edv-buchversand.de

Agile Testing

Der agile Weg zur Qualität

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

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).

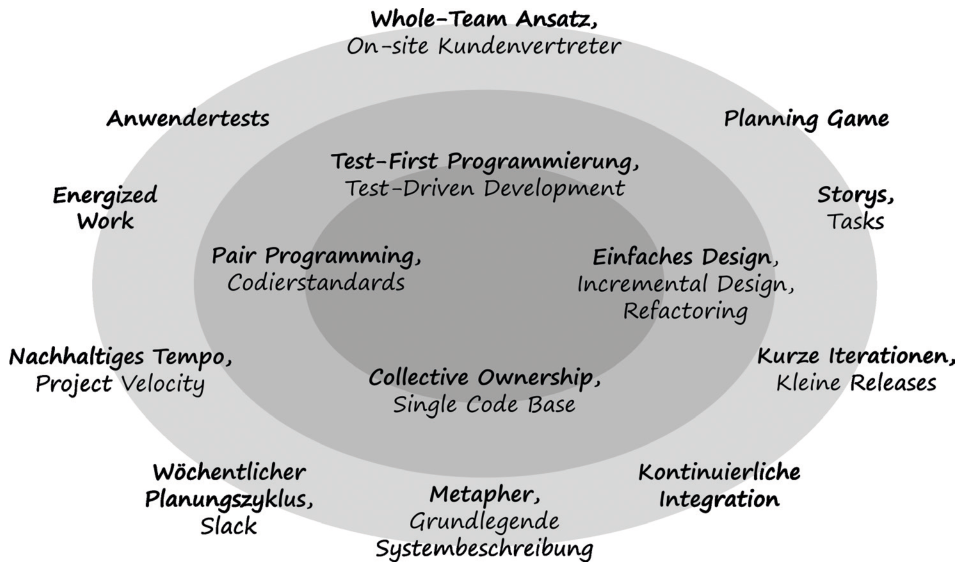

Bild 1.1 XP-Praktiken

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.

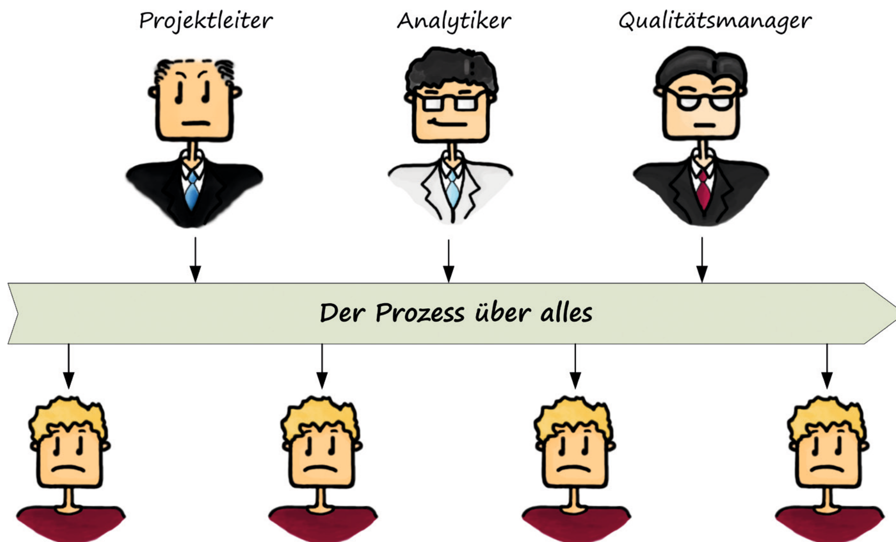

Bild 1.2 Software-Engineering schafft Ordnung in einer chaotischen Softwarewelt

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).

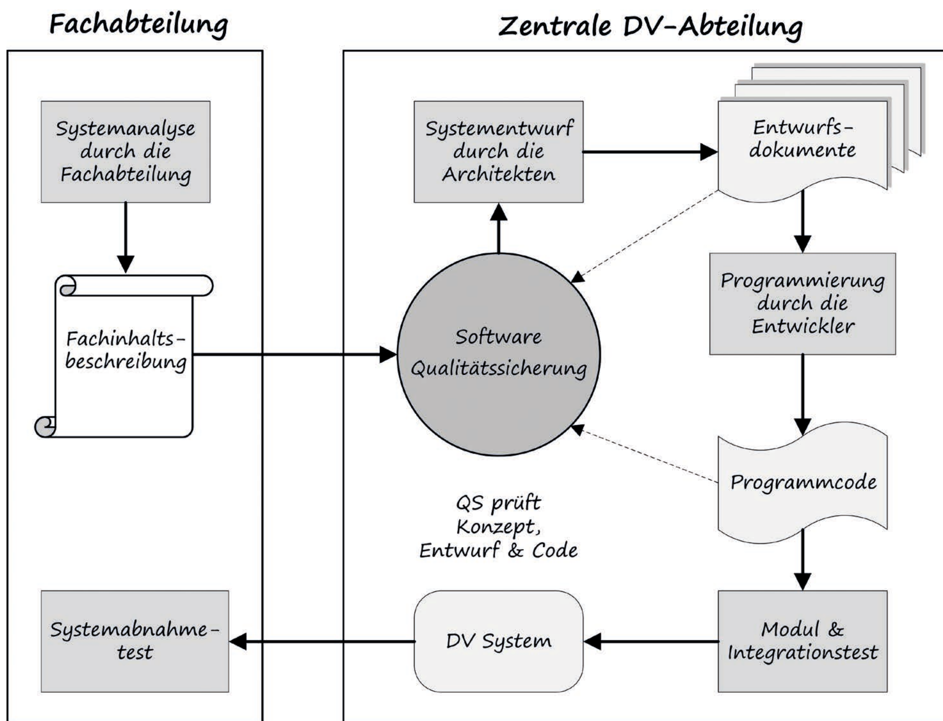

Bild 1.3 Das Software-Engineering-Modell Bertelsmann

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.

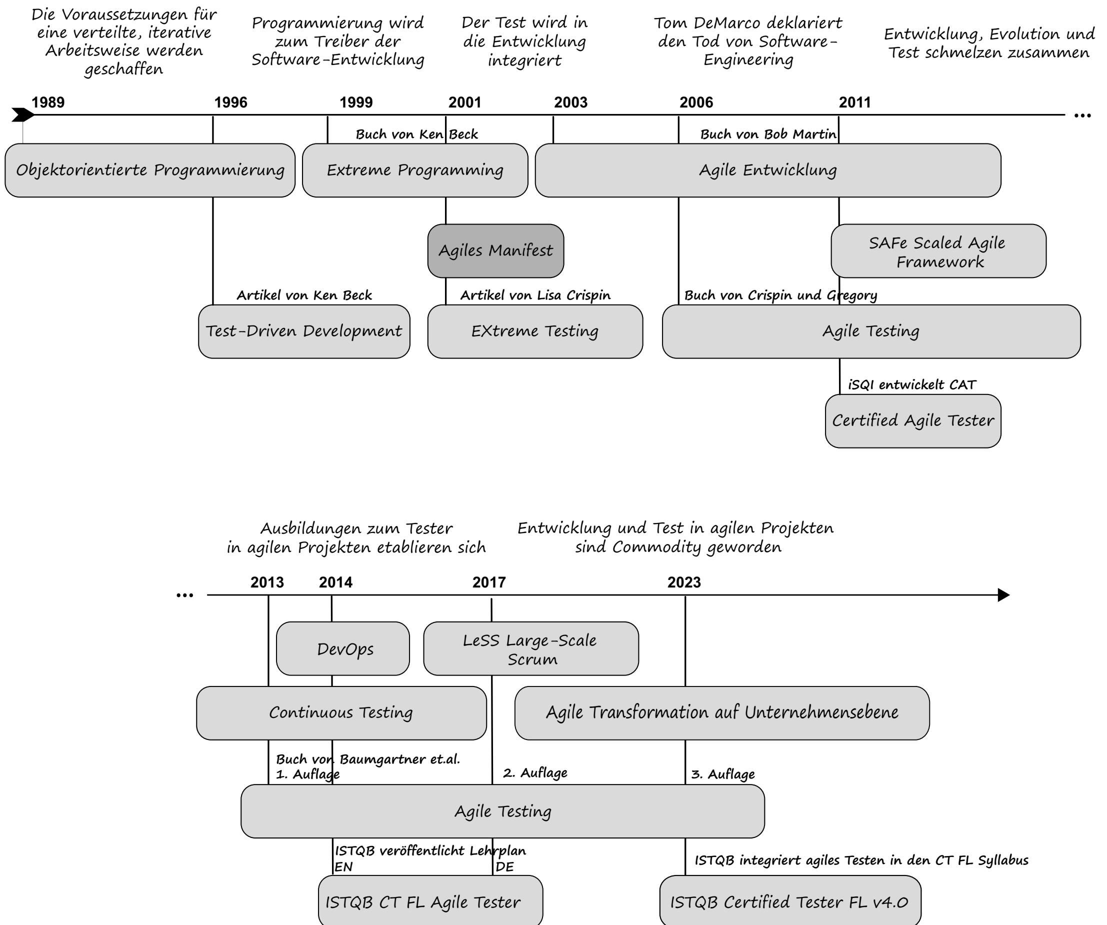

Bild 1.4 Die Geschichte der agilen Entwicklung

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“.