Agiles Testmanagement

Harry M. Sneed
ANECON, GmbH, Wien

1 Einleitung

Unter dem Begriff Management verstehen wir die Planung, Organisation, Überwachung und Steuerung einer Aktivität. Ein Projektmanager ist demnach jemand, der ein Projekt plant, organisiert, überwacht und steuert. Dazu braucht er wissen, Erfahrung, Information und Kompetenz. Um das Management eines agilen Tests zu verstehen müssen wir uns erst mit dem Management agiler Projekte auseinander setzen. Dazu ist schon vieles geschrieben worden. Es gibt etliche Ansätze, u.a. Scrum, Kanban, Extreme Programming und Lean Management, die darauf zielen, ein agiles Projekt einschließlich der Aktivität Testen zu planen, zu organisieren und zu steuern. Die Frage stellt sich, ob es überhaupt möglich ist das Management der Testaktivitäten getrennt zu betrachten. In konventionellen Projekten war dies möglich, weil das Testen ein eigenes Subprojekt bildete. Das Testprojekt lief mehr oder weniger parallel zum Entwicklungsprojekt [Blac99].

In einem agilen Projekt ist dies nicht mehr der Fall. Der Test ist verwoben mit der Entwicklung. Es mag zwar Tester geben, also Spezialisten mit dem Schwerpunkt Qualitätssicherung und Akzeptanztest, aber eine separate Testschiene soll es nicht geben. Der Tester im Team hat daher eine duale Rolle, zum Einen als Tester und zum Zweiten als Testmanger. In dieser Rolle antwortet er gegenüber dem Produkt-Owner, bzw. dem Benutzervertreter. Er managt praktisch sich selbst. Falls es mehrere Tester gibt, können sie diese Aufgaben untereinander aufteilen [Beck99]. (siehe Abbildung 1)

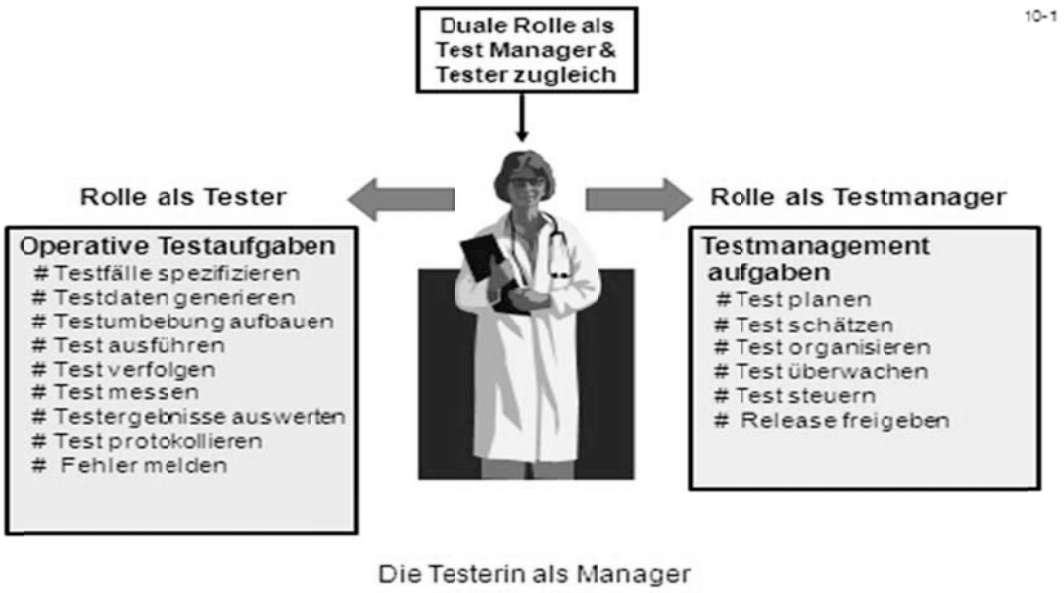

Abbildung 1: Rolle des Testers/der Testerin in agilen Projekten

Da Management ein Sammelbegriff ist für unterschiedliche Aktivitäten, müssen wir, um es zu beschreiben, uns mit den einzelnen Aktivitäten auseinander setzen. Diese sind

- Testplanung
- Testschätzung
- Testorganisation
- Testüberwachung und
- Teststeuerung.

2 Agile Testplanung

Bei den konventionellen Software Entwicklungsprojekten wurde der Test als eigenständige Aktivität geplant. Der Testplan bzw. die Prüfspezifikation galt als Pflichtlieferung. Es gibt sogar Konventionen, z.B. die IEEE Standard 829, wie der Testplan auszusehen hat. In dem Testplan werden die Testziele definiert, die Testaufgaben beschrieben, die Testergebnisse festgelegt und die Testressourcen bestellt [IEEE829]. Es werden auch Termine gesetzt und Aufgaben zugeteilt. Schließlich werden für das ganze Projekt Aufwände geschätzt und Kosten ermittelt.

In einem agilen Entwicklungsprojekt ist dieser Gesamtheitsansatz zum Testen nicht möglich. Da die Anforderungen Stück für Stück während des Projektes erst ermittelt werden, können wir nicht wissen, was wir testen sollten. Was wir aber wissen, ist wie wir testen sollten. Die groben Testaufgaben werden für jedes Release dieselben sein, d.h. wir können die Aktionen ohne deren Gegenstand planen. Die Gegenstände ergeben sich für jedes neue Release aus den letzten User Stories. Wir wissen, was für Ergebnistypen wir liefern wollen, auch wenn wir die einzelnen Ergebnisse noch nicht kennen. Das Gleiche trifft für die Ressourcen zu. Die Ressourcentypen sind vorsehbar, die konkreten Ressourcenausprägungen jedoch nicht. Diese werden für jedes Release ermittelt. Für ein agiles Projekt machen wir also nicht einen endgültigen Testplan, sondern nur einen Plan des Planes – eine Schablone, die für jedes Release neu ausgefüllt wird. Wir können auch nicht zu Beginn des Projektes eine endgültige Schätzung der Testkosten abgeben. Wir können allenfalls aufgrund der Erfahrung aus ähnlichen Projekten einen Kostenrahmen anstecken. Darin werden eine Ober- und eine Untergrenze an Testaufwand zunächst provisorisch festgelegt und später nach jedem Release korrigiert [Seib12]. (siehe Abbildung 2)

Die Kunst der agilen Testplanung liegt darin, mit einem Minimum an Information, trotzdem Planziele zu setzen. Wir planen z.B.

- den Unit-Test der Entwickler abzunehmen
- den Source-Code zu bewerten
- die Abnahmetestfälle zu spezifizieren und
- die Testergebnisse zu kontrollieren

und diese wiederholen wir für jede neue Iteration. D.h. wir dürfen nur das planen, was wir in zwei bis sechs Wochen erreichen können. Wir wissen ebenfalls, welche Tester zur Verfügung stehen, auch wenn wir nicht genau wissen, was sie testen werden. Wir wissen mindestens, was für Testtypen sie durchführen werden. Also plant man das ganze Projekt nur in groben Zügen mit der wenigen Information, die man hat. Dafür plant man den Test der Iterationen recht detailliert mit Zielen, Aufgaben, Ergebnissen, Ressourcen und Aufwände. Die Abnahmekriterien bilden einen Anhang zum Plan. Der Iterationstestplan wird in einem

vorgefertigten Musterdokument erfasst und dem Projektteam zur Diskussion vorgelegt. Das Team kann Korrekturen verlangen, sodass der Tester bereit sein muss, den einen oder anderen Punkt zu überarbeiten. Es ist wichtig, dass das Team als Ganzes hinter dem Iterationstestplan steht. Wenn es sein muss, wird der Plan einige Male nachgebessert, aber bis Ende der ersten Woche muss er endgültig stehen. Der Iterationstestplan ist die Basis für den Iterationstest. Die beigelegten Abnahmekriterien dienen als Testorakel.

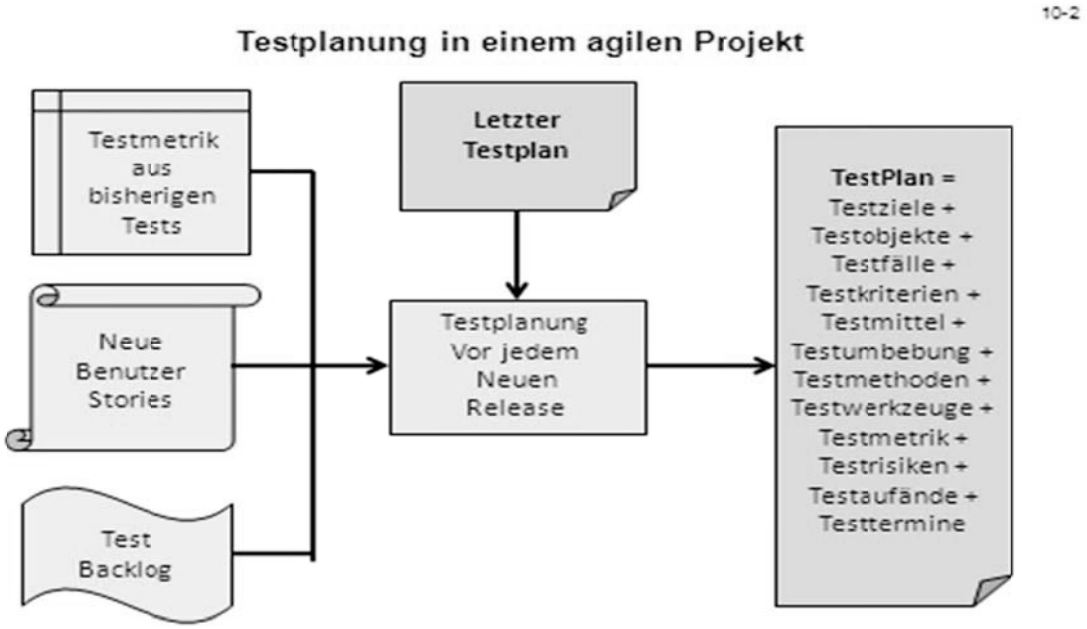

Abbildung 2: Testplanung in einem agilen Projekt

3 Agile Testaufwandsschätzug

In einem konventionellen Entwicklungsprojekt ist der Test oft ein paralleles Projekt, welches getrennt geplant, geschätzt, organisiert und verfolgt wird. Das Testprojekt hat einen eigenen Testmanager, der das Team leitet. Zu seinen Aufgaben gehört es, den Testaufwand zu schätzen, und zwar den ganzen Testaufwand. Geschätzt wird in der Regel auf der Basis der Anforderungsspezifikation bzw. des Lastenheftes. Falls es ein Entwurfsmodel gibt, wird dies hinzugezogen. Durch die Analyse der Anforderungen wird die Anzahl der erforderlichen Testfälle ermittelt. Entweder nimmt der Testmanager die Anzahl Testfälle als das Maß für den Umfang des Tests oder er justiert diese Zahl durch die Anzahl Benutzerschnittstellen und Datenbanken, um daraus Test-Points zu errechnen. Ob mit der nackten Anzahl Testfällen oder mit der Anzahl justierter Testfälle bzw. Test-Points, der Testumfang wird mit einem Größenmaß ausgedruckt. Dieser Größenmaß wird anschließend über eine Testproduktivitätstabelle in Tester-Arbeitstage umgesetzt, z.B. 8 Testfälle bzw. 10 Testpunkte pro Testertag. Daraus folgt der geschätzte Testaufwand in Personentage [Sned03]. Die Voraussetzung für diese Schätzung ist eine zuverlässige Testproduktivitätstabelle, die aus der Erfahrung mit bisherigen Testprojekten abgeleitet wird. Ohne zu wissen, wie umfangreich der Test wird und wie produktiv die Tester arbeiten, kann der Testmanager nur grobe Analogien zu vergleichbaren Testprojekten ziehen [MMSW13].

Natürlich spielt der Grad der Testautomation hier auch eine Rolle. Entweder spiegelt sich dieses Grad bereits in der Testproduktivität wieder, oder sie wird als Multiplikationsfaktor genommen, um den geschätzten Aufwand zu justieren. Ist der geschätzte Testaufwand 400

Personentage und das Automatisierungsgrad 50%, wird der Aufwand auf 200 Personentage reduziert. Der Grad der Testautomatisierung hat einen großen Einfluss auf die Testproduktivität und die Testproduktivität bestimmt zusammen mit dem Testumfang und der angestrebten Testüberdeckung den Testaufwand. Es kann sein, dass insgesamt 2000 Testfälle sich aus der Anforderungsanalyse ergeben, aber nur 75% Anforderungsüberdeckung angestrebt wird. In diesem Fall reduziert sich die Testfallanzahl auf 1500. Es bleibt dann zu entscheiden, welche 500 Testfälle ausgeklammert werden. Hierzu werden Methoden wie Risikobasiertes Testen und Triage angewandt. Es kommt selten vor, dass sämtliche erforderliche Testfälle wirklich getestet werden, auch nicht in konventionellen Projekten, in agilen Projekten wird es noch seltener sein [SnJu06].

In agilen Entwicklungsprojekten gelten die User-Stories, in denen die Anforderungen verkleidet sind, nur für das nächste Release. Demnach kann nur der Aufwand für den Test des nächsten Releases geschätzt werden, d.h. die Testerin muss die Testaufwandsschätzung alle 4-6 Wochen wiederholen. Der Vorteil dabei ist, dass sie die Testproduktivität von Release zu Release kalibrieren kann, so dass ihre Schätzungen immer besser werden. Der Nachteil ist, dass sie zu Beginn der Entwicklung nicht wissen kann, wie umfangreich der Test insgesamt wird. Das kann sie nur raten, oder allenfalls Vergleiche zu anderen agilen Projekten ziehen.

Der Testaufwand für das nächste Release lässt sich sehr wohl systematisch schätzen weil er überschaubar ist. Statt eine vollständige Anforderungsdokumentation zu analysieren, analysiert der Tester die einzelnen Stories. Der Begriff „Story-Point“ wird in der agilen Entwicklung verwendet, um den Aufwand für die Entwicklung des nächsten Release vorherzusagen. Falls es sich herausstellt, dass der Aufwand für die Implementierung der anstehenden „Stories“ zu hoch ist um in einem Release zu bringen, werden Storys gestrichen oder geschmälert. Das Team hat also die Wahl ganze Storys wegzulassen oder einzelne Storys zu vereinfachen. Diese Verkleinerung des geplanten Release wird so lange wiederholt, bis der geschätzte Aufwand zum verfügbaren Aufwand passt. Ein Team mit 5 Entwickler und einem Tester hat bei einem Release-Zyklus von einem Monat eben nur 5 Personenmonate zur Verfügung. Es kann nur so viele Storys annehmen, die in diesem Zeitrahmen hineinpassen. Wenn das Team 5 Story-Points pro Tag implementieren kann, sind das in einem Monat 100 Story-Points. Die Zahl der Story-Points für ein Release darf diese Grenze nicht überschreiten.

Story-Points sind nirgendwo genau definiert. Am ehesten entsprechen sie einer Regel oder einem Schritt in einem Anwendungsfall. Eine Story entspricht in etwa einem Anwendungsfall – z.B. wie buche ich einen Flug nach Timbuktu. Der Benutzer formuliert den Vorgang, den er automatisieren möchte als Story. Jeder Vorgang hat *n* mögliche Ausgänge – z.B. der Flug ist ausgebucht, der Flug ist gestrichen oder ich bekomme einen Platz. Diese Varianten sind Testfälle. Sie können auch Story-Points sein. Es wird für das Schätzen einfacher, wenn Story-Points und Testfälle gleichgesetzt werden. Ein Testfall testet ein Story-Point bzw. eine Anforderung. Eine Testfallreihe bzw. eine Testszenario testet ein Story, bzw. einen Anwendungsfall [Seib12].

So muss der Tester nur wissen, wie viele Testfälle = Story-Points er pro Tag testen kann. Wenn er 8 Testfälle testen kann, sind das rund 150 Testfälle oder Story-Points pro Monat. Da kann er die 100 Story-Points, die von den Entwicklern in einem Release implementiert werden, leicht bewältigen. Wenn er jedoch nur 4 Testfälle pro Tag testen kann, sind das 80 Story-Points in einem Release – 20 weniger, als von den Entwicklern implementiert werden. Jetzt müsste er entscheiden, welche 20 Fälle er nicht testet und dies mit dem Team besprechen. Wenn das Team damit einverstanden ist, ist es ok, sonst müssen die Entwickler ihm helfen die verbleibenden Fälle zu testen.

Ein Grundprinzip der agilen Entwicklung ist der Release-Umfang der verfügbaren Kapazität anzupassen. Stories oder Features, die über den möglichen Aufwand hinausgehen, kommen als Einträge in das Backlog und werden auf das nächste Release verschoben. Das gleiche gilt

für Testfälle. Beim Schätzen schließt der Tester von der verfügbaren Zeit und Kapazität auf den Umfang seines Tests, bzw. auf die Anzahl der Testfälle die er testen kann. (siehe Abbildung 3)

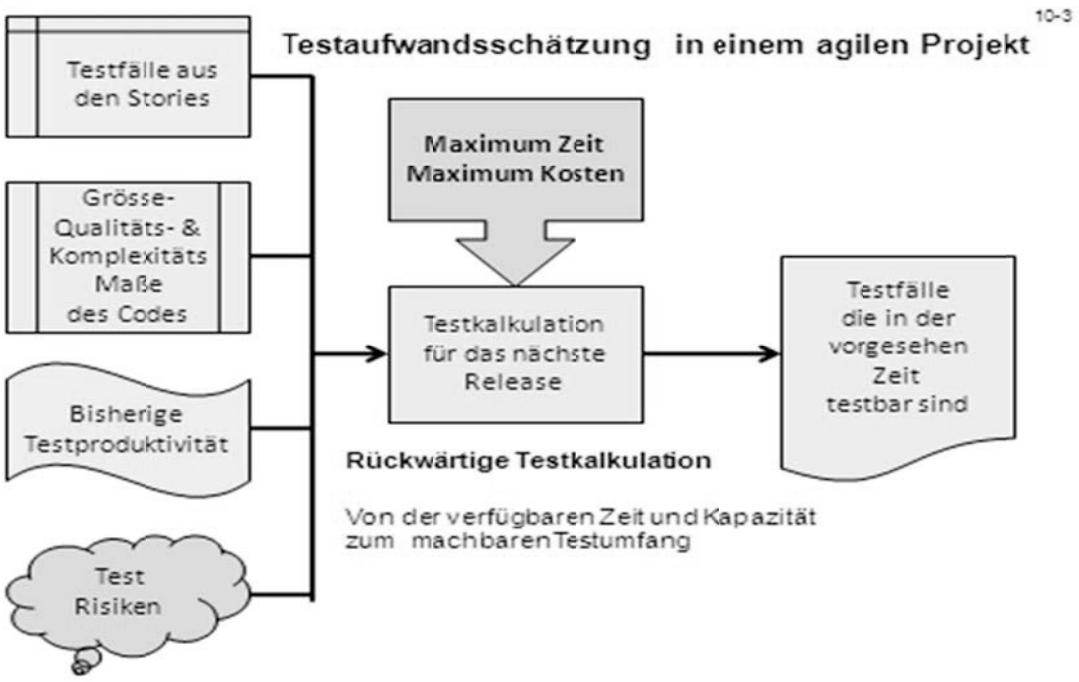

Abbildung 3: Testaufwandschätzung in einem agilen Projekt

Falls die erforderliche Testfallanzahl über die Zahl hinausgeht, die der Tester in einem Release testen kann, werden sie in das Test-Backlog eingetragen und bleiben in der Warteschlange. Beim nächsten Release werden sie wieder berücksichtigt. Es kann aber sein, dass das nächste Release auch zu viele Testfälle hat. In dem Fall werden sie wieder verschoben. Auf dieser Weise wächst die Zahl der ungetesteter Story-Points bzw. Testfälle von Release zu Release. Um sie wieder abzubauen, muss irgendwann ein Test-Release eingeschoben werden. Sonst wäre das „technical debt“ zu hoch. Dieses ist ein Release, in dem nur getestet und korrigiert wird. Möglicherweise könnte das gleiche Test-Release von den Entwicklern benutzt werden um den Code zu sanieren. Derartige Releases müssten regelmäßig stattfinden, um die Qualität der Software auf einem gewissen Niveau zu halten. Es gebe auch Verfahren um die Höhe der technischen Schulden zu errechnen [CuSS12]. Dies zu tun wäre auch eine Aufgabe der Tester.

Gerade deshalb ist es unerlässlich den Testaufwand für jedes Release neu zu schätzen. Dafür gibt es gewichtige Gründe. Falls es sich herausstellt, dass der Testumfang in Testfälle oder in Test-Points zu hoch ist, kann der Tester

a) das Team ersuchen, das Release zu verkleinern in dem es einige Storys zurückstellt
b) den nötigen Aufwand reduzieren, in dem Testfälle entweder ganz gestrichen oder zurückgestellt werden.

Für die letztere Alternative muss der Tester eine Testrisikoanalyse durchführen und die Testfälle nach ihrer Fehlerfindungswahrscheinlichkeit sowie nach ihrer Kritikalität ordnen. Die Kritikalität hängt davon ab, welche Funktionen mit dem Testfall ausgelöst werden, ob das kritische oder weniger kritische Funktionen sind. Kritische Funktionen wiegen natürlich mehr. Die Fehlerfindungswahrscheinlichkeit hängt von der Anzahl Funktionen, die ein Testfall durchläuft, bzw. von der Länge des Testpfades ab. Je mehr Funktionen von einem

Testfall erreicht werden, desto höher die Wahrscheinlichkeit, dass ein Fehler in einer der Funktionen gefunden wird [OGPM12].

Es gibt ein breites Angebot an Literatur zum Thema „Risikobasiertes Testen“ auf das hier verwiesen wird [Duva07]. Jedenfalls obliegt es dem Tester, den erforderlichen Testaufwand mit der Releasezeit in Einklang zu bringen, Das, was er sich vornimmt, muss bis zur Releasefreigabe wirklich erfüllt sein, auch wenn er nur die Hälfte der Testfälle sind, die er testen kann. Die Testüberdeckung sollte geplant sein und nicht einfach zufällig geschehen.

4 Agile Testorganisation

Es gibt nicht die optimale Organisation für einen agilen Test. Bei konventionellen Projekten hat es eine separate Testgruppe mit einem eigenen Testmanager, ein Projekt im Projekt. So etwas gibt es nicht in der agilen Entwicklung, Testmanager schon gar nicht. In der agilen Entwicklung managt jeder sich selbst. Dies gilt auch für die Tester.

Es kann jedoch durchaus sein, dass mehrere Tester in einem agilen Projektteam mitarbeiten, und einer ist der „primus inter pares“, also der Leittester. Er sorgt dafür, dass der Iterationsplan samt Abnahmekriterien zustande kommt und dass jeder Tester einen gerechten Anteil an den Testaufgaben bekommt. Natürlich wird die Aufteilung der Anstehenden Testaufgaben im Team besprochen und mit dem Einverständnis aller Beteiligten verabschiedet. In diesem Punkt sind die Prinzipien des agilen Managements etwas widersprüchlich. Auf der einer Seite soll alles im Einvernehmet mit allen Teammitgliedern entschieden. Auf der anderen Seite soll das Release in kürzester Zeit ausgeliefert werden. Gleichzeitig darf nicht mehr als 40 Stunden der Woche gearbeitet werden. Dass klingt in etwa wie die Quadratur des Kreises, aber aus Berichten aus der agilen Praxis soll es möglich sein [Eck04]. Es ist auf jeden Fall anzustreben, nach Konsens zu arbeiten und dass setzt ein recht homogenes Team voraus. Es muss vom Anfang an darauf geachtet werden, wer in das Team hineinkommt, auch als Tester [Cohn04].

In der Regel wird es, wenn überhaupt, nur ein oder zwei Tester in einem agilen Entwicklungsprojekt geben. Sie werden sich einigen können, wer was macht. Sie werden auch das Einverständnis der anderen Teammitglieder gewinnen. Wichtig ist, dass sie geschlossen auftreten. Wenn die eine Testerin abwesend ist, kann die Andere sie vertreten. Vielleicht ist die eine Testerin mehr fachlich orientiert und die Andere mehr technisch. Jedenfalls muss man gut überlegen, wer mit wem zusammenarbeitet. Gemeinsam sollen sie die Testaufgaben mit dem Anwendervertreter, bzw. mit dem Produkt-Owner, absprechen und die Abnahmekriterien ermitteln. Die Testerin spezifiziert die Testfälle, baut die Testumgebung auf, bedient die Testwerkzeuge und führt den Test durch. Wenn sie technische Unterstützung braucht, wendet sie sich an die Entwickler. Wenn sie fachliche Unterstützung braucht wendet sie sich an die Anwender. Die allein arbeitende Testerin im agilen Team muss jedenfalls sehr vielseitig sein und auch noch kommunikativ. Drum ist es oft besser gleich zwei Tester zu haben, damit sie sich gegenseitig unterstützen können. (siehe Abbildung 4)

Bei größeren agilen Projekten mit über 20 Mitarbeitern wird es möglich sein ein separates Test-Team zu bilden. Dieses Team arbeitet in klassischer Weise neben den Entwicker-Teams her. Der Unterschied zu früher ist eben, dass die Release-Intervalle kurzer sind. Möglicherweise sind die Tester ein Release zurück. Derartige Projekte soll es auch in der agilen Welt geben, die entfernt von der Entwicklung stattfinden [Fowl06].

Schließlich wird es agile Projekte geben – kleine Projekte mit bis zu vier Personen – in denen es gar keinen Tester gibt. Die Entwickler übernehmen die Testaufgaben, einschließlich der Testplanung und der Testauswertung. Jeder Entwickler integriert und testet die eigene Komponente und präsentiert die Ergebnisse dem Anwender [Ehle05]. Der Test des ganzen

Systems ist Sache des Anwenders, er muss es auch abnehmen, Natürlich wird er dabei von den Entwicklern unterstützt. In solchen Projekten liegt der Test in der Verantwortung aller Teammitglieder. Sie müssen sich einigen, wer welche Testaufgaben übernimmt.

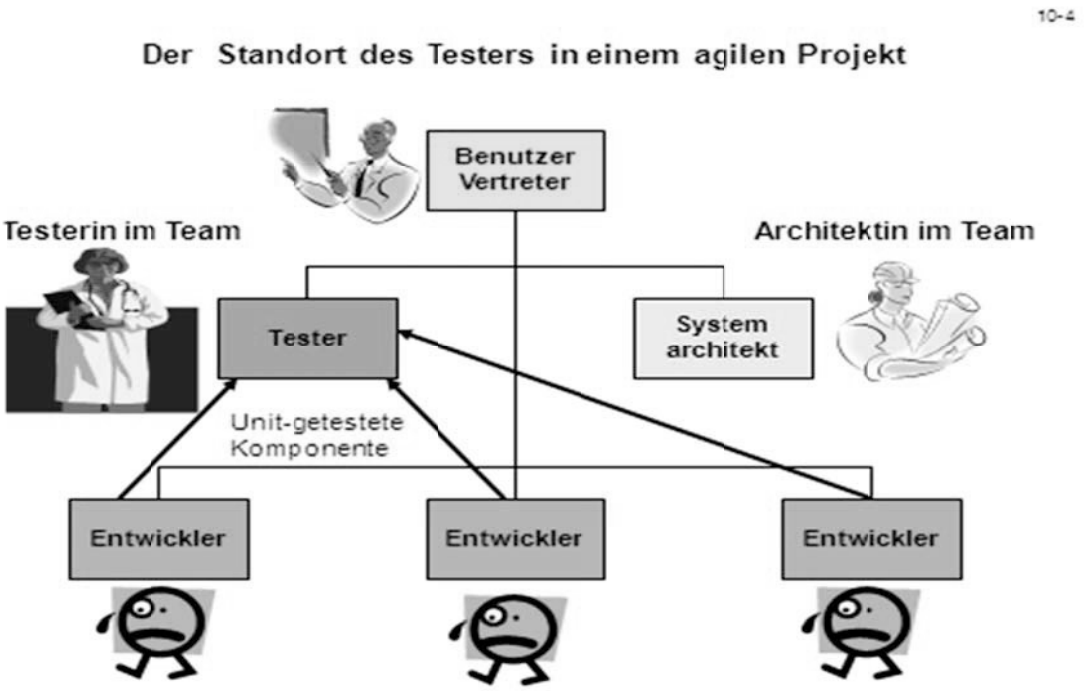

Abbildung 4: Standort des Testers in einem agilen Projekt

5 Agile Testüberwachung

Testüberwachung ist die Voraussetzung für Teststeuerung. Um steuern zu können, braucht man Information und die Information stammt aus der Überwachung. In klassischen Testprojekten wurde

- der Status der gemeldeten Fehler
- die Fehlerhäufigkeit
- die Testüberdeckung
- die Testaufwände
- die Testproduktivität und
- die Testtermine

überwacht. Die gleichen Überwachungsmaßnahmen gelten für den agilen Test. Die Tester brauchen nach wie vor diese Berichte um ihren eigenen Test zu steuern. Was bei einem agilen Test anders geworden ist, ist das Adressat der Berichte. In konventionellen Projekten wurden diese Berichte für den Testmanager oder den Projektleiter gedacht. Sie haben dazu gedient eine bürokratische Steuerung zu ermöglichen und einem Qualitätsmanagement zu futtern. In modernen agilen Projekten gibt es weder ein Testmanager noch einen Projektleiter noch ein Qualitätsmanagement. Der Sinn einer „Agilen Lean Entwicklung“ besteht darin diese Overhead Funktionen abzustreifen [PoCu12]. Wer außer dem Scrum-Master nicht operativ tätig ist – als Benutzervertreter, Entwickler oder Tester – hat in einem agilen Projekt nichts zu suchen. Das Team steuert sich selbst und dies gilt auch für die Tester. Das hat zur Folge, dass

die Berichte nicht den gleichen Grad an Formalität haben müssen wie in nicht-agilen Projekten. Es komme nur auf den Inhalt an und nicht auf die Form. (siehe Abbildung 5)

5.1 Fehlermeldung

Der Hauptwert eines agilen Tests liegt darin Fehler rechtzeitig aufzudecken [Bull00]. Deshalb werden die Fehlermeldungen auch als Karteikarten an einem Fehlerbrett im Projektteam-Raum ausgestellt. Die Fehlerkarten sollten dort nach Priorität der Behebung geordnet werden mit den kritischen Fehlern oben und den geringen Fehler unten. Ein Farbschema soll die Fehlerschwere sichtbar machen, z.B. rot für kritisch, orange für schwerwiegend, gelb für mittelschwer und blau für gering. Der Status der offenen Fehler sollte ein Thema beim täglichen Stand-Up-Meeting sein [Risi02]. Es soll kein Tag vergehen ohne den Backlog an Fehlern zu besprechen. Das Team soll an den offenen Problemen ständig erinnert werden. Sobald ein Fehler beseitigt ist wird seine Karte von dem Fehlerbrett entfernt. Die offenen Fehlermeldungen sind auch ein Gesprächsthema bei den täglichen „Stand-Up“ Treffen. Sie zu berichten und zu verfolgen ist eine Aufgabe der Tester. (siehe Abbildungen 5 und 6)

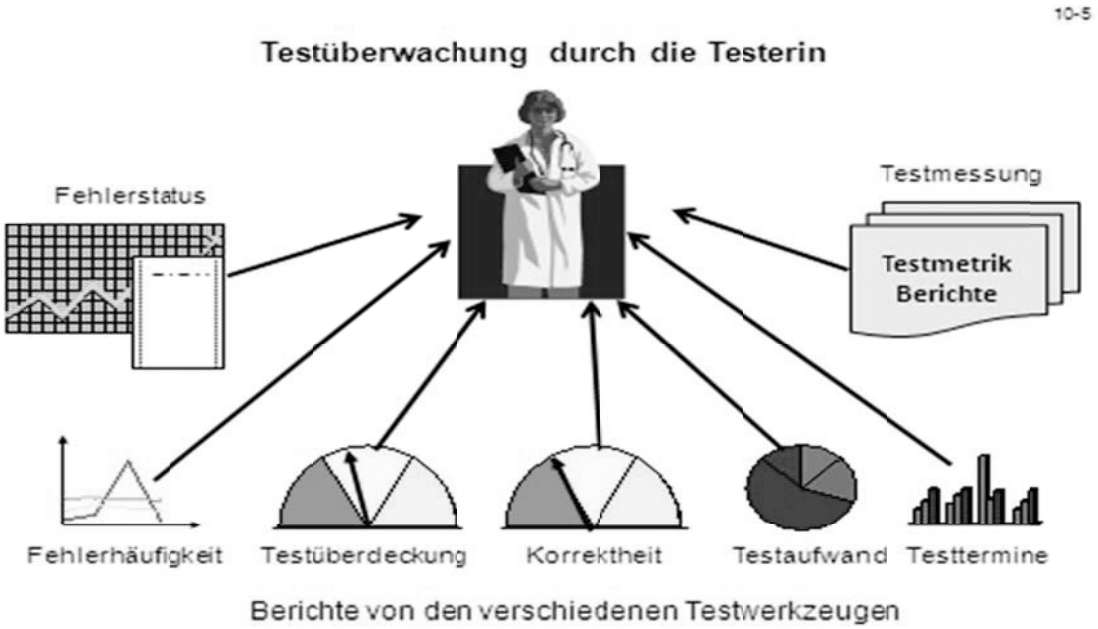

Abbildung 5: Testüberwachung

5.2 Fehlerhäufigkeit

Die Fehlerhäufigkeit ist ein Indikator für die Qualität der Software. Gemessen wird sie an Hand des Verhältnisses der gemeldeten Fehler zu den getesteten Testfällen. Am besten wird sie in einer Graphik dargestellt, die im Projektraum ausgestellt wird. Sie soll dort mindestens täglich aktualisiert werden. So bekommen die Entwickler ihre eigene Leistung immer vors Auge geführt. Die Fehlerübersicht dient auch als Nachweis für die Leistung der Tester im Team. Ihre Aufgabe ist es die Fehler der Entwickler vor der nächsten Release-Freigabe zu finden und möglichst unbürokratisch mitzuteilen. Es ist außerdem ihre Aufgaben die Anzahl der zu erwartenden Fehler im nächsten Release hochzurechnen [Fried02]. Ein Histogramm welches das Verhältnis der Fehler zu der Codemenge visualisiert ist ein geeignetes Mittel dies zu erreichen. (siehe Abbildung 7)

Abbildung 6: Fehlermeldungstafel

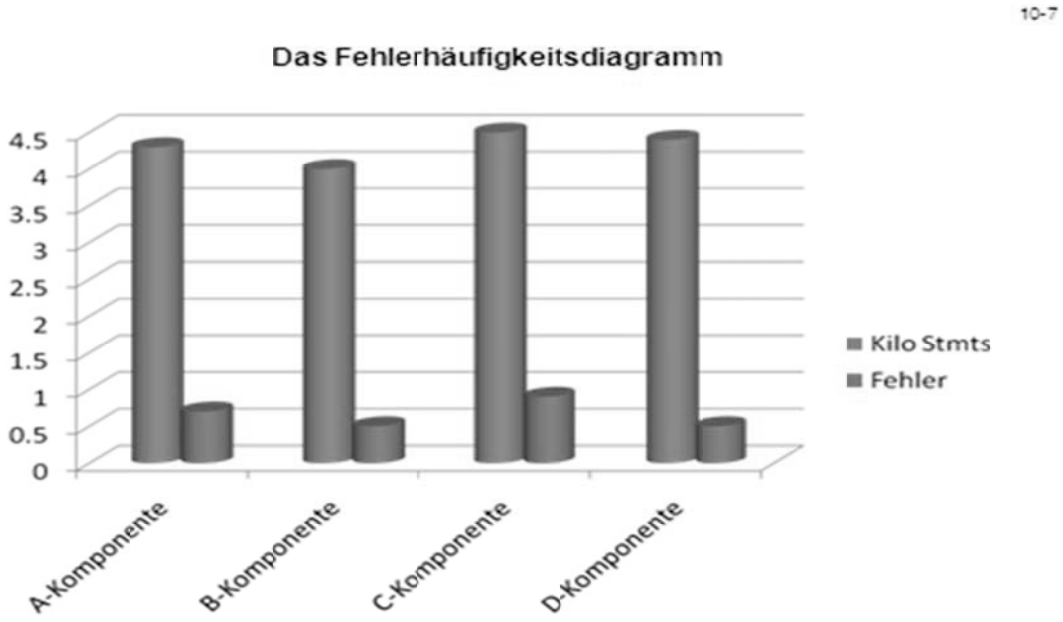

Abbildung 7: Fehlerhäufigkeien

5.3 Testüberdeckung

Die Testüberdeckung ist der Grad zu dem die Software getestet wird. Sie kann viele Bedeutungen haben, je nach dem, was gemessen wird. Codeüberdeckung ist der Grad, zu dem bestimmte Codeelemente ausgeführt worden sind. Es werden z.B. Anweisungen, Zweige, Pfade oder Methoden gezählt. Einmal, wie viele es insgesamt gibt und zweitens, wie viele davon im Test durchlaufen werden. Dieser Maß gilt hauptsächlich für den Unit-Test bzw. für

den Test der Entwickler. Datenüberdeckung ist der Grad, zu dem die Daten verwendet werden. Datenfelder, deren Inhalte sich im Laufe des Tests mindestens einmal verändert, gelten als verwendet, auch wenn sie von dem Benutzer gesetzt oder von einem Tool generiert werden. Das Ziel hier ist, möglichst alle Datentypen in den Test einzubeziehen. Dieser Maß ist besonders für den Integrationstest interessant, bei dem die Datenschnittstellen getestet werden. Funktionsüberdeckung ist eigentlich mit Anforderungsüberdeckung gleichzusetzen. Sie ist der Grad, zu dem alle Anforderungen ausprobiert worden sind [Kuhn09].

Dies kann auch bedeuten, alle Pfade durch alle Anwendungsfälle zu testen. Dies ist ein Maß für den Systemtest. Für den agilen Abnahmetest empfiehlt sich die Abnahmeüberdeckung. Hier wird gezählt, wie viele der Abnahmekriterien sind geprüft worden. Das ist ein anderes Maß. Schließlich kann der Grad der Testfallüberdeckung gemessen werden. Das ist der Anteil der spezifizierten Testfälle, der tatsächlich ausgeführt wird. Es kann sein, dass viele spezifizierte Testfälle aus Zeitgründen nie ausgeführt werden. Dann ist die Summe aller Testfälle nur ein Wunschziel – das Soll. Die Summe der ausgeführten Testfälle ist die Realität – das Ist. Die Testüberdeckung ist ein Verhältnis zwischen dem Test-Soll und dem Test-Ist [Roth01].

Das Team soll bei der Planung eines Release bestimmen, welche Testüberdeckungsziele anzustreben sind. Die Testwerkzeuge werden messen, zu welchem Grade diese Ziele erreicht sind. Die Berichte der Werkzeuge, auch die graphische Darstellungen wie Kuchendiagramme, Histogramme, Verteilungskurven und Dashboards, sind täglich im Projektraum auszuhängen. Ein Kuchendiagramm ist besonders geeignet die Testüberdeckung darzustellen. Da kann jedes Teammitglied den Fortschritt des Tests, sowohl was der Unit-Test als auch was der Integrationstest und der Abnahmetest anbetreffen, verfolgen.

5.4 Testaufwandserfassung

Es stimmt zwar, dass agile Projekte auf Vertrauen zueinander aufgebaut sind [McCL12]. Dennoch soll jedes Teammitglied verpflichtet sein, seine Arbeitsstunden zu buchen. Dafür wird es ein Aufwands-Erfassungssystem geben. In diesem System wird es Aufwandskategorien geben, wie Kommunikation, Entwicklung, Test und Dokumentation. Damit wird festgehalten, wie viele Stunden in jeder Tätigkeitskategorie gearbeitet werden, auch wie viele Stunden getestet werden. Es wird für die Entwickler nicht immer einfach sein, ihre Teststunden von den Entwicklungsstunden zu unterscheiden, vor allem dann, wenn sie testgetrieben arbeiten. Deshalb ist zu empfehlen, nur die Stunden, die ausschließlich dem Test gewidmet sind, z.B. den Integrationstest, als Teststunden zu buchen. Der Tester wird hingegen den großen Anteil seiner Arbeitszeit als Test anrechnen. Der Rest der Zeit wird auf Kommunikation und Dokumentation aufgeteilt. Sollte jedoch der Tester auch Werkzeuge entwickeln oder installieren, soll diese Zeit gegen die Entwicklung gebucht werden.

Das Aufwands-Erfassungssystem wird verschiedene Berichte produzieren, darunter die Aufwandsverteilung nach Tätigkeitskategorie. Ein Histogramm soll das Verhältnis des geplanten Testaufwandes zum gebuchten Testaufwand für alle im Team sichtbar machen. (siehe Abbildung 8).

Darin ist zu erkennen, wie viel Aufwand bis zu diesem Zeitpunkt in den Test eingeflossen ist. Manche werden fragen: wozu das gut sein soll? Dieser Aufwand soll mit der Fehleranzahl und der Testüberdeckung vom Team verglichen werden, um die eigene Testleistung zu beurteilen. Denn auch in einem agilen Projekt gilt das Leistungsprinzip. Die Leistung, sprich die Produktivität soll steigen. sie soll auch mit anderen Teams vergleichbar sein. Schließlich will das Management wissen, ob der agile Ansatz sich lohnt. Was der Test anbetrifft, ist Leistung das Verhältnis der Testüberdeckung zu dem Testaufwand. Das Ziel ist eine

möglichst hohe Testüberdeckung mit einem möglichst geringen Aufwand zu erreichen [JaJa12].

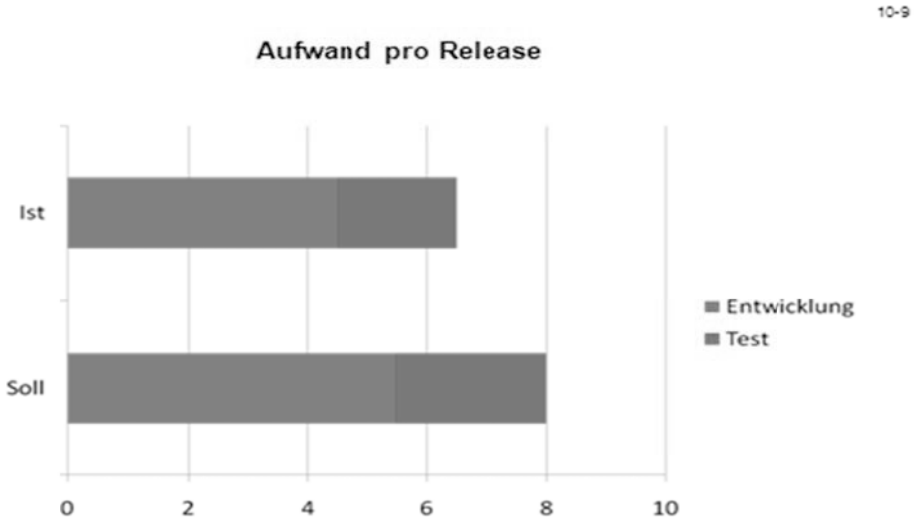

Abbildung 8: Testaufwand pro Release

5.5 Testproduktivitätsmessung

Bei einer agilen Entwicklung kommt es darauf an, eine möglichst hohe Produktivität zu erreichen. Dies ist letzten Endes der Sinn der Agilität – Produktivität kommt vor Qualität. Entwickler sollten möglichst viele Story-Points mit möglichst wenig Aufwand abarbeiten. Übertragen auf den Test heißt dass möglichst viele Testfälle in möglichst kurzer Zeit ausführen. Das Produktivitätsmaß für den Test ist die Anzahl Testfälle pro Testertrag [Sned05]. Dies gilt weiterhin für den agilen Test. Die agile Testerin soll ihre eigene Produktivität messen und versuchen sie zu steigern. Mit einer entsprechenden Graphik kann sie die anderen Teammitarbeiter von ihrer Leistung überzeugen. Sie muss schließlich ihre Leistung verkaufen. (siehe Abbildung 9)

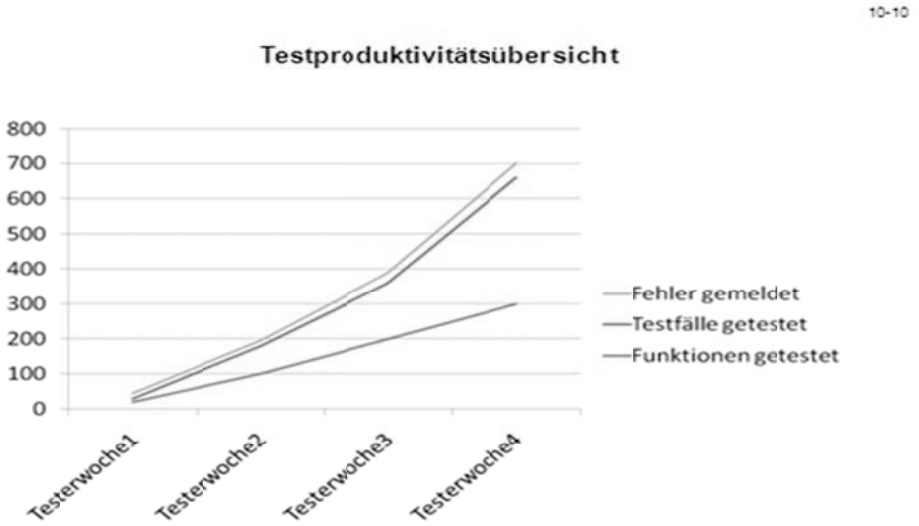

Abbildung 9: Testproduktivität

5.6 Testtermine

Das letzte Fortschrittsmaß ist der Grad, zu dem die Termine eingehalten werden. Der wichtigste Termin ist der vereinbarte Release-Termin. Ein jedes Teammitglied soll ständig vors Auge geführt bekommen, wie viel Zeit noch bis zum Release-Termin übrig bleibt. Sein augenblickliches Tun hängt von dieser Metrik ab. Jede Aktion, die er unternimmt, jedes Gespräch, das er führt und jede Entscheidung, die er trifft, wird von der verbleibenden Zeit bestimmt. Zeit ist Trumpf. Darum ist es wichtig, die Zeit für alle sichtbar zu machen.

Es wird auch innerhalb einer Iteration Zwischentermine geben. Ein solcher Termin ist die Abgabe der Komponente durch die Entwickler. Die Zwischentermine der Testerin sind:

- die Fertigstellung der Abnahmekriterien
- die Abgabe des Iterationstestplanes
- die Integration einzelner Komponente
- der Abschluss des Regressionstests
- der Abschluss des Abnahmetests.

Die Testerin soll ihre Termine in Absprache mit dem Benutzervertreter selber setzen und selber kontrollieren. Die Terminplanung dient letztendlich zur Selbstkontrolle. Es wird keine Instanz von außen mit Repressalien drohen, wenn ein Zwischentermin nicht eingehalten wird. Der einzige Termin, der von außenkontrolliert wird, ist der Releasetermin. Er ist unumstößlich.

5.7 Testfortschrittsüberwachung

Manch einer wird fragen, ob die Projektüberwachung in einem agilen Entwicklungsprojekt überhaupt sinnvoll ist? Wurde der agile Ansatz nicht eingeführt, um die Software-bürokratie abzuschaffen? Projektüberwachung ist aber ohne Bürokratie nicht möglich. Die Alternative wäre keine Daten zu erheben und das Projekt einfach laufen zu lassen, bis das Budget und/oder die Geduld der Stake-Holders erschöpft sind. Natürlich hat der Anwender alle 2-6 Wochen ein neues lauffähiges System und kann daran den Fortschritt des Projektes erkennen. Anderseits weiß er nicht, was ihm diese weitere Funktionalität kostet, wenn nicht die Aufwände irgendwie gebucht werden. Man könnte auch nicht wissen, wie viel Anteil der Test an den Gesamtkosten hat, und ob dieser Anteil gerechtfertigt ist [KoKr12].

Crispin schreibt in ihrem Buch, dass auch in den agilen Projekten ein Minimum an bürokratischem Overhead unvermeidlich sei. Das Team schuldet dies den Geldgebern. Sie erwähnt sogar die Rolle eines Projekt-Trackers, jemand, der nur damit beschäftigt ist, die Kosten und Termine des Projektes zu verfolgen. Sie wehrt sich allerdings dagegen, dass der Tester in diese Rolle gedrängt wird. Also hat die Projektverfolgung ihre Rechtfertigung [CrGr09]. Crispin, wie alle Befürworter der agilen Entwicklung, befindet sich hier in einem gewissen Dilemma. Sie wollen mit möglichst wenig Management-Overhead auskommen. Anderseits lassen sich gewisse administrative Aufgaben nicht vermeiden. Das agile Team ist verpflichtet seine Kosten gegenüber dem Geldgeber zu rechtfertigen. Die Projektbeteiligten müssen eben die Zeit nehmen, um ihren Aufwand zu erfassen und ihre Zwischentermine zu posten. Es gebe zwar keinen Projektleiter, um diese Daten auszuwerten, aber die Information dient dem Team selbst seine Entscheidungen zu treffen und seine weiteren Iterationen zu planen. Der Projektgeldgeber hat auch ein Recht darauf über die Höhe seiner Unkosten informiert zu werden.

6 Agile Teststeuerung

In konventionellen Projekten mit einem ordentlichen Projektleiter dient die Projektüberwachung dazu dem Projektleiter mit Information zu versorgen um das Projekt steuern zu können. Es liegt bei ihm, Termine zu verschieben Überstunden anzuordnen oder Funktionen zu streichen. Er vertritt das Projekt gegenüber dem Kunden und trägt dafür die Verantwortung. Demzufolge hat er eine gewisse Macht über die anderen Projektmitglieder. Entweder hat er das Recht Aufgaben anzuweisen oder die Fähigkeit, Menschen zu bewegen etwas zu tun, was sie sonst von sich aus nicht machen würden. Eine dritte Möglichkeit Menschen zu führen ist es, sie sich selbst zu überlassen und sich selber aus Überzeugung oder aus Einsicht in die Notwendigkeit vorantreiben. Das Letztere ist der Weg der agilen Entwicklung [Blac09]. Die Tester im Team sollten selber wissen was sie zu tun haben was der Test anbetrifft. Sie brauchen keinen Leiter um Ihnen zu sagen was sie als nächstes machen müssen. Was sie selber nicht schaffen können, müssen sie von den Entwicklern anfordern. Ein Anweisungsrecht gibt es nicht. Jedes Teammitglied muss sich auf die Einsicht seiner Teampartner verlassen können, dass sie ihm helfen wenn Not am Mann ist. Die Tester im Team sind jedenfalls dafür verantwortlich, dass gewisse Testaufgaben in einer gewissen Reihenfolge erledigt werden. (siehe Abbildung 10)

In einem konventionellen Projekt sind die Tester einem Testmanager unterstellt. Entweder gibt es ein eigenständiges Testprojekt im Projekt mit einem eigenen Testprojektleiter, oder die Tester werden von der Qualitätssicherungsabteilung ausgeliehen und unterstehen dem Qualitätsmanager. Wenn es keinem expliziten Testmanager gibt, haben die Tester an den Projektleiter zu berichten. Jedenfalls gibt es jemanden über sie, der sie steuert. In einem agilen Projekt gibt es keine oben und unten. Alle Teammitglieder sind gleich. Ergo ist jeder auf sich selbst angewiesen. Die Tester sind nur dem Team gegenüber verantwortlich, d.h. sie berichten an das Team. Sie planen die eigenen Zwischenziele und teilen ihre Aufwände selber ein.

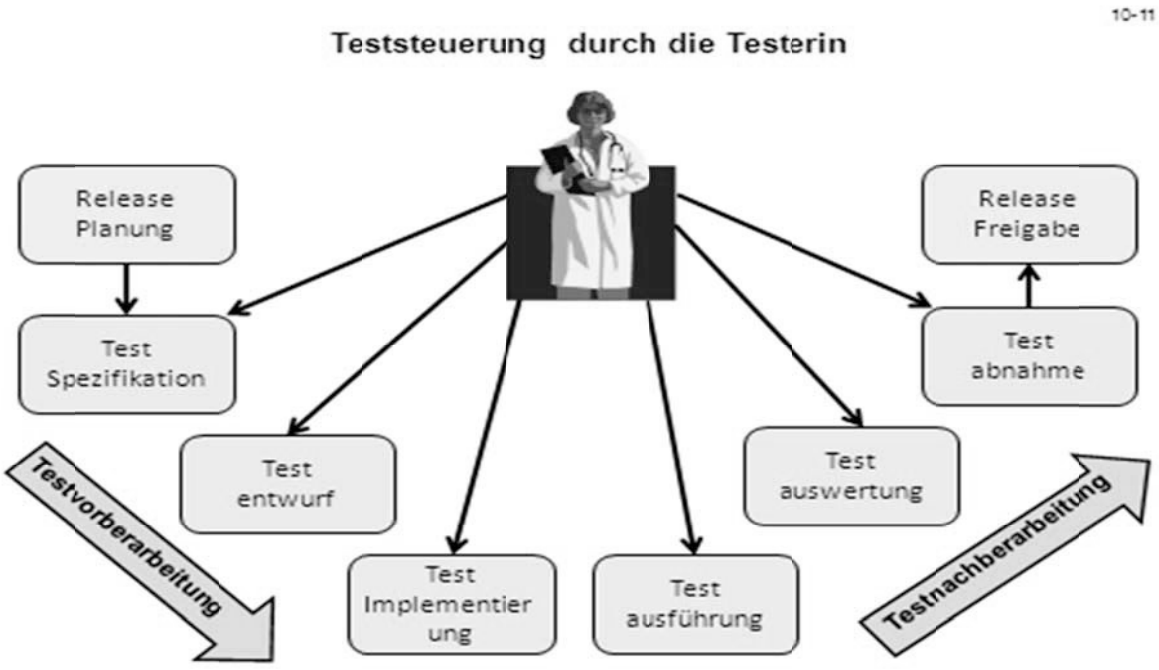

Abbildung 10: Teststeuerung

Selbststeuerung ist nicht jedermanns Sache. In Amerika, die Geburtsstätte der agilen Bewegung sind die Menschen eher daran gewohnt, sich selbst zu steuern. Sie sind auch daran gewohnt, für sich selber zu sorgen und die Verantwortung für das eigene Versagen zu tragen. In Mitteleuropa, die Geburtsstätte des Sozialstaates haben die Menschen weniger Erfahrung damit. Sie sind gewohnt, gesteuert zu werden und die Verantwortung für ihr Tun und Lassen anderen, höhergestellten zu überlassen. Von den Mitteleuropäern verlangt agile Entwicklung, und damit auch der agile Test, eine gewisse Anpassung. Sie sollen lernen mehr Verantwortung für sich selbst zu übernehmen.

Agile Tester müssen sich von hierarchischen Denkmustern befreien und sich als gleichberechtigter Partner in einem Team von Spezialisten sehen, in dem jede seine Rolle zu spielen hat. Die Rolle der agilen Tester ist es, die Qualität des Produktes zu sichern. Wie sie dies erreichen, ist ihnen selbst überlassen. Sie können eine passive Rolle annehmen und die anderen Teammitglieder dazu animieren, mehr selber zu tun, um die Qualität ihrer Ergebnisse zu sichern, oder sie übernehmen eine aktive Rolle und sichern die Qualität der Software selber durch Tests und Prüfungen, die sie in eigener Regie ausführen. Idealerweise machen sie beides. Sie fordern die Entwickler auf, mehr selber zu testen und unternehmen gleichzeitig von sich aus eigene Aktivitäten, um die Qualität der Software sichern. Sie nehmen den Unit-Test der Entwicklern ab, prüfen den Code, testen die Integration der Komponente und führen den Abnahmetest durch. Alles, was sie tun und lassen, müssen sie dem Team als ganzes gegenüber rechtfertigen und dessen Unterstützung dafür gewinnen. Die Testerin steuert zwar sich selbst, wird aber letztendlich von dem Team mitgesteuert. Das Team setzt die Prioritäten für den Test, also was als nächstes zu testen ist und zu welchem Grade. Sichtbar werden die Prioritäten an Hand der Kanban Technik, die den Standort der anstehenden Aufgaben relativ zueinander abbildet [Gart12]. Demnach sollten agile Tester kollegiale und kompromissbereite Personen sein. Sie haben sich in das Dreiecksverhältnis – Benutzer, Entwickler, Tester – einzufügen. Ihre Rolle dort ist als Vermittler zwischen Benutzern und Entwicklern zu agieren. Sie sollen in dieser Rolle dafür sorgen, dass die Benutzer das bekommen was sie verlangt haben und das die Entwickler ausreichende Qualität liefern. Die technischen Schulden, bzw. „technical debt“, sollten möglichst gering bleiben [Gern03]. (siehe Abbildung 11)

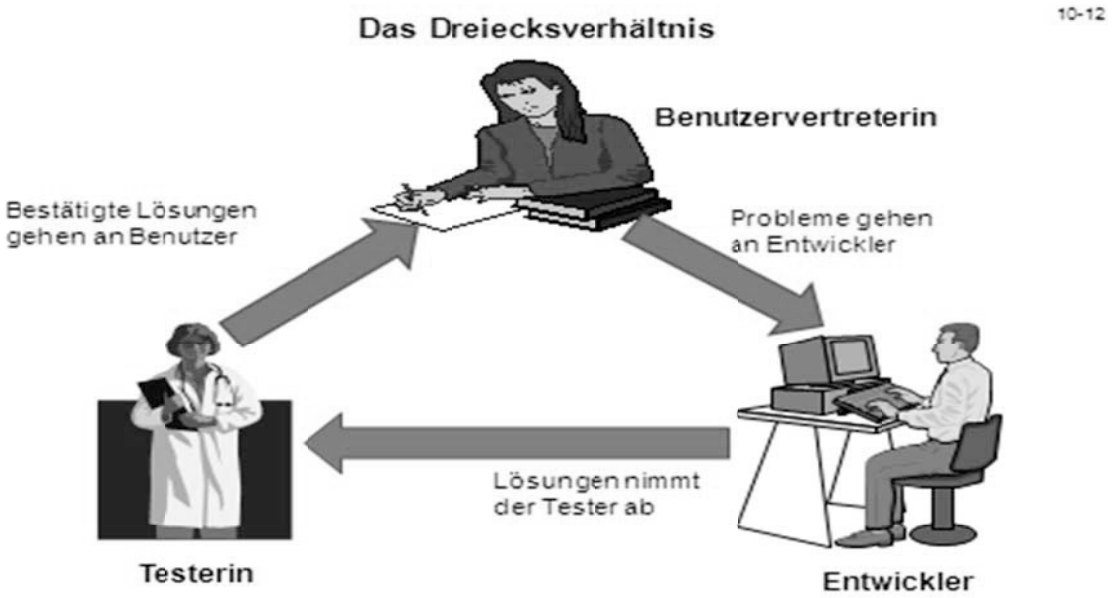

Abbildung 11: Dreiecksverhältnis – Benutzer, Testerin, Entwickler

7 Folgen agiler Projekte für die Organisation

Betriebe die agile Projekte betreiben, müssen gewisse Folgen aus dem Wesen jener Projekte ziehen. Agile Projekte setzen eine unbürokratische, flexible, flache und offene Umgebung voraus. In einer Sonderausgabe des Objektspektrums von September 2012 setzen sich mehrere Wortführer der agilen Bewegung im deutschsprachigen Raum mit dem Thema agiles Management auseinander [Cold12]. Johannes Mainusch, Leiter der Software Entwicklung bei der Otto GmbH stellt die Frage, ob Agil und Management sich nicht grundsätzlich widersprechen. Er setzt dabei Management und Hierarchie auf die gleiche Stufe und benutzt das Beispiel der Wiener Polizei, als abschreckendes Beispiel einer klassischen Managementhierarchie. Was er dabei übersieht ist das die Stadt Wien mit derartigen Hierarchien aus der Kaiserzeit in allen Lebenslagen seit Jahrhunderten überlebt hat, auch in der IT. Das muss nicht unbedingt schlecht sein. In einer Hierarchie benutzen Manager auf einer höheren Hierarchiestufe Information, die nur ihnen zusteht, um Mitarbeiter auf einer unteren Hierarchiestufe zu beherrschen. Das setzt voraus, dass die Unteren sich aus beherrschen lassen, was in Wien lange der Fall war. Das Problem heute ist dass jeder Zugang zur gleichen Information hat. Also hat keiner die Möglichkeit dem Anderen was vorzuenthalten.

Diese Art hierarchisches Management ist nicht nur unzeitgemäß, sie steht auch klar im Widerspruch zu den Prinzipien der Agilität. In einer agilen Welt hat jeder Zugang zu allen Informationen, also kann keiner den anderen etwas vorenthalten. Sie sind alle gleich. Es dürfte nach dem radikaler Vertreter agiler Entwicklung überhaupt keine Manager mehr geben. Laut Mainusch soll der agile Manager „*das Ziel haben, den eigenen Verantwortungsbereich möglichst so eigenständig zu machen, dass er selbst als Manager überflüssig wird. Die agile Organisation sollte daher als Wert haben, die überflüssigen agilen Ex-Manager auf einem anderen Gebiet mit neuen, spannenden Aufgaben wieder einzusetzen…*“ [Manu12]. Dies trifft um so mehr für den Bereich Test zu, wo Tester ohnehin in einem selbstbestimmten Team integriert sind. Für die heutigen Testmanager bedeutet dies, dass die ihren Managertitel abgeben und zurück ins Reih und Glied treten. Möglicherweise könne sie als Berater und Lehrer der anderen Tester weiterarbeiten.

In der gleichen Ausgabe des Objektspektrums erscheint ein Praxisbericht der Allianz AG. Die Allianz betreibt agile Entwicklungsprojekte mit einer Tochtergesellschaft der Allianz, Cornhill Information Services – ACIS – mit Sitz in Trivandrum, Indien. Bei ihnen läuft neben dem Entwicklungsprojekt ein Continouos Integration Project, in dem die abgegebenen Komponente qualitätsgeprüft und getestet werden. Die Integrationsarbeiten finden in einer separaten Umgebung mit einem eigenen Server statt. Dort wird mit dem Statischen Analysator Sonar und dem Testwerkzeug Selenium gearbeitet. Sonar prüft den Code auf die Einhaltung der Richtlinien während Selenium die Funktionalität der Komponente bestätigt. Dieses Integrationsprojekt finden in der Konzernzentrale statt, aber auch dieses Testteam managt sich selbst, Von Testmanager ist keine Rede, Allianz legt großen Wert auf die Selbstorganisation für Teams und dies gilt ebenso für das Testteam [RuGr12].

Für Menschen, die eine Managementfunktion anstreben, bietet der agile Testbereich wenige Möglichkeiten, es sei denn sie finden sich mit der Rolle eines Projekt-Trackers ab. Lisa Crispin erwähnt diese Rolle in Zusammenhang mit ihren Projekten. Der Projekt-Tracker sammelt Daten über das Projekt – Termine, Aufwände, Fehlerzahlen, Fertigungsgrad usw.von den anderen Teammitgliedern und berichtet diese an die Stakeholders. Diese ist als reine administrative Funktion einzustufen. Der Projekt-Tracker verfügt über keinerlei

Führungskompetenz und darf nicht in das Projekt hineinreden. Er ist nur dazu da, den Fortschritt zu verfolgen.

Ansonsten gibt es in agilen Projekten keine Manager. Wenn einer nicht selber testet oder Testwerkzeuge entwickelt und betreut, hat er in einem agilen Projekt nichts zu suchen. Er kann allenfalls, wie bereits erwähnt, als Testberater tätig sein und die Entwickler bei den eigenen Tests beraten so wie das der Fall in einem israelischen Luftwaffenprojekt war. Das Wort „Manager“ ist in der agilen Welt ohnehin zum Unwort geworden. Deshalb ist es etwas widersprüchlich überhaupt von einem agilen Testmanagement zu sprechen. Agilität ist eigentlich der Totengräber der Testmanager.

Literaturhinweise

[Black99] Black, Rex: Managing the Testing Process, MicroSoft Press, Redmond, 1999

[Beck99] Beck, K.: Extreme Programming explained, Addison-Wesley, Reading, 1999

[IEEE829] IEEE: ANSI/IEEE Standard 829 – Standard for Software Test Documentation, ANSI/IEEE Standard 829-1998, Computer Society Press, New York, 1998

[Seib12] Seibert, J.: „Agiles Schätzen im Team – Verfahren in der agilen Softwareentwicklung“, Objektspektrum, Nr. 5, Sept. 2012, S. 48

[Sned03] Sneed, H.: Software Testmetriken für die Kalkulation der Testkosten und die Bewertung der Testleistung, GI Softwaretechnik Trends, Band 23, Heft 4, Nov. 2003, S.11

[MMSW13] Metzger, A./Münzel, J,/ Simon, F./ Weißleder, S.: „ISQB-Expert Level – Test Management in modernen Systemen“, Objektspektrum, Nr. 1, Jan. 2013, S. 58

[SnJu06] Sneed, H./ Jungmayr, S.: „Produkt- und Prozessmetriken für den Softwaretest“, Informatikspektrum, Band 29, Nr. 1, Feb. 2006, S. 23

[CuSS12] Curtis, B./ Sappidi, J./ Szynkarski, A.: “Estimating the Principle of an Application’s Technical Debt”, IEEE Software, Dez. 2012, S. 34

[OGPM12] Opelt, A., Gloger, B./ Pfarl, W./ Mittermayr, R.: Der agile Festpreis, Hanser Verlag, München, 2012

[Duva07] Duvall, P./Matyas, S./Glover, A.: Continuous Integration – Improving Software Quality and reducing Risk, Addison-Wesley, Reading Ma., 2007

[Eck04] Eckstein, J.: Agile Software Development in the Large- Diving into the Deep, Dorset House, Boston, 2004

[Cohn04] Cohen, C./ Birkin, S./ Garfield, M./ Webb, H.: „Managing Conflict in Software Testing”, Comm. of ACM, Vol. 47, Nr. 1, Jan. 2004, S. 76

[Fowl06] Fowler, M.: “Using an Agile Software Process with Offshore Development”, martinfowler.com/articles/agileOffshore.htm, 2006

[Ehle05] Ehle, B., Sirotin, V.: „Erfahrungsbericht – Testgetriebene Entwicklung von Web-Anwendungen, Objektspektrum, Nr. 2, 2005, S. 61

[PoCu12] Poppendieck, M./ Cusumano, M.: “Lean Software Development – A Tutorial”, IEEE Software, Sept. 2012, S. 26

[Risi02] Rising, L.: „Agile Meetings“, STQE Magazine, Vol. 4. Nr. 3, June 2002, S. 42

[Fried02] Friedenberg, G.: “Forecasting Software Defects”, STQE Magazine, Vol. 4. Nr. 3, June 2002, S. 26

[Kuhn09] Kuhn, R./ Kacker, R./ Lei, Y.: „Combinatorial Software Testing“, IEEE Computer Magazin, August 2009, S. 94

[Roth01] Roth, A.: „How Good is this Software“, STQE Magazine, Vol. 3. Nr. 5, Sept. 2001, S. 35

[McCL12] McHugh, O., Conboy, K., Lang, M.: “Agile Practices – The Impact of Trust in Software Project Teams”, IEEE Software, May 2012, S. 71

[JaJa12] Janus, A./Jäger, J/ Gaedke, M.: „Agile Praktiken – oder doch Impediments – Bewertung der Agilität von Praktiken in der Softwareentwicklung“, Objektspektrum, Nr. 5, Sept. 2012, S. 28

[Sned05] Sneed, H.: Software-Projektkalkulation – Aufwandsschätzung verschiedener Projekttypen, Hanser Verlag, München, 2005

[KoKr12] Köster, M./ Krakau, H.: „Total Quality Management – Nutzung von Agilität zur Adaption in der IT“, Objektspektrum, Nr. 5, Sept. 2012, S. 58

[CrGr09] Crispin, L. / Gregory, J.: Agile Testing – A practical Guide for Testers and agile Teams, Addison-Wesley-Longman, Amsterdam, 2009

[Blac09] Black, S./Boca, P./Hinchley, M.: “Formal versus Agile – Survival of the Fittest”, IEEE Computer, Sept. 2009, S. 37

[Gart12] Gartner, M., Roeck, A.: „Kanban fuer Tester – Evolutionäres Qualitätsmanagement“, Objektspektrum, Nr. 4, 2012, S. 30

[Gern03] Gernert, C.: Agiles Projektmanagement – Risikogesteuerte Softwareentwicklung, Hanser Verlag, München, 2003

[Cold12] Colderley, J.: ‘Agiles Management – eine Herausforderung” Objektspektrum, Nr. 5, Sept. 2010, S. 80

[Manu12] Mainusch, J.: “Agiles Management – ein Widerspruch in sich?”, Objektspektrum, Nr. 5, Sept. 2022, S. 8

[RuGr12] Ruoss, H.,, Graßl, M.: „Agilität in der Allianz Global Reinsurance – Softwareentwicklung mit Offshore-Partnern“, Objektspektrum, Nr.5, Sept. 2012, S.22