte in ihren bzw. seinen eigenen Projekten und laden sie/ihn gleichzeitig ein, uns, die Autoren, auf unserer Internetplattform *www.agile-testing.eu* zu besuchen.

Manfred Baumgartner, Wien 2024

Martin Klonk, St. Pölten 2024

Christian Mastnak, Wien 2024

Richard Seidl, Essen 2024

■ Die Praxisbeispiele im Buch

Das Praxisbeispiel für EMIL in diesem Buch stammt von einem Unternehmen aus der Gesundheitsbranche, das auf 25 Jahre erfolgreiche Produkt- und Softwareentwicklung zurückblicken kann. Mit dem Wachstum des Unternehmens, neuen Kundenanforderungen und strengeren regulatorischen Vorgaben wuchs auch die Notwendigkeit, die Entwicklungs- und Testprozesse zu optimieren und effizienter zu gestalten. Der Gedanke, vom traditionellen auf den agilen Entwicklungsprozess umzusteigen, kam bereits hier und da im Unternehmen auf. Das Softwareentwicklungsprojekt EMIL war der Ausgangspunkt für diese Initiative. Ziel des Projekts war die Neuimplementierung einer Analysesoftware, die seit zehn Jahren weltweit erfolgreich eingesetzt und von verschiedenen Entwicklern entwickelt worden war. Insbesondere aus technischer und architektonischer Sicht konnten die neuen Anforderungen nicht mehr problemlos umgesetzt werden; viele Funktionen waren im Laufe der Jahre als „provisorische Anbauten“ hinzugekommen, aber nie entfernt oder integriert worden. Ein grober Zeitrahmen für die Neuimplementierung aller Funktionen der bestehenden Software wurde auf etwa zweieinhalb Jahre geschätzt. Die größten Herausforderungen auf dem Weg zur agilen Entwicklung waren die mangelnde Erfahrung mit der Zieltechnologie und die regulatorischen Anforderungen, die das Gesundheitswesen mit sich bringt. Die positiven und negativen Erfahrungen, die aufgetretenen Probleme und die Lösungsversuche aus den ersten eineinhalb Jahren des Projekts finden sich in diesem Buch wieder und sind in den entsprechenden Kapiteln entsprechend gekennzeichnet.

Ein weiteres Praxisbeispiel liefert OTTO. Als Online-Händler agiert OTTO in einem sehr agilen Marktumfeld und nutzt innovative Technologien, um auf otto.de und in seinen Filialen ein positives Einkaufserlebnis zu bieten. Als Teil der Otto Group ist OTTO eines der erfolgreichsten E-Commerce-Unternehmen in Europa und Deutschlands größter Online-Händler für Mode und Lifestyle im B2C-Bereich. Über 90 Prozent des Gesamtumsatzes werden online generiert. In den Praxisbeispielen berichtet Frau Diana Kruse von ihren Erfahrungen auf dem Weg von der Testerin und Testmanagerin zum Quality Specialist bzw. Quality Coach, visualisiert durch Grafiken ihres Kollegen Torsten Mangner.

Die Autoren

Manfred Baumgartner

Manfred Baumgartner verfügt über mehr als 30 Jahre Erfahrung in der Softwareentwicklung, insbesondere in der Softwarequalitätssicherung und im Softwaretest. Nach dem Studium der Informatik an der Technischen Universität Wien war er als Softwareentwickler bei einem großen Softwareunternehmen im Bankensektor und später als Quality Director eines CRM-Lösungsanbieters tätig. Seit 2001 hat er die QS-Beratungs- und Schulungsangebote der ANECON, später Nagarro GmbH, eines der führenden Dienstleistungsunternehmen im Bereich Softwaretest, auf- und ausgebaut. Er ist Vorstandsmitglied im Arbeitskreis für Softwarequalität und Fortbildung (ASQF) und Mitglied des Austrian Testing Board (ATB). Seine umfangreichen Erfahrungen sowohl in der klassischen als auch in der agilen Softwareentwicklung bringt er als beliebter Referent auf international renommierten Konferenzen und als Autor und Co-Autor einschlägiger Fachbücher ein: „Der Systemtest – Von den Anforderungen zum Qualitätsnachweis“ (2006, 2008, 2011), „Software in Zahlen“, (2010), „Basiswissen Testautomatisierung“ (2012, 2015, 2021), „Agile Testing – Der agile Weg zur Qualität“ (2013, 2018, 2023).

Martin Klonk

Martin Klonk ist Senior-Testexperte bei Sixsentix Austria GmbH. Ausgebildet als Wirtschaftsingenieur an der Technischen Universität Berlin (und der Université Libre de Bruxelles), begann er seine Karriere 1996 als Softwaretestspezialist bei SQS Software Quality Systems (heute Expleo) in Köln und München. Später wechselte er zu SQS, ANECON und Nagarro Austria in Wien. Martin Klonk hat in den unterschiedlichsten Branchen gearbeitet und war in fast allen Bereichen des Softwaretestens aktiv. Als Präsident des Austrian Testing Board des ISTQB arbeitet er regelmäßig an Lehrplänen und deren deutscher Übersetzung mit und führt auch selbst Schulungen durch. Seit ihm 2007 die Umsetzung erfolgreicher Teststrategien in einem agilen Projekt gelungen ist, ist Martin Klonk ein starker

Verfechter agiler Praktiken im Testen und hat mehrere agile Projekte als Testspezialist geleitet. Er ist zertifizierter Agiler Tester, Projektmanager und Scrum Master.

Christian Mastnak

Christian Mastnak arbeitet als Principal Software Testing Consultant bei Nagarro und verfügt über mehr als 15 Jahre Erfahrung im Bereich QA. Er leitet Nagarros globale „Agile Testing Practice“ und implementiert innovative Lösungen für internationale Kunden in verschiedenen Branchen. Da er in all seinen Projekten einen sehr praxisorientierten Ansatz verfolgt, wechselt er gerne zwischen verschiedenen QA-Rollen – vom Testmanager zum Agile Quality Coach, vom Testautomatisierungsarchitekten zum Testberater. Diese Rollen ermöglichen es ihm, seiner Leidenschaft für die kontinuierliche Verbesserung von QS-Prozessen und -Methoden nachzugehen. Christian Mastnak gibt sein Fachwissen als gefragter Trainer und Redner auf internationalen Konferenzen weiter, darunter EuroSTAR, Agile Testing und die Software Quality Days.

Richard Seidl

Richard Seidl ist Agile Quality Coach und Softwaretestexperte. In seiner abwechslungsreichen beruflichen Laufbahn hat er schon viel Software gesehen und getestet: gute und schlechte, große, kleine, alte und neue. Seine Erfahrungen bündelt er nun zu einem ganzheitlichen Ansatz, denn Entwicklungs- und Testprozesse können nur dann erfolgreich sein, wenn die unterschiedlichsten Kräfte sowie Stärken und Schwächen ausbalanciert sind. So wie ein Ökosystem nur mit allen Aspekten in seiner ganzen Qualität harmonisch existieren kann, müssen die Prozesse im Testumfeld als ein Netzwerk verschiedener Akteure betrachtet werden. Agilität und Qualität wird dann zu einer Haltung, die wir wirklich leben können, anstatt sie nur abzuarbeiten. Als Autor und Co-Autor hat er verschiedene Fachbücher und Artikel veröffentlicht, darunter „Der Systemtest – Von den Anforderungen zum Qualitätsnachweis“ (2006, 2008, 2011), „Der Integrationstest – Von Entwurf und Architektur zur Komponenten- und Systemintegration“ (2012) und „Basiswissen Testautomatisierung“ (2012, 2015, 2021).

Danksagungen

Wir danken den Firmen Nagarro GmbH, GETEMED Medizin- und Informationstechnik AG und Otto (GmbH & Co KG) für ihren Beitrag zur Arbeit an diesem Buch.

Wir danken auch unseren Kolleginnen und Kollegen für deren eifrige Unterstützung und unseren Reviewern, die uns mit kritischen Anmerkungen geerdet und einen wertvollen Beitrag zu diesem Buch geleistet haben:

Sonja Baumgartner (Grafik), Stefan Gwihs, Diana Kruse & Torsten Manger (Praxisbeispiele und Grafik Otto.de), Anett Prochnow, Petra Scherzer, Michael Schlimbach, Silvia Seidl und Harry Sneed. Unser herzlicher Dank gilt auch unseren Lektorinnen Brigitte Bauer-Schiewek, Petra Kienle sowie Kristin Rothe, die unzählige Fehler im Lektorat behoben haben, welche wir übersehen hatten.

Ein besonders großes Dankeschön und eine hohe Wertschätzung gehen an unsere Mitautoren der ersten und zweiten Auflage, Helmut Pichler und Siegfried Tanczos, die uns auch bei der Überarbeitung der vorliegenden Auflage mit ihrer Erfahrung unterstützt haben.

1 Agil – ein kultureller Wandel

Um den kulturellen Wandel hin zur agilen Softwareentwicklung besser zu verstehen und „Agile Testing“ nicht nur als Schlagwort zu verstehen, ist es wichtig, einen Blick in die Vergangenheit zu werfen. Vieles von dem, was wir heute als Allgemeingut wahrnehmen, hat seine Berechtigung im methodischen und technischen Fortschritt der Softwaretechnologie in den letzten 40 Jahren. Erfahrung ist ein wesentliches Element für Innovation und Verbesserung. So lag beispielsweise das Durchschnittsalter der Unterzeichner des Agilen Manifests im Jahr 2001 bei ca. 47 Jahren und damals ging es nicht nur darum, alles anders zu machen, sondern besser. Das wird oft übersehen, wenn „agil“ dazu benutzt wird, unangenehme Dinge einfach loszuwerden oder die eigenen Schwächen zu verbergen. Der ehrliche Ansatz, Software kooperativ, nutzenorientiert und effizient, sprich wirtschaftlich zu entwickeln, ist der Kern der agilen Idee.

1.1 Der Weg zur agilen Entwicklung

Der Übergang zur agilen Entwicklung in der Praxis von IT-Projekten ist seit der Verbreitung der objektorientierten Programmierung in den späten 1980er- und frühen 1990er-Jahren im Gange. Der objektorientierte Ansatz hat die Art und Weise, wie Software entwickelt wird, verändert. Die Hauptziele der Objektorientierung waren

- höhere Produktivität durch Wiederverwendung,
- Verringerung der Codemenge durch Vererbung und Assoziation,
- Erleichterung von Codeänderungen durch kleinere, austauschbare Codebausteine,
- Begrenzung der Auswirkungen von Fehlern durch Kapselung der Codebausteine (Meyer, 1997).

Diese Ziele waren durchaus berechtigt, da die alten prozeduralen Systeme immer größer wurden und aus allen Nähten zu platzen drohten. Die Codemenge drohte ins Unermessliche zu wachsen. Es musste also ein Weg gefunden werden, die Codemenge bei gleicher Funktionalität zu reduzieren. Die Antwort war die Objektorientierung. Neue Programmiersprachen wie C++, C# und Java kamen auf. Die Entwickler begannen, auf die neue Programmiertechnologie umzusteigen (Graham, 1995).

Diese technologische Verbesserung hatte jedoch auch einen Preis – die Zunahme der Komplexität. Durch die Zerlegung des Codes in kleine, wiederverwendbare Bausteine stieg die Zahl der Beziehungen, d. h. der Abhängigkeiten zwischen den Codebausteinen. Bei prozeduraler Software lag die Komplexität in den einzelnen Bausteinen, deren Ablauflogik zunehmend verschachtelt war. Bei objektorientierter Software wurde die Komplexität in die Architektur verlagert. Das machte es schwierig, den Überblick über das Gesamtsystem zu behalten und eine geeignete Architektur im Voraus zu planen. Der Code musste mehrfach überarbeitet werden, bis eine akzeptable Lösung gefunden war. Bis dahin befand sich der Code oft in einem ungeordneten Zustand.

Auf diese Herausforderung gab es zwei Antworten. Die eine war die Modellierung. In den 1990er-Jahren wurden verschiedene Modellierungssprachen vorgeschlagen: OMT, SOMA, OOD usw. Letztendlich hat sich eine von ihnen durchgesetzt: UML Durch die Darstellung der Softwarearchitektur in einem Modell sollte es möglich sein, die optimale Struktur zu finden sowie den Überblick zu gewinnen und zu behalten. Die Entwickler würden – so die Erwartung – das „passende“ Modell erstellen und es dann im konkreten Code umsetzen (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1991).

Das Software-Engineering veränderte sich von der prozeduralen zur objektorientierten Modellierung mit Anwendungsfällen. Die Modellierung war viel detaillierter und wurde durch neue Werkzeuge wie Rational Rose unterstützt (Jacobson, 1992). Die Modellierung erwies sich jedoch als sehr mühsam, selbst mit der besten Werkzeugunterstützung. Der Entwickler benötigte sehr viel Zeit, um das Modell in allen Einzelheiten auszuarbeiten. In der Zwischenzeit hatten sich die Anforderungen geändert und die Annahmen, auf denen das Modell beruhte, waren nicht mehr gültig. Der Modellierer musste bei null anfangen und der Kunde wurde immer ungeduldiger.

Eine andere Antwort auf die Herausforderung der zunehmenden Komplexität war „Extreme Programming“ (Beck, 1999). Da das geeignete Modell für die Software offensichtlich nicht vorhersehbar war, begannen die Entwickler, die Anforderungen in enger Kommunikation mit dem Benutzer und in kurzen Iterationen direkt in den Code zu übersetzen (Beck, 2000).

Dieser Ansatz birgt das Risiko, sich aufgrund ständiger Änderungswünsche in vielen Details in einer Sackgasse zu verlaufen, hat aber den Vorteil, dass der Kunde schnell sieht, was auf ihn zukommt. Wenn die Codebausteine flexibel gestaltet sind, können sie wiederverwendet werden, wenn ein anderer Weg eingeschlagen werden muss. Ein weiterer Vorteil des Extreme Programming war, dass der Benutzer mit auf die Reise genommen werden konnte. Er konnte die Ergebnisse der Programmierung – die realen Benutzeroberflächen, Listen, Nachrichten und Datenbankinhalte – mitverfolgen, was ihm bei der abstrakten Modellierung nicht möglich war. So setzte sich Extreme Programming in der Praxis durch und die Modellierung blieb in der akademischen Ecke (siehe Bild 1.1).

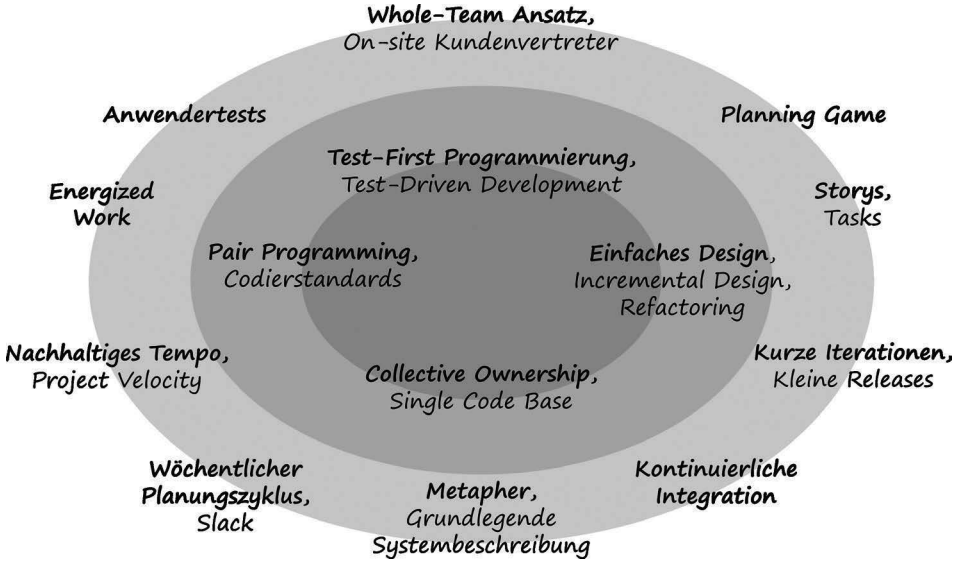

Bild 1.1 XP-Praktiken

Die „Testgetriebene Entwicklung“ (Test-Driven Development, kurz TDD) erwies sich als eine nützliche Konsequenz des Extreme Programming (Beck, 2003). Wenn man unbekanntes Terrain betritt, muss man sich schützen. Der Schutz bei der Codeentwicklung ist der Testrahmen. Die Entwickler bauen zunächst einen Testrahmen und füllen es dann mit kleinen Codebausteinen (Janzen & Kaufmann, 2005). Jede Komponente wird sofort getestet, um zu sehen, ob sie funktioniert. Die Entwickler bahnen sich ihren Weg durch den Code und stellen dessen Status immer wieder durch Tests sicher. Auf diese Weise erreichen sie schließlich einen zufriedenstellenden, getesteten Zwischenstand, den sie dem Benutzer präsentieren können.

In der Softwareentwicklung gibt es nur Zwischenstadien, da Software per Definition nie ganz fertig ist. Die testgetriebene Entwicklung hat sich auch außerhalb von Extreme Programming als ein sehr solider Ansatz erwiesen. Dies wurde auch durch mehrere wissenschaftliche Studien bestätigt und der Einsatz von Unittest-Frameworks und kontinuierliche Integration sind zum Standard in der Softwareentwicklung geworden.

Es versteht sich von selbst, dass Extreme Programming und Test-Driven Development im Widerspruch zu den vorherrschenden Managementmethoden standen. Das Management von Softwareprojekten erforderte eine planbare, prädisponierte Entwicklung, bei der bestimmt werden kann, was, wann und zu welchen Kosten geliefert wird. Systematisches Software-Engineering sollte dies gewährleisten (siehe Bild 1.2).

Die 1990er-Jahre waren auch das Jahrzehnt der Prozessmodelle, des Qualitätsmanagements und des unabhängigen Testens, kurz gesagt, das Jahrzehnt des Software-Engineerings. Software-Engineering sollte durch klar definierte Prozesse mit strikter Arbeitsteilung Ordnung in die Softwareentwicklung und -pflege bringen. Viele Maßnahmen wurden vom Management ergriffen, um die Softwareentwicklung endlich unter Kontrolle zu bringen. Das V-Modell ist repräsentativ für diese Versuche, die Softwareentwicklung zu strukturieren (Höhn

& Höppner, 2008). Leider standen die meisten dieser Maßnahmen in krassem Widerspruch zu der neuen „extremen“ Entwicklungstechnik.

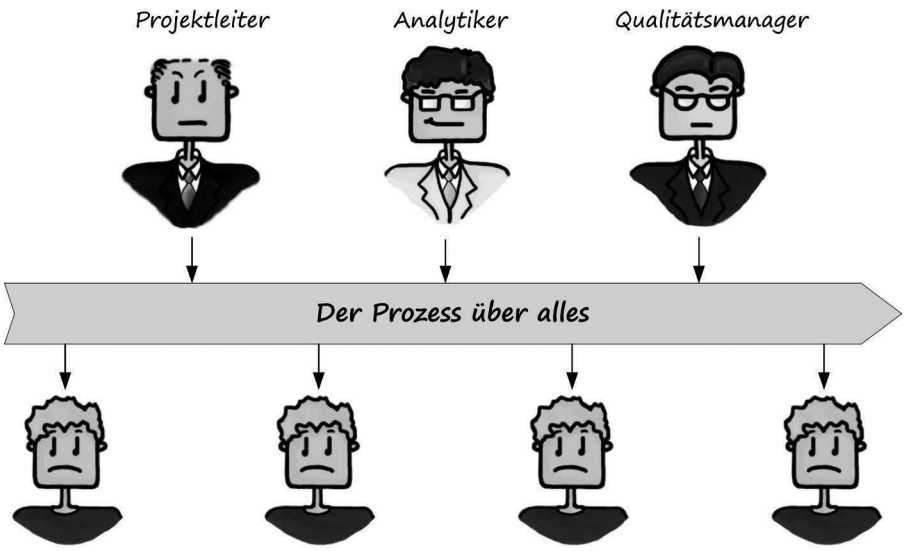

Bild 1.2 Software-Engineering schafft Ordnung in einer chaotischen Softwarewelt

■ 1.2 Gründe für die agile Entwicklung

Eines der wichtigsten Argumente für die agile Entwicklung ist die Nähe zum Benutzer. In der traditionellen, nicht-agilen Entwicklung hatte sich die Kluft zwischen Entwicklern und Benutzern immer weiter vergrößert. In den 1970er-Jahren war diese Kluft noch nicht so groß. Als Harry Sneed, der freundlicherweise das Geleitwort zu diesem Buch verfasst hat, seine Karriere als Entwickler begann, pendelte der Entwickler jeden Tag zwischen seinen Auftraggebern in der Fachabteilung, seinem Schreibtisch und dem Rechenzentrum hin und her. Fast täglich besprach der Entwickler die Aufgabe mit dem Benutzer, schrieb das Programm und probierte es im Rechenzentrum aus, meist am Abend. Die Nähe zum Kunden war das Wichtigste.

Die Arbeitsweise hat sich in den 1980er- und 1990er-Jahren geändert. In der traditionellen Testwelt, die von Gelperin und Hetzel Mitte der 1980er-Jahre geschaffen wurde, herrscht ein grundsätzliches Misstrauen gegenüber dem Entwickler. Man ging davon aus, dass die Entwickler – auf sich allein gestellt – fehlerhafte und qualitativ schlechte Software produzieren würden (Hetzel, 1988). Außerdem würden sie ihre eigenen Fehler nicht erkennen, und wenn doch, würden sie diese als unvermeidliche Eigenschaften der Software deklarieren: „It’s a feature, not a bug.“ Über die Qualität ihrer Architektur und ihres Codes ließen

sie nicht mit sich reden. In ihren Augen sei immer alles in Ordnung: „Bei mir, in der Entwicklung, hat es funktioniert“. Es gäbe keinen Grund, etwas zu verbessern.

Mit diesem Bild des Entwicklers im Hinterkopf wurde die Forderung nach einer eigenen Testorganisation laut. Bei größeren Projekten sollte es eine eigene Testgruppe geben, die mehrere Projekte betreut. In jedem Fall musste die Testgruppe von den Entwicklern unabhängig sein. Dies sei die Voraussetzung dafür, dass die Tester effektiv arbeiten können. Es sollte ein System geschaffen werden, in dem die Tester die Arbeit der Entwickler kontrollieren. Die Entwickler produzieren die Fehler und die Tester finden sie. Ein Fehlerberichts- und -verfolgungssystem sollte die Kommunikation zwischen den beiden Gruppen unterstützen.

Diese Arbeitsteilung zwischen Entwicklern und Testern wurde weltweit propagiert und praktiziert. Neue Begriffe wie „Quality Engineering“ und „Qualitätsmanagement“ wurden geschaffen und jede größere Organisation sollte einen Qualitätsmanager haben. Dies wurde durch die ISO-9000-Normen gefordert. Und wo es Management gibt, gibt es auch Bürokratie. Auf der Grundlage von Normen und Vorschriften wurde eine Bürokratie für die Softwarequalität geschaffen, um die Entwickler anzuleiten, korrekt zu arbeiten (ISO 9000, 2005).

Der Entwicklungsprozess bei der Bertelsmann AG – das Bertelsmann Software-Engineering-Modell – war ein typisches Beispiel dafür. Nach diesem Modell sollte die Fachabteilung zunächst eine vollständige Funktionsbeschreibung des Themas erstellen. Diese wurde von den Abteilungen Qualitätssicherung und Entwicklung akzeptiert und eine Aufwandsabschätzung wurde erstellt (Bender, et al., 1983) (siehe Bild 1.3).

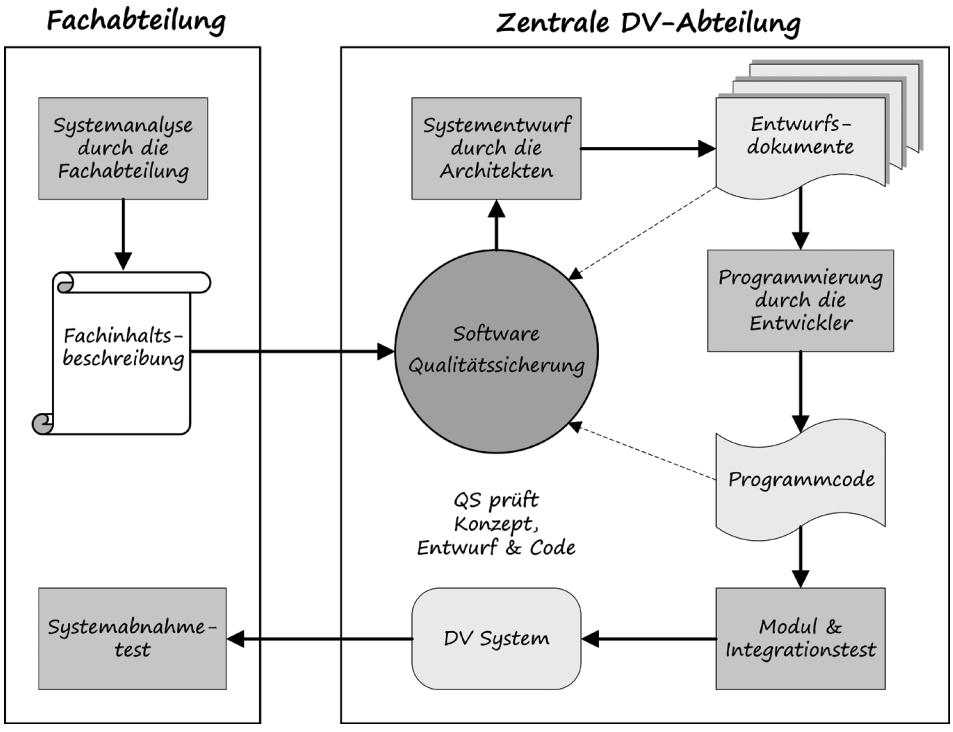

Bild 1.3 Das Software-Engineering-Modell Bertelsmann

Auf der Grundlage dieser Aufwandsschätzung wurde mit der Fachabteilung eine Vereinbarung getroffen: mit einem festen Preis, einem festen Termin und einem festen Ergebnis. Die Anforderungen wurden dann eingefroren und zunächst in einem Systementwurf umgesetzt. Dieser wurde dem Kunden präsentiert, der diesen selten verstand, oder besser gesagt, hätte verstehen können. Meistens nickten die Benutzer nur mit dem Kopf und sagten, es sei in Ordnung. Auf den Systementwurf folgten die Implementierung und die Tests, wobei die Tests immer ein Engpass waren. Das fertige System wurde dem Benutzer viele Monate, manchmal sogar Jahre später präsentiert. Die Reaktion des Benutzers war oft, dass er es so nicht erwartet hätte. Bei Bertelsmann führte dieser gut gemeinte, aber schwerfällige Prozess schließlich zu einer Reorganisation der IT und der Verteilung der Entwickler auf die Abteilungen. Auch andere deutsche Unternehmen hatten das Bertelsmann-Modell übernommen, aber das Ergebnis war meist das gleiche wie bei Bertelsmann: enttäuschte Benutzer. Die Schlussfolgerung ist, dass die Trennung von Entwicklern und Benutzern noch nie gut funktioniert hat.

Ein klassisches Beispiel für einen sehr strukturierten Entwicklungsprozess ist das V-Modell oder das V-Modell-XT (Rausch & Broy, 2006). Dieses Modell wurde in erster Linie für die Softwareentwicklung in deutschen Behörden entwickelt. Es schreibt jeden Schritt des Prozesses vor. Die Anforderungen werden gesammelt und in einem Lastenheft festgelegt. Auf der Grundlage des Lastenhefts wird ein Projekt ausgeschrieben und Angebote werden eingeholt. Das günstigste oder beste Angebot wird ausgewählt und der Gewinner der Ausschreibung erstellt ein Pflichtenheft und legt es dem Auftraggeber vor. Sofern dieser etwas davon versteht, hat er die Möglichkeit, Korrekturen vorzunehmen. Anschließend wird das System implementiert und getestet. Viele Monate später wird das mehr oder weniger getestete Endprodukt an den Auftraggeber zur Abnahme übergeben. Oft stellt sich dann heraus, dass das Produkt in der gelieferten Form nicht oder nicht im erwarteten Umfang genutzt werden kann und der Wartungs- oder Evolutionsprozess beginnt. Es werden Änderungen in und an der Software vorgenommen, bis sie schließlich den Erwartungen des Benutzers entspricht. Dies kann Jahre dauern.

Im Jahr 2009 hat Tom DeMarco buchstäblich das Todesurteil für solche starren, bürokratischen Entwicklungsprozesse geschrieben. Software-Engineering ist ein Ansatz, „whose time has come and gone“ (DeMarco, 2009). Software-Engineering war von Anfang an auf große Projekte wie die des US-Verteidigungsministeriums ausgerichtet, die eine strikte Rollenteilung erforderten. Rückblickend müssen wir uns fragen, ob dieser Ansatz jemals für kleinere Projekte funktioniert hat. Tatsächlich gab es von Anfang an einen gravierenden Fehler: die lange Zeitspanne zwischen der Vergabe des Entwicklungsauftrags und der Auslieferung des Endprodukts. In dieser Zeit hatten sich die Anforderungen und Kundenerwartungen zu stark verändert. Das war schon in den 1980er-Jahren so und ist heute in unserer schnelllebigen Welt noch viel mehr der Fall. Ergo muss die Nähe zum Kunden, dem Auftraggeber, erhalten bleiben und die Entwicklungszyklen müssen verkürzt werden.

Was für die Zusammenarbeit zwischen Entwicklern und Benutzern zutrifft, gilt auch für die Zusammenarbeit zwischen Entwicklern, Testern und dem Betrieb. Auch hier hatte sich die Kluft im Laufe der Zeit vergrößert. Eine Kluft, die heute unter dem Titel DevOps wieder geschlossen wird. Als die Testdisziplin in den späten 1970er-Jahren aufkam, testeten die Entwickler ihre Software meist selbst. Damals war das „Outsourcing“ des Testens ein revolutionäres Ereignis. In den folgenden Jahren ist es zu einer Selbstverständlichkeit geworden. Die Kritik ist jedoch dieselbe wie bei der Softwareentwicklung. Die Zeitspanne zwischen

der Übergabe an die Tester und den Rückmeldungen, sprich, den Fehlermeldungen ist einfach zu lang. Wenn die ersten Fehlermeldungen eintreffen, hat der Entwickler bereits vergessen, wie sie entstanden sind. Das ist der Grund, warum Tester und Entwickler gemeinsam an einem Stück Software arbeiten sollten und warum Tester die fertige Komponente sofort nach ihrer Erstellung testen müssen. Danach kann der Entwickler die Probleme sofort mit dem Tester besprechen und bis zur nächsten Komponentenlieferung beheben. Dieses schnelle Feedback ist das A und O des agilen Testens.

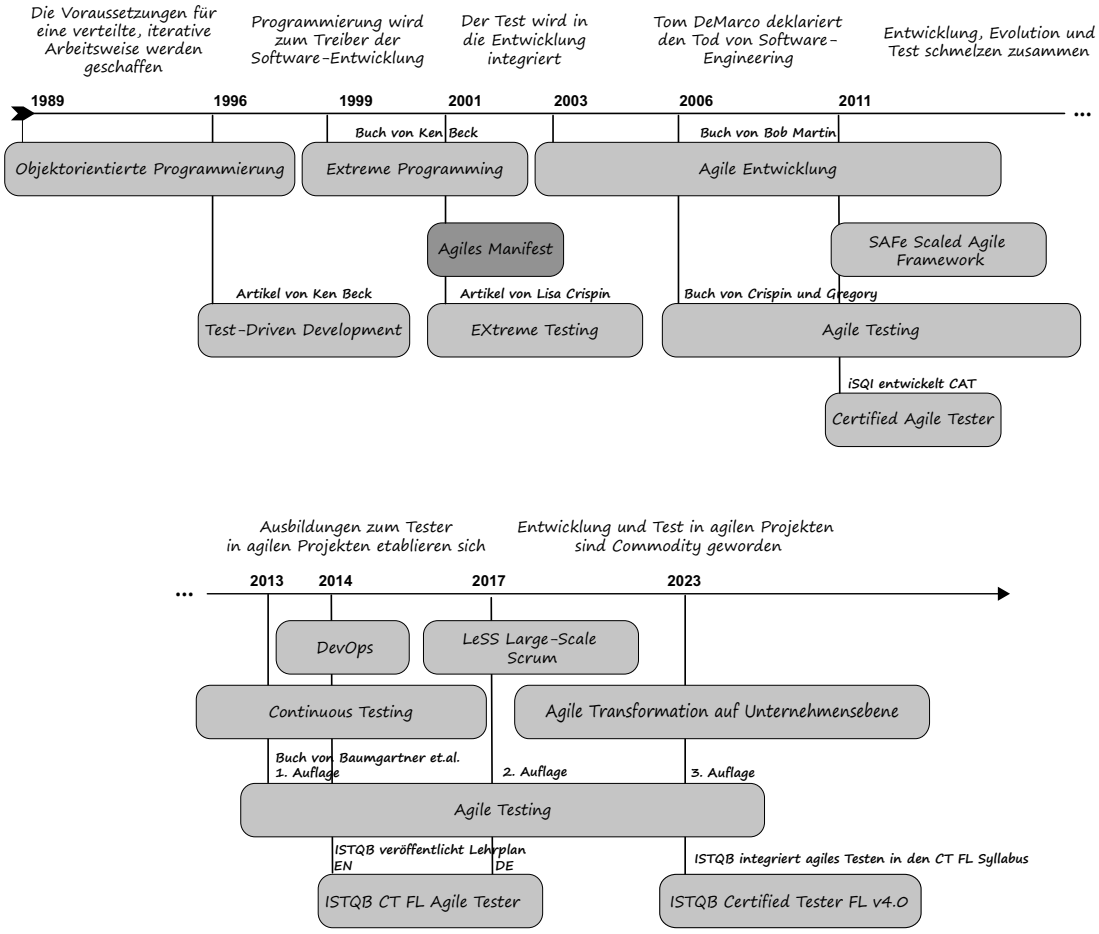

Bild 1.4 Die Geschichte der agilen Entwicklung

1.3 Die Bedeutung des Agilen Manifests für das Testen von Software

Das Agile Manifest wurde im Winter 2001 in einer Skihütte im Bundesstaat Utah verfasst. Der Begriff „Manifest“ deutet bereits auf etwas Revolutionäres hin und war von den Autoren wohl sehr bewusst gewählt. Sie wollten eine Revolution in der Softwarewelt starten, um sich von der Tyrannei der Prozesse zu befreien, und das haben sie auch erreicht. Mit ihrer Revolution ist es ihnen gelungen, diese Welt von Grund auf zu verändern.

Die Beweggründe der Autoren, allesamt herausragende Persönlichkeiten der amerikanischen Programmierszene, waren edel: Sie wollten die Softwarewelt für Benutzer und Entwickler gleichermaßen verbessern. Erreichen wollten sie dies durch die „Zwölf Prinzipien Agiler Softwareentwicklung“:

- Den Kunden durch frühzeitige und kontinuierliche Bereitstellung von wertvoller Software zufriedenstellen.
- Auf sich ändernde Anforderungen eingehen, auch in späten Phasen der Entwicklung. Denn agile Prozesse nutzen Veränderungen als Wettbewerbsvorteil für den Kunden.
- Die häufige Lieferung von funktionierender Software in kurzen Abständen.
- Die tägliche, enge Zusammenarbeit zwischen Benutzern und Entwicklern während des gesamten Projekts.
- Die Gestaltung von Projekten rund um motivierte Personen und Bereitstellung eines Umfelds und der Unterstützung, die sie brauchen, mit dem Vertrauen, dass sie die Aufgabe bewältigen werden.
- Die effizienteste und effektivste Methode des Informationsaustauschs innerhalb eines Entwicklungsteams ist im Gespräch von Angesicht zu Angesicht.
- Funktionierende Software ist der wichtigste Maßstab für den Fortschritt.
- Agile Prozesse fördern eine nachhaltige Entwicklung.
- Ein ständiges Augenmerk auf technische Exzellenz und gutes Design erhöht die Agilität.
- Einfachheit – die Kunst, die Menge an nicht notwendiger Arbeit zu reduzieren – ist wesentlich.
- Die besten Architekturen, Anforderungen und Entwürfe entstehen in selbstorganisierten Teams.
- Das Team reflektiert in regelmäßigen Abständen, wie es effektiver werden kann, und passt sein Verhalten entsprechend an.

In dem Manifest betonen die Autoren die folgenden vier „revolutionären“ Grundsätze , in denen sie

- Individuen und persönliche Interaktion mehr schätzen als Prozesse und Werkzeuge,
- funktionierende Software mehr schätzen als umfassende Dokumentation,
- die Zusammenarbeit mit dem Kunden mehr schätzen als Vertragsverhandlungen,
- die flexible Reaktion auf Veränderungen mehr schätzen als das starre Befolgen eines Plans.

Die Autoren stellen im Manifest auch ausdrücklich fest, dass die Aspekte auf der rechten Seite ebenso ihren Wert haben, die Aspekte auf der linken Seite aber von ihnen höher bewertet werden (Beck, et al., 2001).

Die 17 Autoren des Manifests wollten aus dem Protektorat der Softwarebürokraten ausbrechen. In ihren Augen waren administrative Projektmanager, Qualitätsmanager, Auditoren, Prozessfetischisten und alle, die Softwareprojekte überwachen und behindern, überflüssig. Entwickler sollten von den Fesseln der aus ihrer Sicht unsinnigen Vorschriften, Richtlinien und Standards befreit werden. Sie sollten frei sein, um ihre Arbeit mit dem Benutzer selbst zu gestalten, ohne Kontrolle von außen.

Dies mag aus der Sicht eines Entwicklers sehr verlockend klingen. Entwickler haben sich schon immer über Behinderungen durch Management und Qualitätssicherung beschwert. Sie versuchen immer, ihrem kreativen Drang ungehindert nachzugehen, und hassen nichts mehr als Versuche, sie daran zu hindern. Dieser Konflikt zwischen Kreativität auf der einen Seite und Disziplin auf der anderen Seite war schon immer ein Problem in der Softwareentwicklung. Bei der Konzeption und Entwicklung eines Softwareprodukts ist mehr Kreativität gefragt, bei der Wartung und Weiterentwicklung mehr Disziplin (Sneed, 1976). Die Väter der agilen Entwicklung betonten die Kreativität. Das Jahr der Veröffentlichung des Manifests – 2001 – folgte auf ein Jahrzehnt der Versuche, durch Phasenkonzepte, Vorgehensmodelle, Qualitätsrichtlinien, Prozessideologien, unabhängige Tests und eine Vielzahl anderer Regulierungs- und Standardisierungsmaßnahmen Ordnung in Entwicklungsprojekte zu bringen. Bei den Entwicklern selbst waren diese Maßnahmen von Anfang an unbeliebt und sie leisteten oft passiven Widerstand gegen das, was sie als einschränkendes System empfanden. Mit dem agilen Manifest kündigten sie ihre Revolution an.

Es versteht sich von selbst, dass sich diese neue Bewegung vor allem gegen die bisherigen Managementmethoden richtete. Dazu gehören die Qualitätssicherung durch außerhalb des Projekts stehende Technokraten und die Trennung von Entwicklung und Test. Nach der ursprünglichen Ideologie der agilen Entwicklung sollten sich die Entwickler selbst um die Qualität ihrer Software kümmern und die Tester sollten sie, wenn überhaupt, nur unterstützen. Leider hat sich diese Haltung in vielen agilen Projekten lange durchgesetzt. Es hat ein Machtwechsel stattgefunden. Es sollte nicht überraschen, dass das Wort „Manager“ mittlerweile ein Unwort ist. Es gibt den Benutzervertreter, den Product-Owner und den Scrum-Master, aber keine Manager – weder Projekt- noch Produktmanager. In einem agilen Projekt managt sich das Team selbst (Schwaber, 2007).

Für Tester und Qualitätssicherungsexperten bedeutete diese Revolution zunächst auch einen Verlust an Rollen und Macht. Die Aufgaben und die Rolle des Testers in agilen Methoden und Projekten sind oft nicht oder nicht klar definiert. Aber bereits Martin Fowler hatte in seinem Essay „The New Methodology“ darauf hingewiesen, dass neben dem Entwickler viele weitere Personen am Softwareentwicklungsprozess beteiligt sind, darunter auch Tester (Fowler, 2000). Diese sind ebenfalls Teil des interdisziplinären Teams, wie Entwickler, Architekten, Requirements Engineers, etc. Alle Beteiligten müssen jedoch interdisziplinär denken und zunehmend Tätigkeiten übernehmen, die über ihre zentralen Aufgaben hinausgehen. Dieses Umdenken ist auch eine Herausforderung für die Tester. In agilen Teams kann der Entwickler manchmal Testaufgaben übernehmen und der Tester wird – je nach Anforderungen und Skills – auch Entwickleraufgaben übernehmen. Es wäre jedoch ein Fehler, anzunehmen, dass dies so einfach ist: Weder kann ein ausgebildeter Entwickler per Knopfdruck ein guter Tester werden, noch sind Tester automatisch mit entsprechender Pro-

grammiererfahrung ausgestattet. Es wäre auch fatal, wenn nicht alle Disziplinen des Software-Engineerings im Team in Exzellenz vertreten wären. Agile Methoden erfordern ein Höchstmaß an Kompetenz und Erfahrung:

- Requirements Engineers, die die Anforderungen der Benutzer klar verstehen und gesamthaft erfassen können,
- erfahrene Architekten, die eine Architektur entwerfen, die nicht durch ständige Änderungen ins Chaos führt,
- Entwickler, die ihren Code so schreiben, dass für ihn dasselbe gilt wie für die Architektur, und die konsequent Unittests definieren, die mehr tun, als nur die Existenz von Klassen und deren Methoden zu prüfen,
- professionelle Tester, die ihre Testansätze und Testmethoden an die spezifischen und sich ständig ändernden Aufgabenstellungen anpassen und den Automatisierungsgrad im System- und kontinuierlichen Integrationstest während der Entwicklung ständig erhöhen.

1.4 Agiles Arbeiten erfordert einen kulturellen Wandel bei den Benutzern

Die durch das Agile Manifest ausgelöste Revolution ist nicht auf Softwareentwicklungsprojekte beschränkt. Sie wirkt sich auch auf die Benutzerorganisationen aus, in denen die Projekte durchgeführt werden. Diese Auswirkungen waren zu erwarten. Wenn sich die Art und Weise, wie Projekte durchgeführt werden, fundamental ändert, müssen sich auch die Bedingungen, unter denen die Projekte stattfinden, ändern. In diesem Fall können die Projektabläufe nicht mehr im Voraus festgelegt werden. Jedes Projekt muss – wie das Wasser – seinen eigenen Weg zum Ziel finden. Selbst das Ziel kann sich im Laufe des Projekts ändern. Ein agiles Projekt kann mit einer Expedition in eine fremde Welt ohne Landkarten verglichen werden. Das Expeditionsteam muss seinen eigenen Weg erforschen.

Traditionelle Methoden der Projektplanung und -steuerung sind in einer agilen Welt überholt (Mainusch, 2012). Zu Beginn eines agilen Projekts kann niemand vorhersagen, wie viel es kosten wird oder bis wann das Ziel erreicht ist. Man kann ein Zeit-/Aufwands- und Kostenlimit festlegen, aber nicht, was bis dahin umgesetzt sein soll. Die zur Verfügung stehende Zeit und die Kosten bestimmen den Umfang an Funktionalität und die Qualität, die das Projekt mit seiner Produktivität liefern kann. Was inhaltlich umgesetzt wird, ergibt sich erst im Laufe des Projekts entlang der sich laufend ändernden Anforderungen und Prioritäten

Nach dem traditionellen Ansatz werden Funktionalität und Qualität festgelegt und der Projektleiter muss abschätzen, wie viel Zeit und Aufwand erforderlich sind, um das vorgegebene Ziel zu erreichen. Auf der Grundlage seiner Kalkulation wird eine Vereinbarung mit dem Kunden getroffen und diese Vereinbarung bleibt verbindlich. Auf Basis dieser Berechnungen werden auch Verträge zwischen Auftraggeber und Auftragnehmer geschlossen, die manchmal Vertragsstrafen für den Auftragnehmer vorsehen, wenn dieser die Vereinbarung nicht einhält. Wird die geforderte Funktionalität innerhalb der vereinbarten Zeit mit den

vereinbarten Mitteln nicht erreicht, werden zuerst Abstriche bei der Qualität vorgenommen, und wenn dies nicht ausreicht, wird die nicht erfüllte Funktionalität in die sogenannte Wartungsphase verschoben. Laut den Standish Group's regelmäßigen Chaos Reports erreichen nur sehr wenige IT-Projekte ihre spezifizierte Funktionalität innerhalb des geplanten Zeit- und Arbeitsaufwands (Standish Group, 2020). Dennoch haben die IT-Verantwortlichen die Illusion, dass sie ihre IT-Projekte planen können. Das Agile Manifest räumt mit dieser Vorstellung auf. Die Zusammenarbeit zwischen Auftraggeber und Auftragnehmer hat Vorrang vor Verträgen und festen Vereinbarungen. Gemäß dem Agilen Manifest sollten sie gemeinsam ihre Ziele erkunden und darauf hinarbeiten und es steht ihnen frei, diese Ziele jederzeit zu ändern und dabei neue Erkenntnisse zu berücksichtigen. Auf diese Weise werden die Ziele kontinuierlich an das Erreichbare angepasst.

In dieser Hinsicht hat die agile Bewegung durchaus einen Einfluss auf die Organisation. Das Management kann keine festen Ziele mit festen Kosten und einem festen Termin mehr setzen. Das Management selbst muss flexibel sein. Die Ziele werden nicht mehr im Sinne eines Pflichtenhefts formuliert, sondern als Wert/Nutzen, der für das Unternehmen geschaffen wird. Die zu erreichende Funktionalität und Qualität werden dem agilen Entwicklungsteam überlassen, das auch die Benutzer einbezieht. Das Management kann bestenfalls ein Zeit- und Kostenlimit setzen, das aber auch angepasst werden kann, wenn es wirtschaftlich sinnvoll ist. Das Management hat nur noch eine richtungsweisende Funktion. Es gibt keine Weisungen mehr, wie Projekte durchgeführt werden sollen. Sein Einfluss darauf ist durch den agilen Ansatz begrenzt (Gloger, 2013).

Was für das Management gilt, gilt auch für das Qualitätsmanagement. Die Qualitätssicherungsabteilung war unter der Leitung des Qualitätsmanagers für die Sicherstellung der Qualität der von den Projekten gelieferten Software verantwortlich. Dies hat zur Trennung von Test und Entwicklung geführt. Der duale Ansatz mit einer Entwicklungsschiene und einer parallelen Testschiene wurde viele Jahre propagiert.

Mit der Einführung der Agilen Entwicklung wurden zentrale Abteilungen für Qualitätssicherung und Qualitätsmanagement in Frage gestellt oder schlichtweg nicht mehr benötigt. Ein großer Teil der Qualitätsverantwortung wurde auf die agilen Teams und auf die Mitarbeiter der beteiligten Fachabteilungen übertragen. Dies gilt sowohl für die Definition von Qualitätsanforderungen und deren Überprüfung als auch für den Ansatz, wie diese erreicht werden sollen.

Die Tester der zentralen Testteams arbeiten direkt in den agilen Projektteams. Die Rolle eines Testmanagers gibt es allerdings nicht mehr. Die bisherige Test- oder Qualitätssicherungsabteilung wird zu einem Ressourcenpool und einer Unterstützungs- und Coachingorganisation für die verschiedenen agilen Projekte in einem Unternehmen (Golze, 2008). So gesehen sind die Qualitätsmanager und Testmanager die großen Verlierer dieses Umschwungs. Es sei denn, sie verstehen es, ihre Rolle neu zu definieren.

In jeder Revolution gibt es Gewinner und Verlierer. Die anderen Verlierer sind die Requirements Engineers, die bisher die Anforderungsspezifikationen geschrieben haben. Sie werden nicht mehr gebraucht, zumindest nicht in der Rolle eines Bindeglieds zwischen Benutzer und Entwickler. In der agilen Welt können sie jedoch als Benutzervertreter auftreten und Storys formulieren. Dazu benötigen sie aber ein viel tieferes Fachwissen, als es die meisten Requirements Engineers in der Vergangenheit hatten. Um als echte Vertreter der Benutzer zu agieren, müssen sie deren Sichtweise übernehmen und über den gleichen Wissensstand verfügen.

Die eindeutigen Gewinner der agilen Revolution sind die Entwickler und potenziell auch die Endbenutzer, wenn sie die Chance ergreifen und sich aktiv an der Gestaltung künftiger Anwendungssysteme beteiligen, insbesondere in der Rolle eines Product-Owners (siehe Bild 1.5).

Bild 1.5
Der Product-Owner als zentrale Rolle in agilen Projekten

■ 1.5 Konsequenzen der agilen Entwicklung für die Softwarequalitätssicherung

Das Aufkommen der agilen Softwareentwicklung hat viele Auswirkungen auch auf das Testen von Software, zum Beispiel räumliche und zeitliche.

1.5.1 Räumliche Auswirkungen

Die Tester waren meist von den Entwicklern räumlich getrennt. In der Vergangenheit arbeiteten sie auf einer separaten Etage des Bürohauses oder in einem anderen Gebäude und besuchten die Entwickler von Zeit zu Zeit. Dies war eine Folge der Philosophie, dass die Qualitätssicherung unabhängig sein muss, um effektiv zu sein. Die Qualitätssicherung hatte sogar das Recht, eine Freigabe zu verschieben oder zu stoppen, oder die Verantwortung, sie freizugeben. Das endete oft in Grabenkämpfen, bei denen der Leiter der Qualitätssicherungsgruppe und der Leiter der Entwicklungsabteilung aneinandergerieten. Die Entwickler wollten ihre Software so früh wie möglich freigeben und die Qualitätssicherer waren der Meinung, dass die Software noch nicht ausgereift genug war. Die Entscheidungen wurden oft an die Geschäftsleitung weitergeleitet. Der Konflikt war eingearbeitet und auch nach dem Prinzip der „Checks and Balances“ gewollt (Evans, 1984).

Für die Analyse der Anforderungsdokumente und Entwürfe sowie für die Inspektion des Codes erhielt die Qualitätssicherung die relevanten Dokumente auf dem Dienstweg und hatte eine gewisse Zeit, sie zu prüfen. Die Prüfer verfassten ihre Berichte und übergaben diese an die Entwicklungsabteilung. Anschließend trafen sie sich, um die Ergebnisse der

Prüfung zu besprechen. Für den Test der Software musste die Entwicklungsabteilung ihre kompilierten und unit-getesteten Komponenten an eine Testbibliothek liefern, von wo sie von der Qualitätssicherungsabteilung für Integrations- und Systemtests übernommen wurden. Die Tester führten ihre vorbereiteten Testläufe durch und meldeten die gefundenen Fehler an die Entwickler. Die Entwickler behoben die Fehler und gaben die Komponenten wieder zurück. Dies wiederholte sich so lange, bis die Qualitätssicherungsabteilung entschied, dass die Software ausreichend getestet worden war, oder bis das Management beschloss, die Software trotz Qualitätsmängeln freizugeben. (Sneed, 1983).

Durch die Trennung zwischen den Entwicklern und den Testern entwickelte sich eine typische „Wir und sie"-Mentalität. Die Entwickler betrachteten die Tester als übermäßig pedantische Querulanten, während die Tester die Entwickler als unfähig ansahen, ihre Arbeit richtig zu machen, und als Mitarbeiter, die sie zu erziehen hatten. Dieses Rollenverständnis in Verbindung mit der räumlichen Trennung ging oft zu Lasten des Projekts. Anstatt sich auf den Inhalt zu konzentrieren, verstrickten sich die „natürlichen Feinde" in unnötige Streitigkeiten über Formalitäten.

Die Väter der agilen Entwicklung gingen davon aus, dass alles besser wird, wenn die organisatorischen und räumlichen Mauern fallen. Die physische Nähe und das gemeinsame Ziel entschärfen die unvermeidlichen Konflikte (Gloger & Häusling, 2011). Dieser Aspekt muss heute berücksichtigt werden, wenn man über Entwicklungs- und Testaktivitäten in verschiedenen Outsourcing Strategien oder in agilen, weltweit verteilten Teams nachdenkt. Die Kommunikationsmöglichkeiten haben sich seit der Veröffentlichung der agilen Prinzipien dramatisch verbessert, aber dennoch sind „tägliche, enge Zusammenarbeit zwischen Benutzern und Entwicklern während des gesamten Projekts" oder „die effizienteste und effektivste Methode des Informationsaustauschs innerhalb eines Entwicklungsteams ist im Gespräch von Angesicht zu Angesicht" nicht automatisch durch die Installation iner Videokommunikationssoftware erfüllt.

1.5.2 Zeitliche Folgen

In zeitlicher Hinsicht hat die agile Entwicklung weitere Konsequenzen für die Qualitätssicherung. Die Zeit, in der sie die Dokumente prüfen und das System testen konnte, gibt es nicht mehr. Traditionell verbrachten Qualitätssicherer mehrere Tage damit, ein Anforderungsdokument oder einen Systementwurf zu prüfen und zu bewerten. Wenn sie nun User-Storys überhaupt prüfen, dann nur an dem Tag, an dem sie ihnen mitgeteilt werden (wie z. B. bei Scrum in der Sprintplanung). Ansonsten ist es zu spät: Die Story wird sofort umgesetzt.

Bei der traditionellen Qualitätssicherung dauerte der Testprozess mehrere Wochen, wenn nicht gar Monate. Das System blieb in der Testphase, bis die meisten Fehler behoben waren – und das konnte, je nach Größe des Systems, sehr lange dauern. Der Entwickler musste warten, bis die Fehler gemeldet wurden, die Tester wiederum mussten warten, bis die Fehler behoben waren, und der Benutzer musste lange warten, bis er das System endlich zu Gesicht bekam.

In der agilen Entwicklung gibt es keine Wochen oder Monate, um ein System zu testen. Das System wird Stück für Stück aufgebaut und jedes Teil wird innerhalb weniger Tage getestet.

Wenn ein Release-Zyklus maximal vier Wochen dauern soll, bleiben maximal ein paar Tage für die abschließenden Tests eines Release. Die Technik der „Continuous Integration“ bietet die Möglichkeit und Notwendigkeit des „Continuous Testing“ (Duvall, Glover, & Matyas, 2007). Jede Komponente wird getestet, sobald sie entwickelt ist. Die Tests beginnen an dem Tag, an dem die erste Komponente übergeben wird.

Dies ist eine gewaltige Veränderung gegenüber der traditionellen Arbeitsweise. Bisher hatten die Tester Monate Zeit, einen Testplan zu entwickeln, Testfälle zu spezifizieren und Testskripte zu schreiben. In der agilen Entwicklung ist die Vorbereitungszeit auf wenige Tage geschrumpft – die kurze Zeit, bis die erste Komponente geliefert wird. Folglich müssen die Tester lernen, parallel zu planen und auszuführen. Während sie eine Komponente testen, planen sie den Test der nächsten Komponente. Sie müssen mehrere Aufgaben gleichzeitig bewältigen.

Niemand kann behaupten, dass die agile Entwicklung den Testern das Leben leichter machen würde. Sie haben nur wenig Zeit, um ihre Aufgaben zu erledigen, und müssen ständig überlegen, welche Aufgabe sie als Nächstes vorziehen. Es gibt keinen Testmanager, der die Testarbeit für sie plant und zuweist. Die Tester müssen sich selbst verwalten und ihre Zeit selbst einteilen. Das mag für einige Tester eine große Herausforderung sein, aber sie müssen die Herausforderung annehmen, um mit dem Team mithalten zu können. Der agile Test ist auf schnelles Reagieren ausgelegt. Die Entwickler gehen in eine bestimmte Richtung und die Tester müssen dieser folgen, der Schwerpunkt liegt auf dem Hier und Jetzt.

Zeit ist der bestimmende Faktor in der agilen Entwicklung. Die Tester müssen daher sicherstellen, dass die Qualität unter den gegebenen zeitlichen Einschränkungen so gut wie möglich ist. Die verkürzte Zeit verändert die Arbeitsbedingungen und geht oft auf Kosten der Qualität. Die Folge sind technische Schulden, auf die wir später noch zu sprechen kommen. Es ist vorerst wichtiger, das richtige Produkt unvollständig zu bauen, als das falsche Produkt richtig. Dies wird jedoch letztlich vom Benutzer beurteilt. Dafür gibt es Folge-Releases, bei denen die Qualität nachgebessert werden kann. Die Wartung findet im agilen Entwicklungsteam statt. Das Wichtigste ist, dass der Benutzer so schnell wie möglich ein vernünftig funktionierendes Produkt erhält. (Martin, 2002)

Projekt EMIL: Kultureller Wandel – was Agilität bedeutet

Die Umstellung eines über 20 Jahre „historisch gewachsenen" Entwicklungs- und Testprozesses wird nicht von heute auf morgen funktionieren. Viele Informationsflüsse, Prozesse und Methoden haben sich eingebürgert und sind nicht einfach zu ändern. Innerhalb eines Projekts ist der Übergang zu einem agilen Ansatz noch steuerbar. Das Wichtigste ist die Akzeptanz oder zumindest die Toleranz des Managements. Schwieriger ist der Veränderungsprozess ab den Schnittstellen zwischen dem Projekt und dem Rest des Unternehmens.

Das Projekt EMIL war nicht dazu gedacht, alle Prozesse im Unternehmen zu verändern, sondern ein Pilotprojekt zu starten. Damit sich das Projektteam voll und ganz auf seine Arbeit konzentrieren konnte, wurden die Schnittstellen zu den anderen Abteilungen durch den Scrum-Master und den Product-Owner bearbeitet. Der Product-Owner übertrug beispielsweise den Projektfortschritt in einen Meilensteinplan, um eine externe Gruppe von Stakeholdern mit Informationen zu versorgen, die sie für ihre Arbeit benötigen. Das Projektteam war damit nicht belastet. Darüber hinaus wurden klassische Projektstatusberichte verfasst und dem Management zur Verfügung gestellt. Auch hier war das Projektteam nicht involviert.

Gleichzeitig präsentierten der Scrum-Master und der Product-Owner die Erkenntnisse und Erfahrungen im Unternehmen, um mit Vorurteilen und Mythen des agilen Vorgehens aufzuräumen. Einer dieser Mythen ist die Entwicklungsgeschwindigkeit. Manche Manager erwarteten beispielsweise schnellere Entwicklungen durch den agilen Ansatz. Das erfüllte sich aber nicht. Ein Grund mehr für die Teammitglieder, darauf hinzuweisen, was das Projekt als „agil" definiert hatte und was für Anwendungsentwicklungen im Gesundheitswesen wesentlich ist:

Agil bedeutet nicht „schneller", agil bedeutet „höhere Qualität".

■

Index

X

Z