ISTQB® Certified Tester

Foundation Level
Extension Syllabus Agile Tester

VERSION 2017, deutschsprachige Ausgabe

International Software Testing Qualifications Board

Herausgegeben durch Austrian Testing Board, German Testing Board e.V. und Swiss Testing Board

Änderungsübersicht

Version	Datum	Bemerkungen
DACH v1.1	Mai 2017	Release-Fassung (basierend auf Review-Kommentaren)
DACH v1.0	September 2014	Release-Fassung (basierend auf ISTQB Syllabus 2014)
DACH v0.9	Juli 2014	Beta-Fassung zur Information für Trainingsanbieter und Zertifizierungsstellen (basierend auf ISTQB Syllabus 2014)

Inhaltsverzeichnis

Dank

Die englischsprachige Fassung wurde erstellt durch: Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Learning Objectives Lead), Debra Friedenberg (Exam Lead), Alon Linetzki (Business Outcomes and Marketing Lead), Tauhida Parveen (Editor) und Leo van der Aalst (Development Lead).

Autoren: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh und Stephan Weber.

Interne Reviewer: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael und Erik van Veenendaal.

Die deutschsprachige Fassung wurde erstellt durch: Armin Born (Autor), Martin Klonk (Autor), Kai Lepler (Reviewer), Tilo Linz (Leitung/Autor), Anke Löwer (Autor), Thomas Müller (Reviewer), Richard Seidl (Leitung Prüfungsfragen/Autor), Alexander Weichselberger (Autor), Dr. Stephan Weißleder (Reviewer) und Markus Zaar (Autor).

Einführung in diesen Lehrplan

0.1 Zweck dieses Dokuments

Dieser Lehrplan dient als Basis für die Internationale Software Test Qualifikation im Foundation Level für agile Tester. Das ISTQB bietet diesen Lehrplan folgenden Zielgruppen an:

- Nationalen Ausschüssen zur Übersetzung in ihre Landessprache und für die Zulassung von Schulungsanbietern.
- Nationale Ausschüsse können den Lehrplan ihren speziellen Sprachgegebenheiten anpassen und die Referenzen in Anlehnung an die örtlichen Veröffentlichungen modifizieren.
- Prüfungsausschüssen für die Erstellung von Prüfungsfragen in ihrer Landessprache, die an die Lernziele für jeden Lehrplan angepasst sind.
- Anbietern von Trainings zur Erstellung von Kursmaterialien und zur Festlegung angemessener Lehrmethoden.
- Kandidaten für die Zertifizierung zur Prüfungsvorbereitung (als Teil eines Schulungskurses oder unabhängig)
- Der internationalen Software- und Systemanalysegemeinschaft zur Förderung des Berufs des Software- und Systemtesters und als Basis für Bücher und Fachartikel.

Das ISTQB kann die Nutzung dieses Lehrplans durch andere Instanzen und zu anderen Zwecken per vorheriger, schriftlicher Erlaubnis zulassen.

0.2 Überblick

Die Qualifikation zum agilen Tester auf Foundation Level Stufe ist durch folgende Lehrpläne definiert:

- ISTQB Certified Tester - Foundation Level Syllabus, [ISTQB_FL_SYL]
- ISTQB Certified Tester - Foundation Level - Agile Tester Extension, (das vorliegende Dokument)

Das Dokument [ISTQB_FA_OVIEW] gibt hierzu weitere Informationen.

0.3 Prüfungsrelevante Lernziele

Auf Basis der Lernziele werden die Prüfungsfragen erstellt, die in einer Zertifizierungsprüfung zum ISTQB Certified agile Tester auf Foundation Level zu beantworten sind. Grundsätzlich sind alle Teile dieses Lehrplans prüfungsrelevant auf der kognitiven Ebene K1. Das bedeutet, der Kandidat soll einen Begriff oder ein Konzept erkennen und sich daran erinnern. Die spezifischen Lernziele für die Stufen K1, K2 und K3 sind am Anfang des entsprechenden Kapitels genannt.

1. Agile Software Entwicklung – 150 min

Schlüsselwörter

Agiles Manifest, agile Softwareentwicklung, inkrementelles Entwicklungsmodell, iteratives Entwicklungsmodell, Softwarelebenszyklus, Testautomatisierung, Testbasis, testgetriebene Entwicklung, Testorakel, User-Story

Kapitelspezifische Lernziele

1.1 Die Grundlagen der agilen Softwareentwicklung

FA-1.1.1 (K1) Das Grundkonzept der agilen Softwareentwicklung basierend auf dem agilen Manifest beschreiben können
FA-1.1.2 (K2) Die Vorteile des Whole-Team-Approaches (interdisziplinäres, selbstorganisiertes Team) verstehen
FA-1.1.3 (K2) Den Nutzen von frühen und häufigen Rückmeldungen verstehen

1.2 Aspekte agiler Ansätze

FA-1.2.1 (K1) Die Ansätze der agilen Softwareentwicklung nennen können
FA-1.2.2 (K3) User-Stories schreiben in Zusammenarbeit mit Entwicklern und Vertretern des Fachbereichs
FA-1.2.3 (K2) Verstehen, wie in agilen Projekten Retrospektiven als Mechanismus zur Prozessverbesserung genutzt werden
FA-1.2.4 (K2) Die Anwendung und den Zweck von kontinuierlicher Integration (continuous integration) verstehen
FA-1.2.5 (K1) Die Unterschiede zwischen Iterations- und Releaseplanung kennen und wissen, wie sich ein Tester gewinnbringend in jede dieser Aktivitäten einbringt

1.1 Die Grundlagen der agilen Softwareentwicklung

Ein Tester in einem agilen Projekt arbeitet anders als ein Tester in einem traditionellen Projekt. Tester müssen die Werte und Prinzipien verstehen, die agile Projekte stützen. Sie müssen verstehen, dass Tester genauso wie Entwickler und Fachbereichsvertreter gleichberechtigte Mitglieder des Teams sind (Whole-Team-Approach). Sie kommunizieren von Beginn an regelmäßig miteinander, was der frühzeitigen Fehlerreduzierung dient und hilft ein qualitativ hochwertiges Produkt zu liefern.

1.1.1 Agile Softwareentwicklung und das agile Manifest

Im Jahr 2001 einigte sich eine Gruppe von Personen, die die am weitesten verbreiteten leichtgewichtigen Softwareentwicklungsmethoden vertrat, auf einen gemeinsamen Kanon von Werten und Prinzipien, die als das Manifest für agile Softwareentwicklung oder das „agile Manifest“ [agilemanifesto] bekannt wurde. Das agile Manifest formuliert vier Werte:

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Funktionierende Software ist wichtiger als umfassende Dokumentation.
- Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Reagieren auf Veränderungen ist wichtiger als das Befolgen eines Plans.

Das agile Manifest postuliert, dass obwohl die Werte auf der rechten Seite wichtig sind, die Werte auf der linken Seite einen höheren Stellenwert haben:

Individuen und Interaktion

Agile Entwicklung ist personenzentriert. Personen schreiben Software im Team und Teams können am effektivsten über direkte, kontinuierliche Kommunikation mit persönlicher Interaktion arbeiten, statt indirekt über Werkzeuge oder Prozesse.

Funktionierende Software

Aus Kundensicht ist eine funktionierende Software mit wenig Dokumentation nützlicher und wertvoller als eine schlechter funktionierende Software mit übermäßig detaillierter Dokumentation. Da funktionierende Software, wenn auch mit reduzierten Funktionalitäten, viel früher im Entwicklungslebenszyklus verfügbar ist, kann agile Entwicklung einen bedeutenden Zeitvorsprung bis zur Marktreife generieren. Das ermöglicht wiederum früheres Feedback der Anwender an die Entwickler. Agile Entwicklung ist daher besonders in sich schnell verändernden Geschäftsumgebungen nützlich, in denen die Probleme und/oder Lösungen unklar sind oder in denen das Unternehmen Innovationen für neue Geschäftsbereiche erzielen möchte.

Zusammenarbeit mit Kunden

Für Kunden stellt die Beschreibung des von ihnen benötigten Systems oft eine große Schwierigkeit dar. Die direkte Zusammenarbeit mit dem Kunden erhöht die Wahrscheinlichkeit, seine Wünsche genau zu verstehen. Verträge mit Kunden sind sicherlich wichtig. Allerdings wird die Wahrscheinlichkeit für den Erfolg eines Projektes durch die regelmäßige und enge Zusammenarbeit mit dem Kunden steigen.

Reagieren auf Veränderungen

Veränderungen sind in Softwareprojekten unumgänglich. Die Umgebung in der das Unternehmen arbeitet, die Gesetzgebung, die Aktivitäten der Mitbewerber, Technologiefortschritte und andere Faktoren können einen starken Einfluss auf das Projekt und seine Ziele haben. Diese Faktoren müssen im Entwicklungsprozess berücksichtigt werden. Eine flexible Arbeitspraxis, die auf Veränderungen reagiert und Pläne in kurzen Zeitintervallen (Iterationen) anpasst, ist erfolgreicher und wichtiger als eine Praxis, die an einmal gefassten Plänen festhält.

Prinzipien

Die wesentlichen Werte des agilen Manifests werden in zwölf Prinzipien festgehalten [agileManifesto Prinzipien]:

1. Es hat höchste Priorität, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufriedenzustellen.
2. Anforderungsänderungen, selbst spät in der Entwicklung, werden begrüßt. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
3. Funktionierende Software wird regelmäßig innerhalb weniger Wochen oder Monate geliefert, mit Präferenz für die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
5. Projekte werden rund um motivierte Individuen aufgebaut - ihnen wird das notwendige Umfeld und die Unterstützung gegeben, die sie benötigen und darauf vertraut, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode Informationen an und innerhalb eines Entwicklungsteams zu übermitteln ist im Gespräch von Angesicht zu Angesicht.
7. Funktionierende Software ist das wichtigste Fortschrittsmaß.
8. Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Anwender sollten ein gleichmäßiges Tempo dauerhaft halten können.
9. Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
10. Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren - ist essenziell.
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Die unterschiedlichen agilen Ansätze liefern konkrete Vorgehensweisen, um diese Werte und Prinzipien umzusetzen.

1.1.2 Whole-Team-Approach

Unter dem Whole-Team-Approach (einem interdisziplinären, selbstorganisierten Team) versteht man den Einbezug aller Personen (Fachbereich, Entwickler, Tester, andere Stakeholder) in enger Zusammenarbeit mit Wissen und Fähigkeiten, die für den Projekterfolg notwendig sind.
Das Team beinhaltet auch Vertreter des Kunden, welche die Produktmerkmale festlegen. Ein erfolgreiches agiles Team sollte klein sein, die optimale Teamgröße liegt zwischen drei und neun Personen, da Selbstorganisation bei dieser Teamgrößen besonders optimal erfolgt. Im Idealfall teilt sich das gesamte Team einen Raum, da diese Anordnung Kommunikation und Interaktion stark vereinfacht. Der Whole-Team-Approach wird durch tägliche Stand-up-Meetings (siehe auch Abschnitt 2.2.1) unterstützt, die alle Teammitglieder einbeziehen, in denen der Arbeitsfortschritt kommuniziert wird und jegliche Hinderungsgründe (Impediments) für den Fortschritt benannt werden. Der Whole-Team-Approach fördert somit eine effektive und effiziente Teamdynamik.

Die Nutzung des Whole-Team-Approaches für die Produktentwicklung wird als einer der Hauptvorteile der agilen Entwicklung angesehen, da er eine Reihe von Vorteilen aufweist, u. a.:

- Er fördert die Kommunikation und Zusammenarbeit innerhalb des Teams.
- Er nutzt die unterschiedlichen Fähigkeiten aller Team-Mitglieder als Beitrag zum Projekterfolg.
- Er erhebt die Qualität zum Ziel jedes Einzelnen.

Tester werden in der Zusammenarbeit sicherstellen, dass die gewünschten Qualitätsstandards erreicht werden. Das beinhaltet unter anderem die Unterstützung und Zusammenarbeit mit den Fachbereichsvertretern, um ihnen zu helfen, passende Abnahmetests zu erstellen, die Zusammenarbeit mit Entwicklern, um sich auf die Teststrategie zu einigen und die Entscheidung über

Automatisierungsansätze. Tester können ihre Testkenntnisse anderen Teammitgliedern vermitteln und so die Entwicklung des Produktes fördern.

Das gesamte Team wird in alle Besprechungen und Meetings einbezogen, in denen Produktfeatures präsentiert, analysiert oder geschätzt werden. Das Konzept Tester, Entwickler und Vertreter des Fachbereichs in alle Diskussionen über die Produktfeatures mit einzubeziehen ist auch als „The Power of Three“ bekannt [Crispin08].

1.1.3 Frühe und regelmäßige Rückmeldung

Agile Projekte haben kurze Iterationen, die es dem Projektteam ermöglichen, frühe und kontinuierliche Rückmeldungen (Feedback) von allen zur Produktqualität über den gesamten Entwicklungslebenszyklus hinweg zu erhalten. Eine Möglichkeit zur schnellen Rückmeldung ist die kontinuierliche Integration (continuous integration) (siehe Abschnitt 1.2.4).

In sequentiellen Entwicklungsansätzen sieht der Kunde das Produkt spät und oft erst am Ende des Entwicklungsprozesses zum ersten Mal. Zu diesem Zeitpunkt ist es dann für das Entwicklungsteam oft zu spät, um Rückmeldungen des Kunden noch zu berücksichtigen.

In agilen Projekten erfolgen Rückmeldungen regelmäßig während des gesamten Projektzeitraums. Dadurch erhalten agile Teams frühzeitig und regelmäßig neue Informationen, die sie kurzfristig in den Produktentwicklungsprozess einfließen lassen können. Dieses Feedback hilft dabei, den Fokus auf die Features zu setzen, die den höchsten wirtschaftlichen Wert, oder das höchste zugeordnete Risiko haben. Diese werden dem Kunden als Erstes geliefert.

Durch regelmäßige Rückmeldungen lernt das agile Team auch etwas über seine eigenen Möglichkeiten. Zum Beispiel, wie viel können wir in einem Sprint oder einer Iteration schaffen? Was könnte uns helfen, schneller zu werden? Was hindert uns daran?

Die Vorteile der frühen und regelmäßigen Rückmeldung beinhalten:

- Vermeiden von Missverständnissen bezüglich der Anforderungen, die erst im Laufe des Entwicklungszyklus erkannt und deren Umsetzung dann teurer würde
- Klären von Kundenanforderungen zu Produktfeatures, indem die wichtigsten Features früh für die Nutzung durch den Kunden zur Verfügung stehen. Auf diese Weise werden Kundenwünsche im Produkt besser abgedeckt
- Frühes Entdecken, Isolieren und Lösen von Qualitätsproblemen durch kontinuierliche Integration (continuous integration)
- Liefern von Informationen für das agile Team bezüglich seiner Produktivität und seiner Fähigkeiten, das Gewünschte zu realisieren
- Fördern einer beständigen Projektdynamik

1.2 Aspekte agiler Ansätze

Es sind eine ganze Reihe von agilen Ansätzen in Gebrauch. Verbreitete Praktiken in vielen agil entwickelnden Organisationen sind z. B. : die gemeinsame Erstellung von User-Stories, die Durchführung von Retrospektiven, kontinuierliche Integration (continuous integration) oder das agile Planen für das gesamte Release sowie für jede einzelne Iteration. In diesem Abschnitt werden einige dieser agilen Ansätze näher betrachten.

1.2.1 Ansätze agiler Softwareentwicklung

Es gibt nicht den einen Ansatz für agile Softwareentwicklung, sondern eine Vielzahl unterschiedlicher Ansätze. Jeder dieser Ansätze setzt die Werte und Prinzipien des agilen Manifests etwas anders um.

In diesem Lehrplan skizzieren wir die populärsten Vertreter dieser agilen Ansätze: Extreme Programming (XP), Scrum und Kanban.

Extreme Programming (XP)

XP, ursprünglich von Kent Beck [Beck04] eingeführt, ist ein agiler Ansatz der Softwareentwicklung, der durch bestimmte Werte, Prinzipien und Entwicklungspraktiken gekennzeichnet ist.

XP beinhaltet fünf Werte, die die Entwicklung leiten sollen:

- Kommunikation,
- Einfachheit,
- Rückmeldung,
- Mut,
- Respekt.

XP beschreibt eine Liste von Prinzipien als zusätzliche Richtlinie:

1. humanity (Menschlichkeit: schaffen einer an den menschlichen Bedürfnissen der Projektmitglieder ausgerichteten Atmosphäre),
2. economics (Wirtschaftlichkeit: die Entwicklung ist sowohl wirtschaftlich als auch wertig),
3. mutual benefit (beidseitiger Vorteil: Die Software stellt alle Beteiligten zufrieden),
4. self-similarity (Selbstgleichheit: Wiederverwendung bestehender Lösungen),
5. improvement (Verbesserung: Aus regelmäßig gewonnenem Feedback wird die Lösung/der Prozess ständig verbessert),
6. diversity (Vielfalt: Vielfalt im Team (Fähigkeiten, Charaktere) erhöht die Produktivität),
7. reflection (Reflexion: Erkennen besserer Lösungen durch stetige Reflexion),
8. flow (Gleichmäßig mit hoher Konzentration arbeiten: Ein stetiger Arbeitsdurchfluss und kurze Iterationen gewährleisten, das Projekt im Fluss zu halten),
9. opportunity (Gelegenheiten wahrnehmen: Schwierigkeiten bzw. Fehlschläge bei der Umsetzung sollten als Gelegenheit und Chance erachtet werden),
10. redundancy (Redundanzen vermeiden: Unnötig wiederholte oder auch manuelle Schritte, die automatisierbar wären, sollen vermieden werden),
11. failure (Fehlschläge hinnehmen: Eine zunächst nicht optimale bzw. fehlerhafte Umsetzung wird akzeptiert, sofern diese als Herausforderung gesehen wird),
12. quality (Qualität: Hohe Softwarequalität ist wichtig),
13. baby steps (kleine Schritte: Kurze Iterationszyklen ermöglichen zeitnahes Feedback, schnelle Kompensation von Fehlschlägen, sowie Flexibilität in Bezug auf Rahmenbedingungen),
14. accepted responsibility (akzeptierte Verantwortung: Die Verantwortung wird durch das Team übernommen. Das bedeutet umgekehrt auch, dass das Management nicht vorschreibt, was und wie etwas zu tun ist).

XP beschreibt darüber hinaus dreizehn primäre Praktiken:

1. sit together (räumliche Nähe: Optimieren der Kommunikation durch gemeinsame Anordnung der Arbeitsplätze),
2. whole-team (interdisziplinäres, selbstorganisiertes Team: Es gilt das Bewusstsein, nur als Gemeinschaft erfolgreich zu sein),
3. informative workspace (informativer Arbeitsplatz: Wichtige Informationen sollen vom Arbeitsplatz aus sichtbar sein (z. B. aktuelle Tasks, Stand des Projekts),
4. energized work (energievolle Arbeit: Motiviertes Arbeiten bei gleichzeitig entspannter Atmosphäre. Entwickeln ohne Überstunden),
5. slack (entspannte Arbeit: Erläuterung siehe energized work)

6. pair programming (Programmieren in Paaren: Förderung des sich selbstregulierenden Teams durch Programmieren mit abwechselnden Partnern),
7. stories (Stories: Die zu entwickelnde Funktionalität wird in Form von User-Stories beschrieben),
8. weekly cycle (wöchentlicher Zyklus: in wöchentlichem Zyklus wird entschieden, was als nächstes umgesetzt wird),
9. quaterly cycle (quartalsweiser Zyklus: Das Projekt selbst wird in quartalsweisen Zyklen geplant),
10. ten-minute build (10-Minuten-Build: Build (das Herstellen eines integrierten Versionsstands) und (automatisierte) Tests sollen in maximal 10 Minuten durchgeführt werden können. Das minimiert Kosten),
11. kontinuierliche Integration (continuous integration): Alle Änderungen sollen in kurzen regelmäßigen Zyklen seitens der Entwickler bereitgestellt werden),
12. test-first programming (testgetriebene Entwicklung: Vor Realisierung der Funktionalität muss der Test geschrieben werden),
13. incremental design (inkrementelles Design: Durch inkrementelles Design, in das Feedback und Erkenntnisse einfließen, wird die Software stetig verbessert).

Viele Ansätze agiler Softwareentwicklung, die heute in Gebrauch sind, wurden durch XP und seine Werte und Prinzipien beeinflusst. Zum Beispiel beziehen agile Teams, die Scrum folgen, oft XP Praktiken mit ein.

Scrum

Scrum, u. a. von Mike Beedle und Ken Schwaber [Schwaber01] eingeführt, ist ein agiles Managementframework und weist folgende wesentliche Projektmanagementinstrumente und Praktiken auf [Linz14]:

- Sprint: Scrum gliedert ein Projekt in kurze Iterationen fester Länge. Eine solche Iteration heißt in Scrum „Sprint“. Sie dauert in der Regel zwei bis vier Wochen.
- Produkterweiterung (Inkrement): Jeder Sprint soll ein potenziell auslieferbares Produkt erzeugen, dessen Leistungsumfang mit jeder Iteration wächst.
- Product-Backlog: Der Product-Owner führt ein sog. Product-Backlog. Es enthält eine priorisierte Auflistung der geplanten Produktfeatures. Das Product-Backlog entwickelt und verändert sich über die Sprints hinweg. Dieses Arbeiten am Backlog wird auch „Backlog Refinement“ genannt.
- Sprint-Backlog: Zu Beginn eines jeden Sprints zieht das Team diejenigen Anforderungen, die im priorisierten Product-Backlog an der Spitze stehen und die es in diesem Sprint umsetzen will, aus dem Product-Backlog in ein kleineres Sprint-Backlog. Da nicht der Product-Owner, sondern das Scrum-Team die Anforderungen auswählt, die in diesem Sprint umgesetzt werden, bezeichnet man die Auswahl als Auswahl nach dem Pull-Prinzip (im Gegensatz zum Push-Prinzip).
- Definition-of-Done: Um abzusichern, dass am Sprint-Ende tatsächlich ein fertiges Produktinkrement vorliegen wird, formuliert das Team zu Beginn eines Inkrementes gemeinsam Kriterien, anhand derer es überprüfen und entscheiden kann, ob die Arbeit an dem Inkrement abgeschlossen ist. Die gemeinsame Diskussion über die Definition-of-Done trägt ganz wesentlich dazu bei, den Inhalt einer Anforderung oder einer Aufgabe zu klären und im Team ein gemeinsames, gleiches Verständnis über jede Aufgabe zu erhalten.
- Timeboxing: Nur solche Aufgaben, Anforderungen oder Features, die das Team erwartungsgemäß innerhalb des Sprints fertigstellen kann, werden in das Sprint-Backlog aufgenommen. Wenn Aufgaben während eines Sprints nicht fertiggestellt werden können, werden die zugehörigen Produktfeatures aus dem Sprint entfernt und die Aufgabe wandert wieder in das Product-Backlog. Timeboxing wird in Scrum nicht nur auf Ebene der Sprints angewendet, sondern in vielen Situationen, in denen es darum geht, fertig zu werden. So ist

Timeboxing ein nützliches Instrument, um z. B. in Meetings einen pünktlichen Beginn und strikte Einhaltung des geplanten Endzeitpunkts durchzusetzen.
- Transparenz: Der Sprint-Status wird täglich (im Daily Scrum, der täglichen Statusrunde des Teams, „Stand-up“) aktualisiert und abgebildet. Dadurch sind der Inhalt und Fortschritt des aktuellen Sprints, einschließlich der Testergebnisse, für das Team, das Management und alle interessierten Parteien gleichermaßen offensichtlich. Beispielsweise kann das Entwicklungsteam den Sprintstatus auf einem Scrum-Board präsentieren.

Scrum definiert drei Rollen:

- Scrum-Master: Er ist verantwortlich dafür, dass die Scrum Praktiken umgesetzt und verfolgt werden. Wenn sie verletzt werden, Personalfragen auftreten oder andere praktische Hindernisse (Impediments) auftreten, ist es Aufgabe des Scrum-Master, diese abzustellen bzw. eine Lösung herbeizuführen. Der Scrum-Master hat allerdings keine Teamleitungsfunktion, sondern er agiert als Coach.
- Product-Owner: Der Product-Owner ist die Person, die das Product-Backlog verantwortet, führt und priorisiert. Er agiert gegenüber dem Team als Vertreter des oder der Kunden. Diese Person hat keine Teamleitungsfunktion, er verantwortet die Produkteigenschaften!
- Entwicklungsteam: Das Entwicklungsteam entwickelt und testet das Produkt. Es ist selbstorganisiert: Es gibt keine Teamleitung, das Team als Ganzes trifft alle Entscheidungen. Das Team arbeitet funktionsübergreifend zusammen (siehe Abschnitt 2.3.2 und Abschnitt 3.1.4).

Im Gegensatz zu XP regelt Scrum nicht, welche Softwareentwicklungstechniken (wie beispielsweise testgetriebene Entwicklung) einzusetzen sind, um Software zu erstellen. Darüber hinaus liefert Scrum auch keine Richtlinien darüber, wie Tests einzusetzen sind.

Kanban

Kanban [Anderson13] ist ein Managementansatz, der gelegentlich in agilen Projekten genutzt wird. Das allgemeine Ziel ist es, den Arbeitsfluss innerhalb einer Wertschöpfungskette abzubilden und zu optimieren. Kanban verwendet dazu drei Instrumente [Linz14]:

- Kanban-Board: Die zu steuernde Wertschöpfungskette wird auf einem sogenannten Kanban-Board visualisiert. Jede Spalte repräsentiert eine Bearbeitungsstation, d.h. eine Menge von zusammengehörigen Aktivitäten, z. B. Entwicklung oder Test. Die zu erledigenden Aufgaben (Tasks) werden durch Karten (Tickets) symbolisiert, die auf dem Board von links nach rechts wandern.
- Work-In-Progress-Limit: Die Menge der gleichzeitig zu erledigenden Aufgaben (Work-in-Progress, WIP) wird limitiert. Dies geschieht durch Limits für die Anzahl der Tickets, die je Bearbeitungsstation und/oder im gesamten Board erlaubt sind. Hat eine Bearbeitungsstation freie Kapazität, dann zieht sich diese Station ein neues Ticket von ihrer Vorgängerstation.
- Lead-Time: Kanban wird genutzt, um durch die Senkung der durchschnittlichen Bearbeitungszeit den kontinuierlichen Fluss von Aufgaben durch die gesamte Wertschöpfungskette zu optimieren.

Dieses Vorgehen ist Scrum sehr ähnlich. In beiden Ansätzen sorgt die Visualisierung an Boards für hohe Transparenz über Inhalt und Bearbeitungsstand aller Aufgaben. Aufgaben, die noch nicht terminiert sind, warten im Backlog und werden erst dann ins Kanban Board gezogen, wenn neuer Platz (neue Produktionskapazität) entstanden ist.

Iterationen oder Sprints sind in Kanban optional. Der Kanban Prozess ermöglicht die Auslieferung von Arbeitsergebnissen Stück für Stück, statt nur als Bestandteil eines Release. Timeboxing als Synchronisationsmechanismus ist daher hier ebenfalls optional, im Gegensatz zu Scrum, bei dem alle Arbeitsaufträge und deren Ergebnisse innerhalb eines Sprints synchronisiert werden müssen.

1.2.2 Kollaborative Erstellung von User-Stories

Die schlechte Qualität von Spezifikationen ist oft ein Grund für das Fehlschlagen eines Projektes. Ursachen können der fehlende Überblick des Kunden über seine tatsächlichen Bedürfnisse, das Fehlen einer globalen Vision für das System, redundante oder sich widersprechende Anforderungen oder andere Fehler in der Kommunikation sein.

Um solche Fehlerquellen zu vermeiden, werden in agilen Entwicklungen User-Stories eingesetzt. Diese beschreiben die Anforderungen aus Sicht der Fachbereichsvertreter, aber auch der Entwickler und Tester und werden von diesen gemeinsam verfasst. In einer sequentiellen Entwicklung muss eine solche gemeinsame Sicht auf das System bzw. seine Leistungsmerkmale durch formale Reviews nach Fertigstellung der Anforderungsbeschreibung erreicht werden. Im agilen Umfeld wird dies durch häufige informelle Reviews während der Phase der Anforderungsbeschreibung erreicht.

Die User-Stories müssen sowohl funktionale als auch nicht-funktionale Eigenschaften behandeln. Jede Story soll die Abnahmekriterien für diese Eigenschaften enthalten. Auch die Kriterien sollten in Zusammenarbeit zwischen Fachbereichsvertretern, Entwicklern und Testern definiert werden. Sie bieten Entwicklern und Testern eine erweiterte Sicht auf das Feature, das die Fachbereichsvertreter bewerten werden. Ein Agile Team sieht eine Aufgabe als erledigt an, wenn eine Menge von Abnahmekriterien erfüllt ist.

Typischerweise verbessert der Blickwinkel des Testers die User-Story, indem er fehlende Details oder nicht-funktionale Anforderungen identifiziert und ergänzt. Ein Tester kann seinen Beitrag auch leisten, indem er den Vertretern des Fachbereichs offene Fragen über die User-Story stellt und Methoden vorschlägt, die User-Story zu testen und die Abnahmekriterien zu bestätigen.

Für die Zusammenarbeit bei der Erstellung der User-Story können Techniken wie z. B. Brainstorming oder Mind-Mapping genutzt werden.

Gemäß dem 3C-Konzept [Jeffries00] ist eine User-Story die Verbindung der folgenden drei Elemente:

- Card: Die Karte (Card) ist das physische Medium, das die User-Story beschreibt. Hier werden die Anforderung, ihre Dringlichkeit, die erwartete Dauer für Entwicklung und Test sowie die Abnahmekriterien für diese Story identifiziert. Die Beschreibung muss genau sein, da sie im Product-Backlog verwendet wird. Ein Beispiel für eine Anforderung aus einer User-Story ist das folgende: „Als Kunde möchte ich, dass ein [neues Pop-Up] Fenster [mit der Erklärung, wie das Registrierungsformular auszufüllen ist] erscheint, sodass ich mich [für die Konferenz] registrieren kann [, ohne Felder zu vergessen].”
- Conversation: Die Diskussion (Conversation) erklärt, wie die Software genutzt werden wird. Sie kann dokumentiert werden oder verbal bleiben, beginnt während der Releaseplanungsphase und wird fortgeführt, wenn die Umsetzung der User-Story zeitlich geplant wird.
- Confirmation: Die Abnahmekriterien, die in der Diskussion festgelegt werden, werden verwendet, um den Abschluss einer User-Story bestätigen (confirm) zu können. Diese Abnahmekriterien können sich über mehrere User-Stories erstrecken. Es sollten sowohl positive als auch negative Aspekte getestet werden, um die Kriterien abzudecken. Während der Bestätigung (Confirmation) nehmen verschiedene Teilnehmer die Rolle des Testers ein. Das können sowohl Entwickler als auch Experten sein, die auf Performanz, Sicherheit, Interoperabilität und andere Qualitätsmerkmale spezialisiert sind. Um eine Story als abgeschlossen zu bestätigen, müssen die definierten Abnahmekriterien getestet sein und als erfüllt eingestuft werden.

Agile Teams unterscheiden sich im Hinblick darauf, wie sie User-Stories dokumentieren. Unabhängig vom Ansatz sollte die Dokumentation präzise, ausreichend und notwendig sein.

Um die Qualität einer User-Story zu beurteilen, kann der Tester die sogenannten INVEST-Kriterien [INVEST] heranziehen: Die User-Story ist

- unabhängig (independent) von anderen User-Stories,
- verhandelbar (negotiable), d.h. bietet noch Gestaltungsspielraum, der im Team gemeinsam verhandelt wird,
- nützlich (valuable), d.h. der Nutzen ist erkennbar,
- so beschrieben und vom Team verstanden, dass sie schätzbar (estimable) ist,
- von angemessener Größe (small) – zu große User-Stories werden heruntergebrochen,
- testbar (testable), z. B. dadurch, dass Abnahmekriterien beschrieben sind.

1.2.3 Retrospektiven

In der agilen Entwicklung bezeichnet eine Retrospektive eine Team-Sitzung am Ende jeder Iteration, in der besprochen wird, was erfolgreich war, was verbessert werden könnte und wie die Verbesserungen in künftigen Iterationen umgesetzt werden können. Dabei wird auch ein Augenmerk auf die Erhaltung gut funktionierender Praktiken gesetzt. Retrospektiven decken Themen wie den Prozess, die beteiligten Personen, Organisationen und Beziehungen sowie die Werkzeuge ab.

Ergebnisse der Retrospektiven können testverbessernde Entscheidungen über Maßnahmen sein, die sich auf die Testeffektivität, die Testproduktivität die Testfallqualität und die Teamzufriedenheit konzentrieren. Sie können auch auf die Prüfbarkeit der Anwendungen, User-Stories, Features oder Systemschnittstellen abzielen. Im Allgemeinen sollten sich Teams nur einige wenige Verbesserungen pro Iteration vornehmen. Dies erhöht die Realisierungschancen und ermöglicht eine kontinuierliche Verbesserung in gleichbleibender Geschwindigkeit.

Wenn die identifizierten Maßnahmen wie z. B. Grundursachenanalysen nachverfolgt und umgesetzt werden, dann leisten diese regelmäßig abgehaltenen Retrospektiven einen entscheidenden Beitrag zur Selbstorganisation des Teams und zur kontinuierlichen Verbesserung von Entwicklung und Test.

Die zeitliche Planung und Organisation der Retrospektiven hängt vom jeweiligen agilen Ansatz ab, die angewendet wird. Fachbereichsvertreter und das Team nehmen an jeder Retrospektive teil, während der Moderator sie organisiert und den Ablauf steuert. In einigen Fällen kann das Team andere Teilnehmer zum Treffen einladen.

Die Tester sollten in den Retrospektiven eine wichtige Rolle spielen. Tester sind Teil des Teams und bringen ihre einzigartige Sichtweise ein [ISTQB_FL_SYL], Abschnitt 1.5. Das Testen erfolgt in jedem Sprint und trägt entscheidend zum Erfolg bei. Alle Teammitglieder, Tester und Nicht-Tester können sowohl zu den Testaktivitäten als auch zu den Nicht-Testaktivitäten einen Beitrag leisten.

Retrospektiven müssen in einer professionellen Umgebung stattfinden, die durch gegenseitiges Vertrauen gekennzeichnet ist. Die Kennzeichen einer erfolgreichen Retrospektive sind die gleichen wie die jedes anderen Reviews wie im Lehrplan zum Foundation Level [ISTQB_FL_SYL, in Abschnitt 3.2] beschrieben.

1.2.4 Kontinuierliche Integration (continuous integration)

Die Auslieferung einer Produkterweiterung (Inkrement) setzt voraus, dass am Ende einer jeden Iteration bzw. eines jeden Sprints eine funktionierende Software vorliegt. Das sicherzustellen ist eine große Herausforderung und die Lösung dafür ist das Verfahren der kontinuierlichen Integration (continuous integration), das alle geänderten Softwarekomponenten regelmäßig und mindestens einmal pro Tag zusammenführt (Build erstellen): Konfigurationsmanagement, Kompilierung, Buildprozess, Verteilung in die Zielumgebung und die Ausführung der Tests sind in einem automatisierten, wiederholbaren Prozess zusammengefasst.

Da die Entwickler ihre Arbeit kontinuierlich integrieren, und kontinuierlich Builds erstellen und diese sofort (automatisiert) testen, werden Fehler im Code schneller entdeckt.

Der Prozess der kontinuierlichen Integration (continuous integration) besteht aus folgenden automatisierten Aktivitäten:

- Statische Codeanalyse: Durchführung einer statischen Codeanalyse und Aufzeichnung der Ergebnisse,
- Kompilieren: Kompilieren und Linken des Codes, Erstellung der ausführbaren Dateien,
- Unittest: Durchführen der Unittests, Prüfung der Codeabdeckung und Aufzeichnung der Testergebnisse,
- Bereitstellung (Deployment): Installieren der Software in eine Testumgebung, Integrationstest: Durchführung der entsprechenden Tests und Aufzeichnung der Ergebnisse,
- Bericht (Dashboard): Veröffentlichung des Status all dieser Aktivitäten an einem öffentlich sichtbaren Ort.
- Ein automatisierter Build- und Testprozess findet täglich statt und erkennt Integrationsfehler frühzeitig und schnell. Kontinuierliche Integration (continuous integration) ermöglicht es agilen Testern, regelmäßig automatisierte Tests durchzuführen – in einigen Fällen als Teil des kontinuierlichen Integrationsprozesses selbst – um schnelle Rückmeldungen über die Codequalität an das Team zu geben. Testergebnisse sind für alle Teammitglieder sichtbar, speziell, wenn automatisch erstellte Berichte in den Prozess integriert sind. Regressionstests können während der gesamten Iteration kontinuierlich durchgeführt werden. Dies kann, wo anwendbar, auch das schrittweise Integrieren eines großen Systems beinhalten. Gute automatisierte Regressionstests decken so viele Funktionalitäten wie möglich ab, darunter auch die User-Stories, die in vorangegangenen Iterationen geliefert wurden. Dies verschafft agilen Testern Freiraum für manuelle Tests, die sich auf neue Features, Änderungen und Fehlernachtests konzentrieren.

Zusätzlich zu automatisierten Tests verwenden Organisationen, die kontinuierliche Integration (continuous integration) nutzen, typischerweise Build-Werkzeuge, um eine kontinuierliche Qualitätskontrolle sicherzustellen. Zusätzlich zur Durchführung von Unit- und Integrationstests können solche Werkzeuge weitere statische und dynamische Test durchführen, Performanz messen und Performanceprofile erstellen, Dokumentation aus dem Quellcode herausziehen und formatieren sowie manuelle Qualitätssicherungsprozesse vereinfachen. Diese kontinuierliche Anwendung von Qualitätskontrollen zielt auf die Verbesserung der Qualität des Produkts sowie auf die Reduzierung der Zeit bis zur Lieferung ab. Im Vergleich dazu wird in traditionellen Vorgehensweisen eher erst nach Fertigstellung der Entwicklung die abschließende Qualitätskontrolle durchgeführt.

Build-Werkzeuge können mit automatischen Deployment-Werkzeugen verbunden werden, so dass ein entsprechender Build vom Continuous-Integration- oder Build-Server gezogen und in eine oder mehrere Umgebungen (Entwicklung, Test, QS, Produktion) überspielt (deployed) werden kann. Dies reduziert Fehler und Verzögerungen, die bei manueller Installation entstehen können.

Kontinuierliche Integration (continuous integration) kann die folgenden Vorteile haben:

- Sie ermöglicht früheres Erkennen und einfachere Grundursachenanalyse von Integrations-problemen und widersprüchlichen Änderungen.
- Sie gibt dem Entwicklungsteam regelmäßige Rückmeldungen darüber, ob der Code funktioniert.
- Die im Test befindliche Version ist höchstens einen Tag älter als die aktuelle Entwicklungsversion.

- Sie vermindert Regressionsrisiken, die mit dem (Refactoring des Codes durch die Entwickler verbunden sind, indem zügig Fehlernachtests und Regressionstests nach jedem kleinen Set an Änderungen stattfinden.
- Sie liefert die Bestätigung, dass die Entwicklungsarbeit jedes Tages solide ist.
- Sie macht den Fortschritt in Richtung der Fertigstellung einer Produkterweiterung (Inkrements) sichtbar, was Entwickler und Tester ermutigt.
- Sie beseitigt die zeitplanerischen Risiken, die mit einer Big-Bang-Integration verbunden sind.
- Sie liefert aktuellste Versionen ausführbarer Software über den gesamten Sprint hinweg, für Test-, Demonstrations- oder Schulungszwecke.
- Sie vermindert sich wiederholende, manuelle Testaktivitäten.
- Sie liefert schnelle Rückmeldungen über Entscheidungen, die getroffen wurden, z. B. um Qualität und Tests zu verbessern.

Kontinuierliche Integration (continuous integration) hat jedoch auch ihre Risiken und Herausforderungen:

- Werkzeuge für die kontinuierliche Integration (continuous integration) müssen eingeführt und gepflegt werden.
- Der Prozess der kontinuierlichen Integration muss aufgesetzt und etabliert werden
- Die nötige Testautomatisierung erfordert zusätzliche Ressourcen im Team und es kann komplex sein, eine solche Testautomatisierung aufzubauen.
- Eine möglichst umfassende Testüberdeckung ist essenziell, um die Vorteile der automatisierten Tests zu nutzen.
- Teams verlassen sich manchmal zu sehr auf Unittests und setzen Integrations-, System- und Abnahmetests in zu geringem Maß ein.

Der Einsatz von kontinuierlicher Integration (continuous integration) erfordert die Verwendung von verschiedenen Werkzeugen wie Testwerkzeugen, Werkzeugen zur Automatisierung des Buildprozesses und Werkzeugen zur Versionskontrolle.

1.2.5 Release- und Iterationsplanung

Wie bereits im Lehrplan zum Foundation Level [ISTQB_FL_SYL] erwähnt wurde, ist die Planung in einem Softwareprojekt eine fortlaufende Aktivität. Dies gilt auch für den agilen Lebenszyklus. Für agile Lebenszyklen gibt es zwei Arten der Planung, die Releaseplanung und die Iterationsplanung.

Releaseplanung

Die Releaseplanung schaut voraus auf die Veröffentlichung (Release) einer Produktversion, die oft einige Monate vom Beginn des Projekts entfernt in der Zukunft liegt. In der Releaseplanung wird das Product-Backlog erstellt und/oder aktualisiert. In ihr können größere User-Stories in eine Sammlung kleinerer Stories heruntergebrochen werden. Sie liefert die Basis für die Testvorgehensweise und Planung der Testaktivitäten für alle Iterationen. Releasepläne sind Pläne auf grobgranularem Niveau.

In der Releaseplanung wählen Vertreter des Fachbereichs die User-Stories für das betreffende Release aus und priorisieren sie in Zusammenarbeit mit dem Team (siehe Abschnitt 1.2.2). Basierend auf diesen User-Stories werden die Projekt- und Qualitätsrisiken identifiziert und es wird eine grobgranulare Aufwandsschätzung vorgenommen (siehe Abschnitt 3.2).

Tester sind an der Releaseplanung beteiligt und leisten insbesondere mit folgenden Aktivitäten einen wichtigen Beitrag:

- Definieren testbarer User-Stories, inklusive Abnahmekriterien
- Teilnehmen an Projekt- und Qualitätsrisikoanalyse

- Schätzen des Testaufwandes in Zusammenhang mit den User-Stories
- Definieren der notwendigen Teststufen
- Planen der Tests für das Release

Nach der Fertigstellung der Releaseplanung beginnt die Iterationsplanung für die erste Iteration. Die Iterationsplanung schaut voraus auf das Ende einer einzelnen Iteration und erstellt das Iteration-Backlog.

Iterationsplanung

In der Iterationsplanung wählt („pulled“) das Team User-Stories aus dem priorisierten Release-Backlog, detailliert sie, nimmt eine Risikoanalyse vor und schätzt die Arbeit, die für jede User-Story benötigt wird. Wenn eine User-Story zu ungenau ist und Versuche der Klärung gescheitert sind, kann das Team sie ablehnen und zur nächsten User-Story aus der priorisierten Liste übergehen. Die Vertreter des Fachbereichs sind verantwortlich für die Beantwortung der Fragen des Teams zu jeder Story, sodass das Team verstehen kann, was implementiert werden soll und wie jede einzelne Story zu testen ist.

Die Anzahl der ausgewählten User-Stories hängt von der Velocity (Maß für Produktivität im agilen Projekt) ab, mit der das Team arbeitet und von der geschätzten Größe der ausgewählten User-Stories. Nachdem der Inhalt der Iteration definiert ist, werden die User-Stories in Aufgaben heruntergebrochen, die von den passenden Teammitgliedern übernommen werden.

Tester werden in die Iterationsplanung einbezogen und erbringen einen besonderen Mehrwert in folgenden Aktivitäten:

- Teilnehmen an der detaillierten Risikoanalyse der User-Stories
- Festlegen der Testbarkeit der User-Stories
- Erstellen der Abnahmetests für die User-Stories
- Herunterbrechen der User-Stories in Aufgaben (insbesondere Testaufgaben)
- Schätzen des Testaufwands für alle Testaufgaben
- Identifizieren funktionaler und nicht-funktionaler Eigenschaften des zu testenden Systems
- Unterstützen von und Mitarbeit an der Testautomatisierung

Releasepläne können sich im Laufe des Projektfortschritts ändern. Darunter fallen auch Änderungen individueller User-Stories im Product-Backlog. Diese Änderungen können durch interne oder externe Faktoren hervorgerufen werden. Unter internen Faktoren versteht man Liefermöglichkeiten, Velocity und technische Schwierigkeiten. Unter externen Faktoren versteht man die Entdeckung von neuen Märkten und Möglichkeiten, neue Mitbewerber, oder Geschäftsrisiken, die Ziele und/oder Zieldaten verändern. Darüber hinaus können sich Iterationspläne während einer Iteration verändern. Zum Beispiel kann sich eine bestimmte User-Story, die während der Schätzung als relativ einfach eingeschätzt wurde, als komplexer herausstellen als angenommen.

Diese Änderungen können eine Herausforderung für Tester sein. Tester müssen das große Ganze des Release für Testplanungszwecke verstehen und sie benötigen eine angemessene Testbasis und ein angemessenes Testorakel in jeder Iteration zu Testentwicklungszwecken wie im Lehrplan zum Foundation Level [ISTQB_FL_SYL] Abschnitt 1.4 ausgeführt.

Die benötigte Information muss dem Tester früh zur Verfügung stehen und dennoch müssen Änderungen gemäß den agilen Prinzipien willkommen geheißen werden. Dieses Dilemma erfordert vorsichtige Entscheidungen über Teststrategien und Testdokumentation (vgl. [Black09] Kapitel 12).

Die Release- und Iterationsplanung soll sowohl die Planung für die Entwicklungs- wie auch die Testaktivitäten adressieren. Unter letzteres fallen:

- Testumfang und -intensität in den zu testenden Bereichen, Testziele und Gründe, die zu diesen Entscheidungen führten
- Teammitglieder, die die Testaktivitäten durchführen
- Testumgebung und -daten, die benötigt werden, zu welchem Zeitpunkt sie benötigt werden und welche Änderungen diese vor oder während der Projektlaufzeit erfahren
- Zeitliche Abfolge, Reihenfolge, Abhängigkeiten und Vorbedingungen für funktionale und nicht-funktionale Tests (z. B. wie häufig werden Regressionstests durchgeführt, welche Features hängen voneinander oder von bestimmten Daten ab). Darunter fällt auch wie die Testaktivitäten mit den Entwicklungsaktivitäten in Zusammenhang stehen bzw. von diesen abhängig sind.
- Projekt- und Produkt- (Qualitäts-)Risiken die zu adressieren sind (siehe Abschnitt 3.2.1)

Zusätzlich sollten größere Teamschätzungen Überlegungen mit einschließen, die die Dauer und den Aufwand zur Umsetzung der erforderlichen Testaktivitäten beinhalten.

2. Grundlegende Prinzipien, Praktiken und Prozesse des agilen Testens – 105 min

Schlüsselwörter

Build-Verifizierungstest, Konfigurationsobjekt, Konfigurationsmanagement

Lernziele für grundlegende Prinzipien, Praktiken und Prozesse des agilen Testens

2.1 Die Unterschiede zwischen Tests in traditionellen und agilen Ansätzen

FA-2.1.1 (K2) Die Unterschiede der Testaktivitäten zwischen agilen und nicht-agilen Projekten benennen und erläutern können.
FA-2.1.2 (K2) Beschreiben können, wie Entwicklungs- und Testaktivitäten in einem agilen Projekt umgesetzt werden.
FA-2.1.3 (K2) Die Bedeutung von unabhängigem Test in agilen Projekten darlegen können.

2.2 Status des Testens in agilen Projekten

FA-2.2.1 (K2) Erläutern können, welches Mindestmaß an Arbeitsergebnissen sinnvoll ist, um den Testfortschritt und die Produktqualität in agilen Projekten sichtbar zu machen.
FA-2.2.2 (K2) Damit vertraut sein, dass sich die Tests über mehrere Iterationen hinweg kontinuierlich weiterentwickeln und daher auch erklären können, warum Testautomatisierung wichtig ist, zum Beherrschen der Risiken im Regressionstest.

2.3 Die Rolle und die Fähigkeiten eines Testers in einem agilen Team

FA-2.3.1 (K2) Verstehen, über welche Fähigkeiten (bzgl. Menschen, Domainwissen und Testen) ein Tester in agilen Teams verfügen muss.
FA-2.3.2 (K2) Wissen, was die Rolle eines Testers in einem agilen Team ist.

2.1 Die Unterschiede zwischen traditionellen und agilen Ansätzen im Test

Wie im Foundation Level-Lehrplan [ISTQB_FL_SYL] beschrieben, stehen Testaktivitäten in engem Zusammenhang zu den Entwicklungsaktivitäten und unterscheiden sich daher je nach den unterschiedlichen Phasen im Produktlebenszyklus voneinander. Tester müssen daher den Unterschied zwischen dem Testen in traditionellen Lebenszyklusmodellen (z. B. sequentiellen wie beispielsweise dem V-Modell oder iterativen wie beispielsweise dem RUP) und dem Testen in agilen Ansätze verstehen, um effektiv und effizient arbeiten zu können. Die agilen Modelle unterscheiden sich von den traditionellen Modellen u. a. in folgenden Bereichen:

- bezüglich der Art und Weise, wie Test- und Entwicklungsaktivitäten in das Vorgehen integriert werden
- bezüglich der Arbeitsergebnisse in einem Projekt
- bezüglich der verwendeten Begrifflichkeiten
- bezüglich der Eingangs- und Endekriterien, die für die verschiedenen Teststufen verwendet werden
- bezüglich des Gebrauchs und Einsatzes von Werkzeugen
- bezüglich dessen, wie unabhängiges Testen effektiv umgesetzt werden kann.

Tester sollten wissen, dass sich die Implementierung der agilen Ansätze zwischen Unternehmen bzw. anderen Organisationen jeweils erheblich unterscheiden können. Gut begründbare und durchdachte Abweichungen von den Idealen der agilen Ansätze (siehe Abschnitt 1.1) können eine sinnvolle und zielführende Anpassung an die Kundenwünsche sein. Die Fähigkeit zur Anpassung an den Kontext eines bestimmten Projektes, also auch an das jeweils verwendete Softwareentwicklungsvorgehen, ist ein Schlüsselfaktor für den Erfolg der Tester [Baumgartner13].

2.1.1 Test- und Entwicklungsaktivitäten

Einer der Hauptunterschiede zwischen traditionellen und agilen Ansätzen ist der Gedanke von sehr kurzen Iterationen. Jede Iteration sollte Software hervorbringen, deren Features für die Stakeholder von Wert sind und funktioniert. Am Anfang eines Projektes gibt es eine Phase der Releaseplanung. Auf diese folgt eine Abfolge von Iterationen. Zu Beginn jeder dieser Iterationen gibt es eine Phase der Iterationsplanung. Sobald der Umfang der Iteration festgelegt ist, werden die ausgewählten User-Stories entwickelt, im System integriert und getestet. Diese Iterationen sind in hohem Maße dynamisch, weil Entwicklungs-, Integrations- und Testaktivitäten über die gesamte Dauer der Iteration hinweg parallel stattfinden und somit erhebliche Überlappungen aufweisen können. Testaktivitäten finden schon während der gesamten Iteration statt und nicht erst als abschließende Aktivität.

Wie auch in traditionellen Modellen haben Tester, Entwickler und Fachbereichsvertreter eine wichtige Funktion im Testen. Entwickler führen Unittests durch, während sie Features aus den User-Stories entwickeln. Tester testen im Anschluss diese Features. Product-Owner überprüfen und bewerten die ganzen Stories ebenfalls schon im Rahmen der Implementierung. Product-Owner können schriftliche Testfälle verwenden, sie können aber auch einfach das Feature nutzen und damit experimentieren, um so ein schnelles Feedback an das Entwicklungsteam geben zu können.

In einigen Fällen finden regelmäßige Stabilisierungsiterationen (sog. „hardening iterations“) statt, die noch nicht behobene Fehler und andere Arten technischer Schwierigkeiten (technische „Schulden“, also Qualitätskompromisse, die eingegangen wurden um z. B. eine rasche „time to market“ zu realisieren) beseitigen sollen. Ein Feature gilt erst dann als erledigt, wenn es im System integriert und im System getestet worden ist. Auch hat sich bewährt, Fehler, die aus der letzten Iteration übrig geblieben sind, direkt zu Beginn der nächsten Iteration als Teil des Überhangs aus dieser Iteration anzugehen (auch bezeichnet als „fix bugs first“). Andere wiederum sind der Ansicht, dass so ein Vorgehen den verbleibenden Aufwand in der Iteration verschleiert. Damit wäre es folglich schwieriger

abzuschätzen, wann die übrigen Features erledigt werden können. Auch können, um nach einer Reihe von Iterationen ein Release auszuliefern, zusätzliche Arbeiten nötig sein.

Dort wo risikoorientiertes Testen Teil der Teststrategie ist, findet während der Releaseplanung schon eine grobe Risikoanalyse statt, in der Tester häufig auch eine führende Rolle übernehmen. Die spezifischen Qualitätsrisiken, die mit jeder Iteration verbunden sind, werden jedoch in der Iterationsplanung identifiziert und im Detail bewertet. Diese Risikoanalyse kann sowohl die Reihenfolge der Feature in der Entwicklung als auch die Priorität und Tiefe des Testens je Feature beeinflussen. Sie beeinflusst außerdem die Schätzung der Aufwände für die erforderlichen Tests pro Feature.

In einigen agilen Ansätzen (z. B. Extreme Programming) kommt das sog. „Pairing“ (paarweises Zusammenarbeiten) zum Einsatz. Es können zum Beispiel zwei Tester zusammen am Test eines Features arbeiten. Pairing kann aber auch bedeuten, dass ein Tester mit einem Entwickler zusammenarbeitet, um ein Feature zu entwickeln und zu testen. Pairing kann schwierig werden, wenn das Testteam nicht an einem Ort zusammen ist. Aber es gibt auch für diese Situation Vorgehensweisen und Werkzeuge, um Pairing zu ermöglichen. Für mehr Informationen zu den Schwierigkeiten, die mit verteilten Teams verbunden sind, siehe [ISTQB_ALTM_SYL], Abschnitt 2.8.

Tester können innerhalb eines Teams auch die Rolle des Test- und Qualitätstrainers einnehmen, indem sie Testwissen verbreiten und die Qualitätssicherungsaufgaben innerhalb des Teams unterstützen. Diese Vorgehensweise fördert das Verständnis für eine gemeinsame Verantwortung für Qualität und Test.

Testautomatisierung findet in vielen agilen Teams auf allen Teststufen statt. Dies kann bedeuten, dass Tester einige Zeit damit verbringen, automatisierte Tests zu erstellen, auszuführen, zu überwachen und zu pflegen. Der verbreitete Einsatz der Testautomatisierung führt dazu, dass ein größerer Teil des manuellen Testens in agilen Projekten mit Hilfe von erfahrungsbasierten und fehlerbasierten Techniken erfolgt, wie angriffsbasiertes Testen, exploratives Testen und Error Guessing (intuitive Testfallermittlung) (siehe [ISTQB_ALTA_SYL], Abschnitte 3.3 und 3.4, sowie [ISTQB_FL_SYL], Abschnitt 4.5). Während sich die Entwickler auf die Erstellung von Unittests konzentrieren, sollten Tester sich auf die Erstellung der automatisierten Integration, der System- und Systemintegrationstests konzentrieren. Das führt dazu, dass in agilen Teams vorzugsweise Tester mit einem soliden technischen und Testautomatisierungshintergrund gesucht werden.

Ein agiles Grundprinzip lautet, dass "Änderungen willkommen sind" und während des gesamten Projektes auftreten dürfen. Daher wird eine leichtgewichtige Dokumentation in agilen Projekten bevorzugt. Änderungen an bestehenden Features haben natürlich Auswirkungen auf das Testen, insbesondere auf die Regressionstests. Testautomatisierung ist eine Möglichkeit, dem aus häufigen Änderungen resultierenden erhöhten Testbedarf zu begegnen. Es ist jedoch wichtig, dass die Änderungsrate die Fähigkeiten des Teams nicht überfordert, mit den damit verbundenen Risiken umzugehen.

2.1.2 Arbeitsergebnisse des Projekts

Arbeitsergebnisse des Projekts, die direkt für Tester wichtig sind lassen sich üblicherweise in folgende drei Kategorien einteilen:

1. Geschäftsprozessorientierte Arbeitsergebnisse, die beschreiben, was die Software leisten soll (z. B. Anforderungsspezifikationen) und wie sie benutzt werden soll (z. B. Benutzerdokumentation)
2. Entwicklungsorientierte Arbeitsergebnisse, die beschreiben, wie das System aufgebaut ist (z. B. Datenbank, ER Diagramme), die das System implementieren (z. B. Code) oder die das System konfigurieren (z. B. Installations-Skripte, Konfigurationsdateien)

3. Testorientierte Arbeitsergebnisse, die beschreiben, wie das System getestet wird (z. B. Teststrategien und Pläne), die das System schließlich auch testen (z. B. manuelle und automatisierte Tests) oder die Testergebnisse darstellen (z. B. Test-Dashboards)

In einem typischen agilen Projekt wird Wert darauf gelegt, die Menge an Dokumentation, die erstellt und gepflegt wird, zu optimieren („just enough documentation"; Vermeidung von Dokumentation ohne Mehrwert). Funktionierende Software sowie automatisierte Tests, die die Erfüllung von Anforderungen nachweisen, haben daher höhere Priorität. In einem erfolgreichen agilen Projekt wird ein Gleichgewicht angestrebt, zwischen einerseits der Steigerung der Effizienz aufgrund reduzierter Dokumentation und andererseits ausreichender Dokumentation, um Unternehmens-, Test-, Entwicklungs- und Wartungsaktivitäten zu unterstützen. Das Team muss während der Releaseplanung eine Entscheidung darüber treffen, welche Arbeitsergebnisse notwendig sind und bis zu welchem Grad eine Dokumentation der Arbeitsergebnisse erforderlich ist.

Typische geschäftsprozessorientierte Arbeitsergebnisse in agilen Projekten sind User-Stories und Abnahmekriterien. User-Stories sind die agile Form der Anforderungsbeschreibung und sollten erklären, wie sich das System in Bezug auf ein einzelnes verständliches Feature oder eine Funktion verhalten soll. Eine User-Story sollte ein Feature beschreiben, das klein genug ist, um es im Rahmen einer einzigen Iteration fertigzustellen. Größere Ansammlungen von untereinander verbundenen Features oder eine Sammlung von Unter-Features, die zu einem einzigen, komplexen Feature zusammengeführt werden, werden als „Epic" bezeichnet. Epics können User-Stories unterschiedlicher Entwicklungsteams beinhalten. Zum Beispiel kann eine User-Story die Anforderungen auf der API-Stufe (Middleware) beschreiben während eine andere User-Story beschreibt, was auf Benutzerebene (Anwendung) gebraucht wird. Diese Sammlungen können über eine Reihe von Iterationen hinweg entwickelt werden. Jede Epic und ihre User-Stories sollten jeweils zugehörige Abnahmekriterien haben.

Typische Arbeitsergebnisse von Entwicklern beinhalten natürlich auch in agilen Projekten Code. Agile Entwickler erstellen allerdings oft automatisierte Unittests. Diese Tests können nach der Kodierung erstellt werden. In einigen Fällen erstellen Entwickler diese Tests jedoch inkrementell und vor der Entwicklung des eigentlichen Codes. Damit haben sie die Möglichkeit, direkt nach Schreiben dieses Teils des Codes automatisiert zu überprüfen, ob er wie gewünscht funktioniert. Dieser Ansatz wird als „Test First"- oder testgetriebene Entwicklung bezeichnet; diese Testfälle können auch als ausführbare Spezifikation angesehen werden (vgl. [Linz14], Kap. 4.2).

Typische Arbeitsergebnisse von Testern in agilen Projekten beinhalten automatisierte Tests sowie Testpläne, Qualitätsrisikokataloge, Testspezifikationen, Fehlerberichte und Testergebnisprotokolle. Diese Dokumente sind in einer möglichst leichtgewichtigen Art und Weise zu erstellen. Tester erstellen außerdem Testmetriken aus Fehlerberichten und Testergebnisprotokollen und auch hier liegt der Fokus auf einem leichtgewichtigen Ansatz.

In einigen agilen Umgebungen, insbesondere in regulierten, sicherheitskritischen, dezentralisierten oder hochkomplexen Projekten bzw. Produkten ist eine weitergehende Formalisierung dieser Arbeitsergebnisse notwendig. Einige Teams übertragen beispielsweise User-Stories und Abnahmekriterien in formellere Anforderungsbeschreibungen. Vertikale und horizontale Rückverfolgungsberichte (traceability reports) können erstellt werden um Auditoren, Regulierungsvorschriften und anderen Anforderungen gerecht zu werden.

2.1.3 Teststufen

Teststufen entsprechen logisch zusammenhängenden Testaktivitäten, deren Zusammenstellung oftmals durch die Reife oder Vollständigkeit des Testobjekts bedingt wird.

In sequentiellen Lebenszyklusmodellen sind die Teststufen häufig so definiert, dass die Ausgangskriterien der einen Stufe Teil der Eingangskriterien für die nächste Stufe darstellen. Für einige iterative Modelle trifft dies nicht zu, da die Teststufen überlappend sind und mit ihnen die Anforderungsbeschreibung, Designbeschreibung und Entwicklungsaktivitäten.

In einigen agilen Ansätzen findet eine solche Überlappung statt, weil Veränderungen an den Anforderungen, am Design und am Code zu jedem Zeitpunkt einer Iteration vorkommen können. Während Scrum Veränderungen an den User-Stories nach der Iterationsplanung nicht zulässt, finden solche Veränderungen in der Praxis dennoch gelegentlich statt.

Während einer Iteration durchläuft eine User-Story in der Regel die folgenden Testaktivitäten der Reihe nach:

- Durchführung von Unittests, typischerweise durch einen Entwickler
- Abnahmetests für das Feature, teilweise in zwei Aktivitäten aufgeteilt:
    - Verifizierungstest des Features, der häufig automatisiert erfolgt. Er kann von Entwicklern oder Testern durchgeführt werden und beinhaltet das Testen gegen die Abnahmekriterien der User-Story
    - Validierungstest des Features, der üblicherweise manuell erfolgt und in den Entwickler, Tester und Vertreter des Fachbereichs einbezogen werden können, um gemeinsam die Einsetzbarkeit des Features festzustellen, um den erzielten Fortschritt sichtbarer zu machen und um eine direktes Feedback von den Fachbereichsvertretern zu erhalten

Darüber hinaus gibt es oft parallel dazu noch Regressionstests, die während der gesamten Iteration stattfinden. Diese beinhalten einen wiederholten Durchlauf der automatisierten Unittests und der Verifikationstests der Feature aus der laufenden und aus den vorangegangenen Iterationen, üblicherweise eingebettet in ein Continuous-Integration-Framework.

In einigen agilen Teams gibt es auch eine Systemteststufe, die beginnt, sobald die erste User-Story für derartiges Testen bereit ist. Diese kann die Durchführung von funktionalen Tests aber auch von nicht-funktionalen Tests bezüglich der Performanz, Zuverlässigkeit, Benutzbarkeit, Stabilität und anderen relevanten Testarten beinhalten.

Agile Teams können verschiedene Formen von Abnahmetests anwenden (im Sinne des Foundation Level-Lehrplan [ISTQB_FL_SYL] hier verwendet). Es können interne Alpha-Tests und externe Beta-Tests vorkommen, entweder am Ende jeder Iteration, nach Abschluss jeder Integration oder nach einer Reihe von Iterationen. Benutzer-Abnahmetests, betriebliche Abnahmetests, regulatorische Abnahmetests und vertragliche Abnahmetests können ebenso vorkommen, ebenfalls am Ende jeder Iteration, nach einer Reihe von Iterationen oder nach Abschluss aller Iterationen.

2.1.4 Werkzeuge zur Verwaltung von Tests und Konfigurationen

Agile Projekte sind häufig durch die starke Nutzung von automatisierten Werkzeugen für die Entwicklung, für Tests und die Verwaltung der Softwareentwicklung gekennzeichnet. Entwickler werden dazu angehalten, Werkzeuge für die statische Analyse und für Unittests zu verwenden (zur Durchführung von Tests und zur Messung der Codeabdeckung). Diese statische Analyse und die Unittests finden nicht nur während der Entwicklung statt, sondern auch nachdem der Entwickler seinen Code im Konfigurationsmanagementwerkzeug eingecheckt hat. Dabei bedient man sich automatisierter Programmier- und Test-Frameworks. Diese Frameworks ermöglichen eine kontinuierliche Integration der Software im System, wobei die statische Analyse und die Unittests immer dann wiederholt ablaufen, wenn neuer Code eingecheckt wird [Kubaczkowski].

Diese automatisierten Tests können auch funktionale Tests auf der Integrations- und Systemstufe umfassen. Solche funktionalen automatisierten Tests können mit Hilfe von funktionalen Testrahmen, mit Open-Source GUI Testautomationswerkzeugen oder auch mit kommerziellen Werkzeugen erstellt und in die automatisierten Tests integriert werden, die als Teil der kontinuierlichen Integration (continuous integration) laufen. In einigen Fällen werden die funktionalen Tests aufgrund ihrer zu langen Laufzeit von den Unittests getrennt und laufen weniger häufig. Beispielsweise können Unittests bei jedem Check-In laufen, während die länger laufenden Tests nur einmal täglich oder nachts oder in noch größeren Intervallen durchgeführt werden.

Ein Ziel der automatisierten Tests ist es zu bestätigen, dass der Build lauffähig und installierbar ist. Falls ein automatisierter Test fehlschlägt, sollte das Team den zu Grunde liegenden Fehler rechtzeitig vor dem nächsten Check-In beheben. Dies erfordert Investitionen in Echtzeit-Testberichte, die eine gute Detailsicht auf die Testergebnisse liefern. Dieser Ansatz hilft jene teuren und ineffizienten Zyklen des „Schreiben-Installieren-Fehlschlagen-NeuSchreiben-NeuInstallierens“ zu reduzieren, die in vielen traditionellen Projekten vorkommen. Änderungen, die einen Build scheitern oder die Installation der Software fehlschlagen lassen, werden so schneller gefunden. [Bucsics14]

Automatisierte Test- und Build-Werkzeuge helfen den Risiken zu begegnen, die mit den häufigen Änderungen in agilen Projekten einhergehen. Allerdings kann es problematisch sein, sich zu sehr auf automatisierte Tests allein zu verlassen, um diese Risiken zu handhaben, da Unittests oft nur eine beschränkte Effektivität bei der Entdeckung von Fehlern aufweisen [Jones11]. Automatisierte Tests auf Integrations- und Systemstufe sind ebenfalls notwendig.

2.1.5 Organisationsmöglichkeiten für unabhängiges Testen

Wie bereits im Foundation Level-Lehrplan [ISTQB_FL_SYL] beschrieben sind unabhängige Tester oft effektiver bei der Suche nach Fehlern. In einigen agilen Teams erstellen Entwickler viele der Tests automatisiert. Ein professioneller Tester kann in das Team integriert sein und Teile des Testens übernehmen. Allerdings führt ein in das Team integrierter Tester zum Risiko des Verlusts an Unabhängigkeit oder des Fehlens von objektiver Beurteilung, die von außerhalb des Teams gegeben wäre.

Andere agile Teams behalten vollständig unabhängige, separate Testteams und weisen Testern auf Abruf während der letzten Tage des Sprints Aufgaben zu. Dies kann die Unabhängigkeit aufrechterhalten und die Tester können eine objektive, unvoreingenommene Beurteilung der Software abgeben. Allerdings führen Zeitdruck, Probleme mit dem Verständnis für die neuen Features des Produkts und Unstimmigkeiten mit den Fachbereichsvertretern und Entwicklern bei diesem Ansatz häufig zu Problemen.

Eine dritte Möglichkeit ist die eines unabhängigen, separaten Testteams. Tester werden hierbei langfristig für agile Teams abgestellt, um ihre Unabhängigkeit zu bewahren und ihnen dennoch die Möglichkeit zu geben, ein tieferes Verständnis des Produkts und eine enge Beziehung zu anderen Teammitgliedern und Stakeholdern aufzubauen. Darüber hinaus kann das unabhängige Testteam einige spezialisierte Tester außerhalb des agilen Teams vorsehen, die an langfristigen und/oder nicht iterations-bezogenen Aktivitäten arbeiten, wie z. B. an der Entwicklung von automatisierten Testwerkzeugen, an der Durchführung nicht-funktionaler Tests, an der Erstellung von Testumgebungen bzw. -daten und an der Durchführung von Teststufen, die nicht gut in einen Sprint passen (z. B. Systemintegrationstests).

2.2 Der Status des Testens in agilen Projekten

In agilen Projekten kommt es permanent zu Änderungen. Diese Änderungen haben zur Folge, dass der Teststatus, der Testfortschritt, und die Produktqualität sich ebenfalls ständig ändern bzw. weiterentwickeln. Tester müssen daher einen Weg finden, diese Informationen so geeignet an das

Team weiterzugeben, dass dieses kluge Entscheidungen für einen erfolgreichen Abschluss jeder Iteration treffen kann. Darüber hinaus können Änderungen auch die schon bestehenden Features aus früheren Iterationen betreffen. Daher müssen manuelle und automatisierte Tests ständig aktuell gehalten werden, um dem Regressionsrisiko effektiv zu begegnen.

2.2.1 Kommunikation über den Teststatus, den Fortschritt und die Produktqualität

In agilen Teams wird Fortschritt dadurch beschrieben, dass am Ende jeder Iteration funktionierende Software zur Verfügung steht. Um feststellen zu können, wann das Team funktionierende Software zur Verfügung stellen kann, muss es den Fortschritt jedes einzelnen Arbeitsschrittes in der Iteration und im Release überwachen. Tester in agilen Teams nutzen unterschiedliche Methoden, um den Testfortschritt bzw. -status nachzuverfolgen. So werden Ergebnisse der Testautomatisierung, der Fortschritt bei der Erledigung von Testaufgaben und Stories im agilen Task-Board und in Burndown-Charts [Crispin08] verwendet, um den Erfolg des Teams darzustellen. Diese Informationen können dann mit diversen Hilfsmitteln, wie Wiki Dashboards, Dashboard-artigen E-Mails oder auch verbal in Stand-up-Meetings kommuniziert werden. Einige Agile Teams verwenden auch Werkzeuge, die automatische Statusberichte basierend auf Testergebnissen und Aufgabenfortschritt generieren, womit dann wiederum die Wiki Dashboards und E-Mails aktuell gehalten werden. Bei dieser Art der Informationsvermittlung werden oft auch Metriken aus dem Testprozess erhoben, die für die Prozessverbesserung genutzt werden können. Die automatische Kommunikation des Testprozessstatus bewirkt außerdem, dass Tester mehr Zeit haben, um sich auf das Design und die Durchführung weiterer Testfälle zu konzentrieren.

Teams können Burndown-Charts nutzen, um den Fortschritt während des gesamten Releases und während einer Iteration nachzuverfolgen. Ein Burndown-Chart stellt die noch unverrichtete Arbeit der für das Release oder die Iteration verfügbaren Zeit gegenüber.

Agile Teams verwenden gerne auch Agile Task-Boards, um sich den Status im Team inklusive des Teststatus zu vergegenwärtigen. Story-Cards, Entwicklungsaufgaben, Testaufgaben und andere Aufgaben, die während der Iterationsplanung erstellt wurden (siehe Abschnitt 1.2.5) werden im Task-Board aufgelistet, oft unter Verwendung von farblich abgestimmten Karten, die den jeweiligen Aufgabentypus repräsentieren. Während der Iteration wird der Fortschritt durch die Bewegung der Aufgaben über das Board hinweg in Spalten wie „Zu erledigen“, „In Bearbeitung“, „Zu prüfen“ und „Erledigt“ verwaltet. Agile Teams setzen zuweilen auch Werkzeuge zur Verwaltung ihrer Story-Cards und Agilen Task-Boards ein, die das Dashboard und den Status automatisch aktualisieren.

Testaufgaben im Task-Board beziehen sich auf die Abnahmekriterien, die in den User-Stories definiert wurden. Wenn Testautomatisierungsskripts, manuelle Tests und explorative Tests für eine bestimmte Testaufgabe erfolgreich gelaufen sind, wandert diese Aufgabe in die Spalte „Erledigt“ des Task-Boards. Das gesamte Team überwacht den Status des Task-Boards regelmäßig und häufig während der täglichen Stand-up-Meetings, um sicher zu gehen, dass die Aufgaben in der gewünschten Geschwindigkeit am Board weiterbewegt werden. Wenn einige Aufgaben (darunter auch Testaufgaben) sich nicht oder nur zu langsam vorwärts bewegen, prüft das Team mögliche Gründe, weshalb der Fortschritt der Aufgabe blockiert sein könnte und kümmert sich um Lösungen.

Das tägliche Stand-up-Meeting (daily stand-up meeting) bezieht alle Mitglieder des agilen Teams ein, somit auch die Tester. In dieser Besprechung tragen alle ihren aktuellen Status vor. Die Agenda ist für jedes Mitglied die folgende [Agile Alliance Guide]:

- Was hast Du seit dem letzten Stand-up-Meeting abgeschlossen?
- Was planst Du bis zum nächsten Stand-up-Meeting abzuschließen?
- Was steht Dir im Weg?

Jegliche Probleme, die den Testfortschritt behindern können, werden während des täglichen Stand-ups angesprochen, somit ist das gesamte Team über die Probleme informiert und kann sie entsprechend lösen.

Zur allgemeinen Verbesserung der Produktqualität führen viele agile Teams Kundenzufriedenheitsbefragungen durch, um eine Rückmeldung (Feedback) darüber zu erhalten, ob das Produkt die Kundenerwartungen erfüllt. Die Teams können zur Verbesserung der Produktqualität auch Metriken verwenden, die denen in traditionellen Entwicklungsmethoden ähnlich sind, wie z. B. Rate erfolgreicher / fehlerhafter Tests, Fehleraufdeckungsrate, Ergebnisse von Bestätigungs- und Regressionstests, Fehlerdichte, gefundene und behobene Fehler, Anforderungsabdeckung, Risikoabdeckung, Codeüberdeckung und Änderungen am Code (Code Churn).

Wie in jedem Projektlebenszyklus sollten die angewandten Metriken genügend Relevanz aufweisen und bei der Entscheidungsfindung helfen. Metriken sollten nicht dazu verwendet werden, um Teammitglieder zu belohnen, zu bestrafen oder zu isolieren.

2.2.2 Das Regressionsrisiko trotz zunehmender Zahl manueller und automatisierter Testfällen beherrschen

In einem agilen Projekt wächst das Produkt mit jeder Iteration. Daher erhöht sich auch der Umfang des Testens. Neben den Tests für die Änderungen am Code in der aktuellen Iteration müssen Tester auch sicherstellen, dass bei bereits getesteten Features aus vorangegangenen Iterationen keine Verschlechterung eingetreten ist. Das Risiko für eine Verschlechterung am Code ist in der agilen Entwicklung groß, da die Änderungen am Code (Codezeilen, die von einer Version auf die nächste hinzugefügt, modifiziert oder gelöscht wurden) im Regelfall sehr umfangreich sind. Da der Umgang mit Veränderungen (responding to change) ein agiles Schlüsselprinzip ist, können auch schon gelieferte Features jederzeit geändert bzw. angepasst werden, um Geschäftsanforderungen zu erfüllen. Um nun die Teamproduktivität (Velocity) beizubehalten ohne aber dabei zu viel an technischen Schulden in Kauf nehmen zu müssen, ist es entscheidend, dass die Teams so früh wie möglich auf allen Teststufen in die Automatisierung von Tests investieren. Es ist außerdem noch wichtig, dass der Bestand an Tests, wie automatisierte Tests, manuelle Testfälle, Testdaten und andere Testartefakte in jeder Iteration aktuell gehalten werden. Es wird dringend empfohlen, den Bestand an Tests samt aller dazu gehörigen Testartefakte in einem Konfigurationsmanagementwerkzeug zu verwalten, um eine Versionskontrolle zu erreichen. Damit ermöglicht man allen Teammitgliedern einen einfachen Zugriff auf die Testartefakte, womit sie jederzeit Anpassungen aufgrund veränderter Funktionalität vornehmen können, ohne die Dokumenthistorie dabei zu verlieren.

Tester müssen in jeder Iteration Zeit für die Aktualisierung vorhandener Testfälle einplanen. Es gilt, manuelle und automatisierte Testfälle aus früheren und aktuellen Iterationen zu prüfen und Testfälle auszuwählen, die weiterhin für die Regressionstests geeignet sind und schließlich auch Testfälle zu entfernen, die nicht länger relevant sind. Tests aus früheren Iterationen haben in späteren Iterationen aufgrund von Feature-Veränderungen oder neuen Features häufig einen geringen Wert, weil sich das getestete Feature inzwischen ganz anders verhalten kann.

Während der Überprüfung der Testfälle sollten Tester auch deren Eignung für die Automatisierung beurteilen. Das Team sollte so viele manuelle Tests aus früheren und aktuellen Iterationen wie möglich automatisieren. Dies reduziert bei den folgenden Iterationen den Durchführungsaufwand der Regression. Der so minimierte Regressionstestaufwand gibt den Testern in der aktuellen Iteration dann den Freiraum, umso sorgfältiger neue Features und Funktionen zu testen.

Es ist von zentraler Bedeutung, dass Tester jene Testfälle aus früheren Iterationen und/oder Releases schnell zu identifizieren und zu aktualisieren wissen die durch Änderungen in der aktuellen Iteration betroffen sein könnten. Die Festlegung, wie das Team Testfälle entwirft, schreibt und speichert sollte schon während der Releaseplanung erfolgen. Gute Vorgehensweisen für den Entwurf und die

Implementierung müssen früh angepasst und durchgängig verwendet werden. Die kurzen Zeitintervalle für das Testen und die permanenten Änderungen in jeder Iteration verschärfen die Auswirkungen von schlechten Testentwurfs- und Implementierungspraktiken noch.

Die Nutzung von Testautomatisierung in allen Teststufen ermöglicht es agilen Teams, ein schnelles Feedback zur Produktqualität zu bekommen. Gut geschriebene automatisierte Tests liefern ein lebendes Dokument der Systemfunktionalität [Crispin08]. Durch das Check-In, (Einchecken bzw. Commit) der automatisierten Tests und ihrer Ergebnisse in ein Konfigurationsmanagement inklusive Versionierung der Produkt-Builds, können agile Teams die getestete Funktionalität sowie die jeweiligen Testergebnisse für jedes Build zu jeder Zeit abrufen.

Automatisierte Unittests werden durchgeführt bevor der Quellcode in die Baseline (= Ausgangspunkt) des Konfigurationsmanagements eingefügt wird, um sicherzustellen, dass der Softwarebuild nicht beschädigt wird. Um Schäden am Build zu vermeiden, die den Fortschritt der gesamten Teamarbeit empfindlich bremsen können, sollte der Code nicht hinzugefügt werden bis alle automatisierten Unittests abgeschlossen sind. Automatisierte Unittest-Ergebnisse liefern ein unmittelbares Feedback zur Code- und Build-Qualität, allerdings nicht zur Produktqualität. [Bucsics14]

Automatisierte Abnahmetests laufen regelmäßig als Teil der kontinuierlichen Integration bei der Erstellung des vollständigen Systems. Diese Tests laufen mindestens einmal täglich gegen einen vollständigen Build des Systems, nicht aber grundsätzlich bei jedem Code-Check-In, da sie länger als automatisierte Unittests dauern und somit den Check-In verlangsamen. Die Testergebnisse der automatisierten Abnahmetests liefern Feedback zur Produktqualität in Bezug auf die Veränderungen seit dem letzten Build, aber sie liefern keine Information zur allgemeinen Produktqualität.

Automatisierte Tests können kontinuierlich gegen das System laufen. Unmittelbar nach der Lieferung eines neuen Builds in die Testumgebung sollte ein Mindestbestand an automatisierten Tests zur Abdeckung der kritischen Systemfunktionalitäten und Integrationspunkte durchgeführt werden. Diese Tests sind allgemein als Build-Verifizierungstest bekannt. Ergebnisse dieser Build-Verifizierungstests liefern ein schnelles Feedback zur Software gleich nach dem Deployment, damit Teams keine Zeit mit dem Test instabiler Builds verschwenden müssen.

Automatisierte Tests, die im Regressionstestset enthalten sind, laufen im Allgemeinen als Teil des täglichen Haupt-Builds in der Continuous-Integration-Umgebung und dann erst wieder, wenn ein neuer Build in die Testumgebung ausgeliefert wird. Sobald ein automatisierter Regressionstest fehlschlägt, unterbricht das Team seine Arbeit und prüft die Gründe für den fehlgeschlagenen Test. Der Test kann durch gewollte Funktionsänderungen in der aktuellen Iteration fehlgeschlagen sein. In diesem Fall muss der Test und/oder die User-Story aktualisiert werden, um die neuen Abnahmekriterien abzudecken. Falls ein neuer Test erstellt wurde, um die Veränderungen abzudecken, kann es sein, dass der alte Test entfernt werden muss. Falls der Test jedoch aufgrund eines Fehlers in der Software fehlgeschlagen ist, ist es ratsam für das Team, den Fehler erst zu beheben, bevor es weitere Features implementiert.

Zusätzlich zur automatisierten Testdurchführung können die folgenden Testaufgaben automatisiert werden:

- Die Erstellung von Testdaten
- Das Laden der Testdaten in die Systeme
- Das Ausliefern von Builds in die Testumgebungen (meist Teil der kontinuierlichen Integration (continuous integration))
- Das Zurücksetzen einer Testumgebung in den Grundzustand (z. B. der Datenbank oder Datensätze für Webseiten)
- Vergleich von Datenergebnissen

Die Automatisierung dieser sich oft wiederholenden Aufgaben reduziert den Aufwand und ermöglicht es dem Team, Zeit für die Entwicklung und den Test von neuen Features zu gewinnen.

2.3 Rolle und Fähigkeiten eines Testers in einem agilen Team

In einem agilen Team müssen die Tester mit allen Teammitgliedern und den fachlichen Ansprechpartnern besonders eng zusammenarbeiten. Daraus ergeben sich eine Vielzahl von speziellen Anforderungen an die fachlichen Fähigkeiten und Kenntnisse der Tester und die Aktivitäten der Tester innerhalb eines agilen Teams.

2.3.1 Fähigkeiten agiler Tester

Agile Tester sollten alle Fähigkeiten haben, die im Lehrplan zum Foundation Level [ISTQB_FL_SYL] erwähnt wurden. Darüber hinaus wird ein Tester in einem agilen Team stark in jenen Bereichen gefordert sein, die eher dem agilen bzw. iterativen Vorgehen zuzuordnen sind:

- Kenntnisse in der Testautomatisierung,
- Vertrautheit mit testgetriebener Entwicklung (“test driven development”, kurz “TDD””) und abnahmetestgetriebener Entwicklung (“acceptance test-driven development”, kurz “A-TDD”),
- Sicherheit im Umgang mit White-Box-, Black-Box- und erfahrungsbasierten Tests,
- Fähigkeit, “just enough” zu dokumentieren.

Da agile Methoden in hohem Maße von der Zusammenarbeit, Kommunikation und Interaktion zwischen den Teammitgliedern und den Stakeholdern außerhalb des Teams abhängen, sollten sich Tester dessen bewusst sein, dass insbesondere ihre zwischenmenschlichen Fähigkeiten (“soft skills”) wichtig sind. Tester in agilen Teams sollten daher:

- positiv und lösungsorientiert gegenüber Teammitgliedern und anderen Mitarbeitern außerhalb des Teams auftreten,
- eine eher kritische, qualitätsorientierte, skeptische Denkweise über das Produkt an den Tag legen,
- Informationen aktiv vom Fachbereich / Auftraggeber einholen (statt sich nur völlig auf die geschriebenen Spezifikationen zu verlassen),
- Testergebnisse, Testfortschritte und Produktqualität genau beurteilen und darüber berichten,
- zusammen mit Kunden bzw. dem Fachbereich effektiv an der Definition prüfbarer User-Stories und insbesondere auch an den Abnahmekriterien arbeiten,
- im Team mitarbeiten, paarweise mit Programmierern und anderen Teammitgliedern arbeiten,
- schnell auf Veränderungen reagieren, d.h. Testfälle anpassen, hinzufügen oder verbessern,
- ihre eigene Arbeit planen und organisieren.

Die kontinuierliche Weiterentwicklung der persönlichen Fähigkeiten, darunter auch zwischenmenschlicher Fähigkeiten, sind für Tester an sich immer wichtig, aber ganz speziell für Tester in agilen Teams.

2.3.2 Die Rolle eines Testers in einem agilen Team

Die Rolle des Testers in einem agilen Team konzentriert sich nicht nur auf das Erheben und Bereitstellen von Informationen zum Teststatus, Testfortschritt und zur Produktqualität. Der Tester muss sich ebenso intensiv mit der Prozessqualität auseinandersetzen. Ergänzend zu den Aktivitäten, die an anderer Stelle in diesem Lehrplan beschrieben sind, obliegen dem Tester noch folgende Aufgaben:

- Die Teststrategie verstehen, implementieren und aktualisieren

- Die Testüberdeckung über alle anzuwendenden Metriken hinweg messen und berichten
- Den richtigen Einsatz der Testwerkzeuge sicherzustellen
- Testumgebungen sowie Testdaten konfigurieren, verwenden und verwalten
- Fehlerberichte erstellen und mit dem Team bei der Fehlerbehebung zusammenarbeiten
- Teammitglieder in die wesentlichen Prinzipien des Testens einzuweisen
- Sicherstellen, dass die Tests angemessen sind und in der Release- und Iterationsplanung berücksichtigt werden
- Mit Entwicklern, Fachbereich und Product-Owner zur Klärung der Anforderungen insbesondere in Bezug auf Testbarkeit, Konsistenz und Vollständigkeit zusammenarbeiten
- An Team Retrospektiven proaktiv teilzunehmen und dort Verbesserungen mit vorschlagen und umsetzen

Innerhalb eines agilen Teams ist jedes Teammitglied für die Produktqualität mit verantwortlich und kümmert sich um die Durchführung testbezogener Aufgaben.

Agil arbeitende Organisationen bzw. Unternehmen müssen auch mit folgenden organisations-bezogenen Risiken rechnen:

- Tester arbeiten so eng mit Entwicklern zusammen, dass sie ihre objektive Sicht verlieren
- Tester tolerieren oder schweigen zu ineffizienten, ineffektiven oder qualitativ schwachen Praktiken innerhalb des Teams
- Tester können mit den permanenten Changes bei gleichzeitig kurzen Iterationszyklen nicht mithalten

Um diesen Risiken zu begegnen, sollten Organisationen Alternativen zur Wahrung der Unabhängigkeit und Objektivität des Testens in Betracht ziehen, wie schon in Abschnitt 2.1.5 beschrieben.

3. Methoden, Techniken und Werkzeuge des agilen Testens – 480 min

Schlüsselwörter

Qualitätsrisiko, sitzungsbasiertes Testmanagement, Test-Charta, testgetriebene Entwicklung, abnahmetestgetriebene Entwicklung, verhaltensgetriebene Entwicklung, Unittest-Framework

Lernziele für Methoden, Techniken und Werkzeuge des agilen Testens

3.1 Agile Testmethoden

FA-3.1.1 (K1) Die Konzepte der testgetriebenen Entwicklung, der abnahmetestgetriebenen Entwicklung und der verhaltensgetriebenen Entwicklung nennen können
FA-3.1.2 (K1) Die Konzepte der Testpyramide nennen können
FA-3.1.3 (K2) Die Testquadranten und ihre Beziehungen zu Teststufen und Testarten zusammenfassen
FA-3.1.4 (K3) Für ein vorgegebenes agiles Projekt die Rolle eines Testers in einem Scrum Team übernehmen

3.2 Qualitätsrisiken beurteilen und Testaufwände schätzen

FA-3.2.1 (K3) Qualitätsrisiken in einem agilen Projekt einschätzen
FA-3.2.2 (K3) Testaufwand auf Basis des Iterationsinhalts und der Qualitätsrisiken schätzen

3.3 Techniken in agilen Projekten

FA-3.3.1 (K3) Die relevanten Informationen interpretieren können, um Testaktivitäten zu unterstützen.
FA-3.3.2 (K2) Den Fachbereichsvertretern erklären können, wie testbare Abnahmekriterien zu definieren sind.
FA-3.3.3 (K3) Für eine vorgegebene User-Story Abnahmetestgetriebene Testfälle (ATDD) schreiben können.
FA-3.3.4 (K3) Auf Basis von vorgegebenen User-Stories mit Hilfe von Black-Box-Testentwurfsverfahren funktionale und nicht-funktionale Testfälle schreiben können.
FA-3.3.5 (K3) Explorative Tests durchführen können, um das Testen eines agilen Projekts zu unterstützen.

3.4 Werkzeuge in agilen Projekten

FA-3.4.1 (K1) Verschiedene für Tester verfügbare Werkzeuge gemäß ihres Zwecks und der Aktivitäten in agilen Projekten kennen.

3.1 Agile Testmethoden

Es gibt bestimmte Testmethoden, die in jedem Entwicklungsprojekt (agil oder nicht) genutzt werden können, um Qualitätsprodukte zu erstellen. Einige davon sind das Schreiben von Tests vorab, um das richtige Verhalten auszudrücken, die Konzentration auf eine frühe Fehlervermeidung und Fehlerbehebung und das Sicherstellen, dass die richtigen Testarten zur richtigen Zeit und als Teil der richtigen Teststufe angewendet werden. Agile Teams versuchen, diese Praktiken früh anzuwenden. Tester in agilen Projekten spielen eine Schlüsselrolle dabei, die Nutzung dieser Praktiken über den gesamten Lebenszyklus zu begleiten.

3.1.1 Testgetriebene Entwicklung, abnahmetestgetriebene Entwicklung und verhaltensgetriebene Entwicklung

Testgetriebene Entwicklung, abnahmetestgetriebene Entwicklung und verhaltensgetriebene Entwicklung sind drei sich gegenseitig ergänzende Techniken, die in agilen Teams genutzt werden, um die Tests auf den verschiedenen Teststufen durchzuführen. Jede Technik ist ein Beispiel für ein grundlegendes Prinzip des Testens, den Vorteil von frühen Test- und Qualitätssicherungsmaßnahmen, da die Tests bereits definiert sind bevor der Code geschrieben ist.

Testgetriebene Entwicklung

Testgetriebene Entwicklung (test-driven development, TDD) wird dazu genutzt, Code mit Hilfe von automatisierten Testfällen zu erstellen. Der Prozess für testgetriebene Entwicklung ist folgender:

- Einen Testfall erstellen und automatisieren, der die gewünschte Funktion eines kleinen Teils an Code beschreibt.
- Diesen Testfall laufen lassen, der fehlschlagen sollte, da der Code ja noch nicht existiert.
- Den Test wiederholt laufen lassen und den Code solange verbessern bzw. ergänzen bis der Test erfolgreich bestanden wird.
- Falls der Code nachträglich geändert wird (z. B. im Zuge von Refactoring), den Test erneut durchlaufen lassen, um sicherzustellen, dass er auch mit dem umgeschriebenen Code bestanden wird.
- Diesen Vorgang für das nächste kleine Stück Code wiederholen und dabei sowohl die vorherigen Tests als auch die hinzugefügten Tests durchführen.

Die geschriebenen Tests befinden sich überwiegend auf Unittest-Stufe und sind code-bezogen, obwohl testgetriebene Entwicklung auch auf Integrations- oder Systemtest-Stufe praktiziert werden kann (s.a. [Linz14], Kap. 6.5). Testgetriebene Entwicklung wurde durch Extreme Programming [Beck02] populär, wird aber auch in anderen agilen Methoden und manchmal auch in sequentiellen Lebenszyklen verwendet. Die Entwickler konzentrieren sich auf klar definierte, erwartete Ergebnisse. Die Tests sind automatisiert und werden in der kontinuierlichen Integration (continuous integration) verwendet.

Abnahmetestgetriebene Entwicklung

Abnahmetestgetriebene Entwicklung [Gärtner13] definiert Abnahmekriterien und Tests während der Entwicklung von User-Stories (siehe Abschnitt 1.2.2). Abnahmetestgetriebene Entwicklung ist ein kollaborativer Ansatz, der es jedem Interessenvertreter ermöglicht, zu verstehen, wie die Softwarekomponente sich verhalten soll und was Entwickler, Tester und Fachbereichsvertreter tun müssen, um dieses Verhalten zu erreichen. Der Prozess der Abnahmetestgetriebenen Entwicklung wird im Abschnitt 3.3.2 erläutert.

In der Abnahmetestgetriebenen Entwicklung werden wiederverwendbare Tests für die Regressionstests erstellt. Spezifische Werkzeuge unterstützen die Erstellung und Durchführung solcher Tests, häufig innerhalb des Prozesses der kontinuierlichen Integration. Diese Werkzeuge

können mit Daten und Serviceebenen der Anwendung verbunden sein, was es ermöglicht, die Tests auf System- oder Abnahmeniveau durchzuführen. Abnahmetestgetriebene Entwicklung ermöglicht eine schnelle Lösung von Fehlern und die Bewertung des Verhaltens des Features. Sie hilft zu bestimmen, ob die Abnahmekriterien für das Feature erfüllt sind.

Verhaltensgetriebene Entwicklung

Verhaltensgetriebene Entwicklung (behaviour-driven development, BDD) [Chelimsky10] ermöglicht es dem Entwickler, sich darauf zu konzentrieren, den Code auf das erwartete Verhalten der Software hin zu testen. Da die Tests auf dem erwarteten Verhalten der Software basieren, sind sie in der Regel für andere Teammitglieder und Interessenvertreter leichter zu verstehen.

Spezifische Frameworks für BDD erlauben die Definition der Abnahmekriterien auf Basis des „Gegeben/Wenn/Dann-Formats“ (sog. Gherkin-Format):

Gegeben einen Einstiegskontext,
Wenn ein Ereignis auftritt,
Dann werden bestimmte Wirkungen sichergestellt.

Aus diesen Definitionen erstellt das Framework ausführbaren Testcode. Verhaltensgetriebene Entwicklung hilft automatisierte Testfälle zu definieren. Diese sind auch für Personen, die nicht programmieren können, lesbar und verständlich sind.

3.1.2 Die Testpyramide

Ein Softwaresystem kann auf unterschiedlichen Stufen getestet werden. Typische Teststufen sind, vom unteren Ende der Pyramide bis zur Spitze, Komponente, Integration, System und Abnahme (siehe [ISTQB_FL_SYL] Abschnitt 2.2) Die Metapher der Pyramide illustriert die größere Anzahl an Testfällen auf Unittest-Stufe (am unteren Ende der Pyramide) im Vergleich zu der geringer werdenden Anzahl von Testfällen mit steigender Teststufe (oberes Ende der Pyramide). Normalerweise sind Tests auf der Unit- und der Integrationsstufe automatisiert und werden mit Hilfe von API-basierten Werkzeugen erstellt. Auf der System- und der Abnahmestufe werden die automatisierten Tests mit Hilfe von GUI-basierten Werkzeugen erstellt. Das Konzept der Testpyramide folgt dem Prinzip der frühestmöglichen Qualitätssicherung (d.h. das Beseitigen von Fehlern so früh wie möglich im Lebenszyklus).

3.1.3 Testquadranten, Teststufen und Testarten

Testquadranten, wie sie von Brian Marick definiert wurden [Crispin08], verbinden die Teststufen mit den passenden Testarten in der agilen Methodologie. Das Modell der Testquadranten und seiner Varianten hilft sicherzustellen, dass alle wichtigen Testarten und Teststufen in den Entwicklungslebenszyklus einbezogen werden. Mit Hilfe dieses Modells können die Testarten auch gegenüber allen Interessenvertretern, darunter Entwickler, Tester und Fachbereichsvertreter, unterschieden und beschrieben werden.

In den Testquadranten werden Tests entlang zweier Achsen klassifiziert. Auf der vertikalen Achse technisch (technology facing) vs. geschäftsprozessorientierte (business facing) Tests und auf der horizontalen Achse teamunterstützende vs. produkthinterfragende Tests. Die vier Quadranten sind die folgenden:

- Quadrant Q1 ist der Unit Level, Technologie-orientiert, und unterstützt die Entwickler. Dieser Quadrant enthält Unittests. Diese Tests sollten automatisiert und im Prozess der kontinuierlichen Integration (continuous integration) enthalten sein.

- Quadrant Q2 ist der System Level, unternehmensorientiert, und bestätigt das Produktverhalten. Dieser Quadrant enthält funktionale Tests, Beispiele, Story Tests, Prototypen für die User Experience und Simulationen. Diese Tests prüfen die Abnahmekriterien und können sowohl manuell als auch automatisiert sein. Sie werden häufig während der Entwicklung der User-Story erstellt und verbessern so die Qualität der Stories. Sie sind von Nutzen für die Erstellung der Testfolgen für die automatisierten Regressionstests.
- Quadrant Q3 ist der System- oder Nutzerakzeptanzlevel, unternehmensorientiert, und enthält Tests, die das Produkt mit Hilfe von realistischen Szenarien und Daten kritisieren. Dieser Quadrant enthält explorative Tests, Szenarios, Prozessabläufe, Tests zur Benutzbarkeit, Benutzer-Abnahmetests, Alpha- und Beta-Tests. Diese Tests sind häufig manuell und nutzerorientiert.
- Quadrant Q4 ist der System- oder betriebliche Abnahmelevel, technologieorientiert, und enthält Tests, die das Produkt kritisieren. Dieser Quadrant enthält Performanz-, Last-, Stress- und Skalierbarkeitstests, Zugriffssicherheitstests, Wartbarkeitstests, Tests bzgl. Kompatibilität und Interoperabilität, Datenmigration, Infrastruktur und Wiederherstellung. Diese Tests sind oft automatisiert.

Während jeder beliebigen Iteration können Tests aus allen Quadranten notwendig sein. Die Testquadranten werden eher auf dynamisches als auf statisches Testen angewendet.

3.1.4 Die Rolle des Testers

Der Lehrplan hat sich bisher auf die agilen Methoden und Techniken sowie die Rolle des Testers innerhalb eines agilen Lebenszyklus bezogen. In diesem Abschnitt wird insbesondere die Rolle des Testers in einem Projekt, das dem Scrum-Lebenszyklus folgt, betrachtet [Aalst13].

Teamwork

Teamwork ist ein Grundprinzip in der agilen Entwicklung. Der agile Ansatz betont den Gesamtteamansatz, in dem das Team aus Entwicklern, Testern und Fachbereichsvertretern besteht, die alle zusammenarbeiten. Die folgenden Best Practices gelten für die Organisation und das Verhalten von Scrum Teams:

- Funktionsübergreifend: Jedes Teammitglied trägt mit seinen speziellen Fähigkeiten zum Team bei. Das Team arbeitet zusammen an der Teststrategie, der Testplanung, Testspezifikation, Testdurchführung, Testbewertung und Testergebnisberichten.
- Selbstorganisierend: Das Team bestimmt selbstständig was aus dem Product-Backlog als nächstes abgearbeitet wird.
- Ortsverbundenheit: Tester sitzen mit den Entwicklern und dem Product-Owner zusammen.
- Zusammenarbeit: Tester arbeiten mit ihren Teammitgliedern, anderen Teams, den Interessenvertretern, dem Product-Owner und dem Scrum-Master zusammen.
- Bevollmächtigt: Technische Entscheidungen bezüglich Design und Test werden vom Team als Ganzes getroffen (Programmierer, Tester und Scrum-Master), in Zusammenarbeit mit dem Product-Owner und wenn nötig mit anderen Teams.
- Engagiert: Der Tester ist engagiert im Hinterfragen und Bewerten des Produktverhaltens und der Produktcharakteristika, in Bezug auf die Erwartungen und Bedürfnisse der Kunden und Nutzer.
- Transparent: Der Vorgang des Programmierens und Testens ist auf dem Agile Task-Board sichtbar (siehe Abschnitt 2.2.1)
- Glaubwürdig: Der Tester muss die Glaubwürdigkeit der Teststrategie, seine Implementierung und Ausführung sicherstellen, da die Interessenvertreter ansonsten den Testergebnissen nicht trauen werden. Oft geschieht dies durch regelmäßige Berichte über den Testprozess an die Interessenvertreter.

- Offen für Rückmeldungen: Rückmeldungen sind ein wichtiger Aspekt für den Erfolg jedes Projekts, insbesondere aber in agilen Projekten. Retrospektiven ermöglichen es den Teams aus Erfolgen und Misserfolgen zu lernen.
- Flexibel: Tester müssen auf Veränderungen reagieren können.

Diese Best Practices maximieren die Wahrscheinlichkeit erfolgreichen Testens in Scrum Projekten.

Sprint Null

Der Sprint Null ist die erste Iteration des Projektes, in der viele Vorbereitungen getroffen werden (siehe Abschnitt 1.2.5) Der Tester arbeitet mit dem Team zusammen an den folgenden Maßnahmen:

- Identifikation des Projektumfangs (d.h. Anfertigung des Product-Backlogs)
- Erstellen einer initialen Systemarchitektur und ggf. erster Prototypen
- Planung, Erwerb und Installation der notwendigen Werkzeuge (z. B. für das Testmanagement, Fehlermanagement, die Testautomatisierung und die kontinuierliche Integration)
- Erstellen einer initialen Teststrategie für alle Teststufen, die (unter anderem) auf folgendes abzielen: Testumfang, technische Risiken, Testarten (siehe Abschnitt 3.1.3) und Überdeckungsziele
- Durchführung einer initialen Qualitätsrisikoanalyse (siehe Abschnitt 3.2.1)
- Definition von Metriken zur Messung des Testfortschritts und zur Produktqualität
- Spezifikation der Definition-of-Done
- Festlegen der Struktur des Task-Boards und initiales "Befüllen" des Boards (siehe Abschnitt 2.2.1)
- Definition des Zeitpunktes zu dem die Tests beendet werden sollen, bevor das System an den Kunden geliefert wird

Der Sprint Null legt die Richtung dafür fest, was in den Tests erreicht werden soll und wie dies in den Tests während der Sprints erreicht werden soll.

Integration

In agilen Projekten ist das Ziel dem Kunden kontinuierlich einen Mehrwert zu liefern (vorzugsweise mit jedem Sprint). Um dies sicherzustellen, sollte die Integration sowohl das Design als auch die Tests berücksichtigen. Um eine kontinuierliche Qualität der gelieferten Funktionalitäten und Charakteristika zu ermöglichen, ist es wichtig, alle Abhängigkeiten zwischen Funktionen und Features zu identifizieren und die Regressionstests sorgfältig auszuwählen.

Testplanung

Da das Testen vollständig in das agile Team integriert ist, sollte die Testplanung während der Releaseplanung beginnen und während jedes Sprints aktualisiert werden. Die Testplanung für das Release wie auch für jeden Sprint sollte sich auf die Themen konzentrieren, die in Abschnitt 1.2.5 erläutert wurden.

Die Sprintplanung liefert als Ergebnis eine Reihe von Aufgaben (Tasks), die ins Task-Board eingefügt werden, in dem jede Aufgabe nicht mehr als ein oder zwei Arbeitstagen umfassen sollte. Darüber hinaus sollten alle Testprobleme nachverfolgt werden, um einen stetigen Testablauf zu gewährleisten.

Agile Testpraktiken

Folgende Praktiken können für Tester in einem Scrum Team von Nutzen sein:

- Pairing: Zwei Teammitglieder (z. B. ein Tester und ein Entwickler, zwei Tester oder ein Tester und ein Product-Owner) setzen sich zusammen, um gemeinsam eine Test- oder eine andere Sprintaufgabe zu bearbeiten.
- Inkrementelles Test Design: Testfälle und Chartas (vgl. 3.3.4) werden schrittweise aus User-Stories und anderen Testgrundlagen aufgebaut. Dabei wird mit einfachen Tests begonnen und man geht dann über zu komplexeren Tests.
- Mind-Mapping: Mind-Mapping ist ein nützliches Werkzeug für das Testen [Crispin08]. Beispielsweise können Tester Mind-Mapping nutzen, um zu bestimmen, welche Testsitzungen durchzuführen sind, um Teststrategien zu demonstrieren und um Testdaten zu beschreiben.

Diese Praktiken werden zusätzlich zu denen angewendet, die in diesem Lehrplan und im Kapitel 4 des Lehrplans zum Foundation Level [ISTQB_FL_SYL] beschrieben werden.

3.2 Qualitätsrisiken bestimmen und Testaufwände schätzen

Ein typisches Ziel für das Testen in allen Projekten, agilen oder traditionellen, ist es, das Risiko von Produktqualitätsproblemen vor der Inbetriebnahme auf ein akzeptables Niveau zu reduzieren. Tester in agilen Projekten können die gleichen Techniken nutzen, die auch in traditionellen Projekten genutzt werden, um Qualitätsrisiken (oder Produktrisiken) zu identifizieren, das zugehörige Risikoniveau abzuschätzen, den Aufwand einzuschätzen, der notwendig ist, um diese Risiken ausreichend zu reduzieren und dann diese Risiken durch Testentwurf, Implementierung und Durchführung zu mildern. Allerdings müssen diese Techniken in einigen Bereichen angepasst werden, um den kurzen Iterationen und dem Grad an Veränderungen in agilen Projekten gerecht zu werden.

3.2.1 Die Produktqualitätsrisiken in agilen Projekten einschätzen

Einige der vielen Herausforderungen im Test sind die geeignete Auswahl, Zuweisung und Priorisierung von Testbedingungen. Das beinhaltet eine angemessene Aufwandsschätzung für das Überdecken jeder Testbedingung durch Tests und für die Definition der Reihenfolge dieser Tests mit dem Ziel, die Effektivität und Effizienz der Tests zu optimieren. Die Techniken der Risikoidentifizierung, -analyse und -begrenzung können von Testern in agilen Teams genutzt werden, um den Testumfang festzulegen. Eine hohe Anzahl interagierender Randbedingungen und Variablen kann hierbei das Eingehen entsprechender Kompromisse erfordern.

Unter Risiko versteht man die Möglichkeit eines negativen oder nicht wünschenswerten Ergebnisses oder Ereignisses. Die Risikostufe wird dadurch festgelegt, dass die Wahrscheinlichkeit des Zutreffens und die Wirkung des Risikos eingeschätzt werden. Wenn der erste Effekt des potenziellen Problems die Produktqualität betrifft, werden diese potenziellen Probleme als Qualitätsrisiken oder Produktrisiken eingestuft. Wenn der erste Effekt des potenziellen Problems den Projekterfolg betrifft, werden diese potenziellen Probleme als Projektrisiken oder Planungsrisiken eingestuft [Black07]; [ISTQB_FL_SYL], Abschn. 5.5.1; [vanVeenendaal12].

In agilen Projekten findet die Qualitätsrisikoanalyse an zwei Stellen statt.

- Releaseplanung: Fachbereichsvertreter, die die Features des Release kennen, liefern einen groben Überblick über die Risiken und das gesamte Team, einschließlich der Tester kann zur Risikoidentifikation und –beurteilung beitragen.
- Iterationsplanung: Das gesamte Team identifiziert und bewertet die Qualitätsrisiken.

Beispiele für Qualitätsrisiken eines Systems sind u. a.:

- Falsche Berechnungen in Berichten (ein funktionales Risiko, das sich auf die Genauigkeit bezieht)

- Langsame Reaktion auf Nutzerangaben (ein nicht-funktionales Risiko, das sich auf die Effizient und Antwortzeit bezieht)
- Schwierigkeit, Bildschirme und Felder zu verstehen (ein nicht-funktionales Risiko, das sich auf Benutzbarkeit und Verständlichkeit bezieht)

Wie bereits zuvor erwähnt, beginnt eine Iteration mit der Iterationsplanung, welche durch die Darstellung der Aufgaben auf dem sog. Task-Board abgeschlossen wird. Diese Aufgaben (Tasks) können teilweise auf Basis der Stufe der mit ihnen verbundenen Qualitätsrisiken priorisiert werden. Aufgaben mit höheren verbundenen Risiken sollten früher beginnen und mehr Testaufwand enthalten. Aufgaben mit geringeren Risiken sollten später beginnen und weniger Testaufwand enthalten.

Ein Beispiel dafür, wie der Prozess der Qualitätsrisikoanalyse in einem agilen Projekt während der Iterationsplanung verlaufen kann wird in den folgenden Schritten dargestellt:

1. Besprechung aller Teammitglieder, einschließlich der Tester
2. Auflistung aller Backlog-Themen für die aktuelle Iteration (z. B. auf einem Whiteboard)
3. Identifikation der Qualitätsrisiken, die mit jedem Thema verbunden sind, unter Berücksichtigung aller für das Produkt relevanten Qualitätsmerkmale.
4. Beurteilung jedes identifizierten Risikos. Dies beinhaltet zwei Maßnahmen: Kategorisierung des Risikos und Bestimmung der Risikostufe auf Grundlage der Wirkung und der Wahrscheinlichkeit von Fehlern.
5. Bestimmen des Ausmaßes an Tests in Abhängigkeit von der Risikostufe.
6. Auswahl der passenden Testtechniken, um jedes Risiko zu mindern. Grundlage hierfür sind das Risiko, die Risikostufe und die relevanten Qualitätsmerkmale.

Der Tester entwirft dann die Tests zur Minderung der Risiken, er implementiert sie und führt sie auch durch. Das beinhaltet die Gesamtheit der Features, Verhaltensweisen, Qualitätsmerkmalen und Eigenschaften, die die Zufriedenheit der Kunden, Nutzer und Interessenvertreter betreffen.

Während des gesamten Projekts sollte das Team sich zusätzlicher Informationen bewusst sein, die die Risiken oder das Risikoniveau der bekannten Qualitätsrisiken verändern könnten. Regelmässige Anpassungen der Risikoanalyse, die sich auch in Anpassungen der Tests niederschlagen, sollten stattfinden. Anpassungen beinhalten die Identifikation neuer Risiken, die Neu-Beurteilung des Niveaus bestehender Risiken und die Beurteilung der Effektivität der Risikominderungsaktivitäten.

Qualitätsrisiken können auch vor Beginn der Testdurchführung gemindert werden. Beispielsweise kann das Projektteam, wenn während der Risikoidentifikation Probleme mit der User-Story entdeckt werden, als abschwächende Strategie die User-Stories im Detail überprüfen.

3.2.2 Schätzung des Testaufwands auf Basis des Inhalts und des Risikos

Während der Releaseplanung schätzt das agile Team den Aufwand, der benötigt wird, um das Release zu vervollständigen. Die Schätzung betrifft auch den Testaufwand. Eine verbreitete Schätztechnik, die in agilen Projekten verwendet wird, ist Planungspoker (Planning Poker), eine Technik, die auf Konsens basiert. Der Product-Owner oder der Kunde lesen den Schätzern eine User-Story vor. Jeder Schätzer hat einen Satz Karten mit Werten ähnlich der Fibonacci-Folge (d.h. 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …) oder in einer ähnlichen frei gewählten Abfolge (z. B. T-Shirt-Größen von XS bis XXL). Die Werte repräsentieren die Anzahl der Story-Punkte, Aufwandstage oder andere Einheiten, in denen das Team seine Schätzung abgibt. Die Fibonacci-Folge wird empfohlen, da die Zahlen in der Sequenz berücksichtigen, dass die Unsicherheit proportional mit der Größe der Story steigt. Eine hohe Schätzung bedeutet üblicherweise, dass die Story nicht leicht verständlich ist oder in mehrere kleinere Stories aufgeteilt werden sollte.

Die Schätzer diskutieren das Feature und klären Fragen, wenn nötig, mit dem Product-Owner. Aspekte wie Entwicklungs- und Testaufwand, Komplexität der Story und der Testumfang spielen eine Rolle für die Schätzung. Daher ist es ratsam, zusätzlich zur vom Product-Owner festgelegten Priorität auch die Risikostufe des Backlog-Themas einzubeziehen, bevor die Planungspokersitzung begonnen wird. Wenn das Feature vollständig besprochen wurde, wählt jeder Schätzer geheim eine Karte aus, die seine Schätzung beschreibt. Alle Karten werden gleichzeitig offengelegt. Wenn alle Schätzer den gleichen Wert gewählt haben, wird dies die offizielle Schätzung. Falls das nicht der Fall ist, diskutieren die Schätzer die Unterschiede der Schätzungen. Dann wird die Pokerrunde wiederholt, bis eine Einigung erreicht ist. Dies geschieht entweder durch Konsens oder durch die Anwendung bestimmter Regeln (z. B. Nutzung des Medians, Nutzung des höchsten Wertes), die die Anzahl der Pokerrunden begrenzen. Diese Diskussionen stellen eine verlässliche Schätzung des Aufwands für die restlichen Product-Backlog-Einträge sicher, die der Product-Owner erwartet. Sie helfen dabei, das gemeinsame Wissen über die zu erledigenden Arbeiten im Team zu erhöhen [Cohn04].

3.3 Techniken in agilen Projekten

Viele der Testtechniken und Teststufen, die für traditionelle Projekte gelten, werden auch in agilen Projekten genutzt. Allerdings gibt es für agile Projekte einige spezifische Erwägungen und Varianten bezüglich der Testtechniken, Terminologie und Dokumentation, die in Betracht gezogen werden sollten.

3.3.1 Abnahmekriterien, angemessene Überdeckung und andere Informationen für das Testen

In agilen Projekten werden die Anfangsanforderungen zu Beginn des Projekts als User-Stories in einem priorisierten Backlog niedergeschrieben. Anfangsanforderungen sind kurz und folgen üblicherweise einem vorab definierten Format (siehe Abschnitt 1.2.2). Nicht-funktionale Anforderungen, wie Benutzbarkeit und Performanz, sind ebenfalls wichtig und können als eigene User-Stories definiert werden oder an andere funktionale User-Stories angeknüpft werden. Nicht-funktionale Anforderungen können einem vorab definierten Format oder Standard, wie z. B. [ISO25000], oder einem industriebezogenen Standard entsprechen.

Die User-Stories sind eine wichtige Testbasis. Auch andere Informationen können als Testbasis dienen:

- Erfahrung aus aktuellen oder vorangegangenen Projekten
- Bestehende Funktionen, Features und Qualitätsmerkmale des Systems
- Code, Architektur und Entwurf
- Nutzerprofile (Kontext, Systemkonfiguration und Nutzerverhalten)
- Informationen zu Fehlern aus aktuellen und vorangegangenen Projekten
- Eine Kategorisierung der Fehler in einer Fehlertaxonomie
- Anwendbare Standards (z. B. [DO-178B] für Avionik Software)
- Qualitätsrisiken (siehe Abschnitt 3.2.1)

In jeder Iteration erstellen Entwickler Code, mit dem die in der User-Story beschriebenen Funktionen und Features mit den relevanten Qualitätsmerkmalen implementiert werden. Dieser Code wird durch Abnahmetests verifiziert und validiert. Abnahmekriterien sollten je nach Relevanz die folgenden Themen abdecken, um testbar zu sein [Wiegers13]:

- Funktionales Verhalten: Das von außen zu beobachtende Verhalten mit Nutzeraktionen als Input. Dies muss unter den relevanten Konfigurationen geprüft werden.
- Qualitätsmerkmale: Wie das System das spezifische Verhalten ausführt. Die Eigenschaften werden häufig auch als Qualitätsattribute oder nicht-funktionale Anforderungen bezeichnet.

Verbreitete Qualitätsmerkmale sind Performanz, Verlässlichkeit, Benutzbarkeit, usw. (siehe auch [ISO25000])
- Szenarios (Use Cases): Eine Abfolge von Aktionen zwischen einem externen Akteur (oft ein Nutzer) und dem System, um ein bestimmtes Ziel zu erreichen oder eine bestimmte Aufgabe zu erfüllen.
- Geschäftsprozessregeln: Aktivitäten, nur unter bestimmten Bedingungen im System ausgeführt werden können, wobei die Bedingungen durch externe Vorgehensweisen und Beschränkungen bestimmt sind. (Beispielsweise die Verfahrensweise die von einem Versicherungsunternehmen verwendet wird, um Versicherungsansprüche zu behandeln.)
- Externe Schnittstellen: Beschreibungen der Verbindungen zwischen dem zu entwickelnden System und der Außenwelt. Externe Schnittstellen können in verschiedene Arten unterteilt werden (Benutzerschnittstelle, Programmierschnittstelle, etc.)
- Einschränkungen: Jegliche Entwurf- und Implementierungsbedingungen, die die Möglichkeiten der Entwickler begrenzen. Geräte mit eingebetteter Software müssen oft physische Grenzen wie Größe, Gewicht und Schnittstellen berücksichtigen.
- Datendefinitionen: Der Kunde kann das Format, den Datentyp, erlaubte Werte und Standardwerte sowie die Anordnung des Datensatzes in einer komplexen Unternehmensdatenstruktur beschreiben (z. B. die Postleitzahl in US-Adressen).

Zusätzlich zu den User-Stories und den mit ihnen verbundenen Abnahmekriterien sind auch die folgenden weiteren Informationen für Tester von Bedeutung:

- Wie das System arbeiten und genutzt werden soll,
- Systemschnittstellen, die genutzt werden können/die zugänglich sind, um das System zu testen,
- ob die aktuell verfügbare Werkzeugunterstützung ausreicht,
- ob der Tester über genügend Wissen und Fähigkeiten verfügt, um die notwendigen Tests durchzuführen.

Tester erkennen oft während der Iterationen den Bedarf für weitere Informationen (z. B. Codeüberdeckung). Sie sollten dann mit den anderen Teammitgliedern zusammenarbeiten, um diese Informationen zu erhalten. Relevante Information spielt eine Rolle bei der Beurteilung ob eine bestimmte Aktivität als erledigt angesehen werden kann. Das Konzept der Definition-of-Done ist für agile Projekte wesentlich. Es wird verschiedenartig verwendet, wie in den Folgeabschnitten dargelegt wird.

Teststufen

Jede Teststufe hat ihre eigene Definition-of-Done. Die folgende Liste zeigt Beispiele, die auf den unterschiedlichen Teststufen relevant sein können.

Unittests:

- 100% Entscheidungsüberdeckung wo dies möglich ist und mit Reviews der nicht abgedeckten Wege.
- Statische Analyse über den gesamten Code.
- Keine ungelösten schweren Fehler (Rangfolge gemäß Risiko und Schwere).
- Keine bekannte inakzeptable technische Schuld im Entwurf oder im Code [Jones11].
- Reviews des gesamten Codes, der Unittests und der Ergebnisse der Unittests sind abgeschlossen.
- Alle Unittests sind automatisiert und vollständig durchgeführt.
- Wichtige Qualitätsmerkmale befinden sich innerhalb der vereinbarten Grenzen (z. B. Performanz).

Integrationstests:

- Alle funktionalen Anforderungen sind getestet, inklusive Positiv- und Negativtests. Die Anzahl der Tests basiert dabei auf der Größe, Komplexität und der Kritikalität der Applikation.
- Alle Schnittstellen zwischen den Komponenten sind getestet.
- Alle Qualitätsrisiken sind gemäß des vereinbarten Ausmasses an Tests abgedeckt.
- Keine ungelösten schweren Fehler (priorisiert gemäß Risiko und Bedeutung).
- Alle gefundenen Fehler sind dokumentiert.
- Alle Regressionstests sind automatisiert, soweit möglich. Alle automatisierten Tests sind in einer gemeinsamen Ablage gespeichert.

Systemtests:

- End-to-End Tests der User-Stories, Features und Funktionen sind durchgeführt und dokumentiert.
- Alle Nutzeridentitäten sind abgedeckt.
- Die wichtigsten Qualitätsmerkmale des Systems sind abgedeckt (z. B. Performanz, Robustheit, Zuverlässigkeit).
- Tests werden in einer produktionsähnlichen Umgebung durchgeführt, in der die gesamte Hardware und Software für alle unterstützten Konfigurationen soweit möglich zur Verfügung stehen.
  Alle Qualitätsrisiken sind gemäß des vereinbarten Testumfangs abgedeckt.
- Alle Regressionstests sind soweit möglich automatisiert und alle automatisierten Tests sind in einer gemeinsamen Ablage hinterlegt
- Alle gefundenen Fehlerzustände sind gemeldet und eventuell behoben.
- Alle wesentlichen Fehlerzustände (priorisiert gemäß Risiko und Bedeutung) wurden behoben.

User-Story

Die Definition-of-Done für die User-Stories kann anhand folgender Kriterien bestimmt werden:

- Die für die Iteration ausgewählten User-Stories sind vollständig, vom Team verstanden und haben detaillierte, testbare Abnahmekriterien.
- Alle Elemente der User-Story sind spezifiziert und einem Review unterzogen worden. Die Abnahmetests der User-Stories sind entworfen.
- Die notwendigen Aufgaben für die Implementierung und das Testen der ausgewählten User-Stories, sind identifiziert und vom Team geschätzt worden.

Feature

Die Definition-of-Done für Features, die mehrere User-Stories oder Epics umfassen, können folgendes beinhalten:

- Jede wesentliche User-Story und ihre Abnahmekriterien sind vom Kunden definiert und abgenommen.
- Der Entwurf ist vollständig
- Der Code ist vollständig
- Unittests wurden durchgeführt und haben den definierten Überdeckungsgrad erreicht.
- Integrations- und Systemtests für das Feature wurden anhand der definierten Überdeckungskriterien durchgeführt.
- Alle schweren Fehler sind korrigiert.
- Die Feature-Dokumentation samt Release-Notes, Bedienungsanleitungen für Nutzer und Online Hilfe-Funktionen ist vollständig.

Iteration

Die Definition-of-Done für die Iteration kann folgendes beinhalten:

- Alle Features der Iteration sind fertig entwickelt und gemäß der Feature-Level Kriterien individuell getestet.
- Jegliche nicht-kritischen Fehler, die während einer Iteration nicht behoben wurden, sind dem Product-Backlog hinzugefügt und priorisiert worden.
- Die Integration aller Features der Iteration ist erfolgt und getestet.
- Die Dokumentation ist geschrieben, einem Review unterzogen und abgenommen worden.

Zu diesem Zeitpunkt ist die Software potenziell auslieferbar da die Iteration erfolgreich abgeschlossen ist, aber nicht alle Iterationen werden released.

Release

Die Definition-of-Done für ein Release, das mehrere Iterationen umfassen kann, kann die folgenden Bereiche abdecken:

- Überdeckung: Alle relevanten Testbasiselemente für den gesamten Inhalt des Release sind durch Tests abgedeckt. Die Angemessenheit der Überdeckung wird dadurch bestimmt, was neu oder geändert ist, sowie durch die Komplexität, Größe und das damit verbundene Risiko für einen Misserfolg.
- Qualität: Die Fehlerhäufung (z. B. wie viele Fehler werden pro Tag oder pro Transaktion gefunden), die Fehlerdichte (z. B. die Anzahl der gefundenen Fehler im Vergleich zur Anzahl der User-Stories und/oder Qualitätsattribute), die geschätzte Anzahl verbleibender Fehler bewegt sich in einem akzeptablen Rahmen, die Folgen der nicht behobenen und verbleibenden Fehler (z. B. die Schwere und Priorität) sind verstanden und akzeptabel, das Restrisiko, das mit jedem identifizierten Qualitätsrisiko verbunden ist, ist verstanden und akzeptabel.
- Zeit: Wenn das zuvor festgelegte Lieferdatum erreicht ist, müssen die geschäftlichen Belange bezüglich der Veröffentlichung oder Nicht-Veröffentlichung abgewägt werden.
- Kosten – Die geschätzten Lebenszykluskosten sollten verwendet werden, um die Rendite des gelieferten Systems zu berechnen (d.h. die berechneten Entwicklungs- und Wartungskosten sollten weit geringer sein als der erwartete Gesamtumsatz des Produkts). Der Hauptteil der Lebenszykluskosten entsteht oftmals durch die Wartung des Produkts nach dem Release, da eine gewisse Anzahl von Fehlern unbemerkt in die Produktion übernommen wird.

3.3.2 Anwendung der Abnahmetestgetriebenen Entwicklung

Die Abnahmetestgetriebene Entwicklung ist ein sog. Test-First-Ansatz. Die Testfälle werden vor Implementierung der User-Story erstellt. Die Testfälle werden vom gesamten agilen Team erstellt, [Gärtner13], und können sowohl manuell als auch automatisiert sein.

Im ersten Schritt wird die User-Story in einem Spezifikations-Workshop von Entwicklern, Testern und Fachbereichsvertretern analysiert, diskutiert und geschrieben. Jegliche Unvollständigkeiten, Mehrdeutigkeiten oder Fehler in der User-Story werden in diesem Prozess korrigiert.

Im nächsten Schritt werden die Tests erstellt. Das kann gemeinsam im Team oder vom Tester allein gemacht werden. Auf jeden Fall beurteilt eine unabhängige Person wie ein Fachbereichsvertreter die Tests. Die Tests sind Beispiele, die die spezifischen Eigenschaften der User-Story beschreiben. Diese Beispiele helfen dem Team, die User-Story korrekt zu implementieren. Da Beispiele und Tests dasselbe sind, werden diese Begriffe oft synonym verwendet. Die Arbeit beginnt mit Basisbeispielen und offenen Fragen.

Üblicherweise handelt es sich bei den ersten Tests um Positivtests, die das Standardverhalten ohne Ausnahme- oder Fehlerbedingungen testen. Sie umfassen die Abfolge der Aktivitäten, die ausgeführt werden, wenn alles nach Plan verläuft. Nachdem die Positivtests abgeschlossen sind, sollte das Team Negativtests schreiben und auch nicht-funktionale Attribute abdecken (z. B. Performanz, Benutzbarkeit). Tests werden so ausgedrückt, dass jeder Interessenvertreter sie versteht, d.h. die notwendigen Vorbedingungen, falls es welche gibt, die Inputs und die damit verbundenen Ergebnisse werden in normaler Sprache ausgedrückt.

Die Beispiele müssen alle Eigenschaften der User-Story abdecken und sollten nichts hinzufügen. Das bedeutet, dass kein Beispiel für einen Aspekt existieren sollte, der in der User-Story nicht dokumentiert ist. Darüber hinaus sollten nicht zwei Beispiele dasselbe Merkmal der User-Story beschreiben.

3.3.3 Funktionaler und Nicht-funktionaler Black-Box-Testentwurf

In agilen Projekten werden viele Tests von den Testern entwickelt während die Entwickler zeitgleich programmieren. So wie die Entwickler auf Grundlage der User-Stories und Abnahmekriterien entwickeln, so erstellen die Tester auch die Tests auf Grundlage der User-Stories und ihrer Abnahmekriterien. Tester können für die Erstellung dieser Tests traditionelle Black-Box-Testentwurfsverfahren anwenden, wie Äquivalenzklassenbildung, Grenzwertanalyse, Entscheidungstabellen, und zustandsbasierte Tests. Beispielsweise könnte die Grenzwertanalyse verwendet werden, um Testwerte auszuwählen, wenn ein Kunde in der Anzahl der Artikel begrenzt ist, die er zum Kauf auswählen kann.

Einige Tests, wie explorative Tests und andere erfahrungsbasierte Tests, werden erst später, während der Testdurchführung erstellt. (Siehe dazu auch Abschnitt 3.3.4.)

In vielen Situationen können nicht-funktionale Anforderungen als User-Story dokumentiert sein. Black-Box-Testentwurfsverfahren (wie die Grenzwertanalyse) können auch verwendet werden, um Tests für nicht-funktionale Qualitätsmerkmale zu erstellen. Die User-Story kann Performanz- oder Zuverlässigkeitsbedingungen enthalten. Beispielsweise darf die Ausführung einer Aktion ein Zeitlimit nicht überschreiten oder die Zahl der Durchführungen darf nur bis zu einer begrenzten Zahl von Fehleingaben erfolgen.

Für mehr Informationen über den Einsatz von Black-Box-Testentwurfsverfahren verweisen wir auf den Lehrplan zum Foundation Level [ISTQB_FL_SYL], Abschnitt 4.3, und auf den Lehrplan für den Advanced Level Test Analyst [ISTQB_ALTA_SYL], Abschnitt 3.2.

3.3.4 Exploratives Testen und agiles Testen

Exploratives Testen ist in agilen Projekten wichtig wegen der begrenzten Zeit, die für die Testanalyse zur Verfügung steht und der begrenzten Detailgenauigkeit der User-Stories. Um die besten Ergebnisse zu erzielen, sollte exploratives Testen mit anderen erfahrungsbasierten Verfahren als Teil einer reaktiven Teststrategie kombiniert werden. Damit kombiniert werden können weitere Teststrategien wie analytisches risikobasiertes Testen, analytisches anforderungsbasiertes Testen, modellbasiertes Testen und regressionsvermeidendes Testen. Teststrategien und die Kombination von Teststrategien werden ebenfalls im Lehrplan zum Foundation Level [ISTQB_FL_SYL], Abschn. 5.2.6, behandelt.

Beim explorativen Testen finden Testentwurf und Testdurchführung zur selben Zeit statt. Beides wird durch eine vorbereitete Test-Charta unterstützt. Eine Test-Charta liefert die Testbedingungen, die während einer zeitlich begrenzten Testsitzung abgedeckt werden müssen (Jeweils basierend auf den Ergebnissen der zuletzt durchgeführten Tests werden die nachfolgenden Testfälle entworfen). Für den

Testentwurf können die gleichen White-Box- und Black-Box-Verfahren wie bei vorentworfenen Tests verwendet werden.

Eine Test-Charta kann die folgenden Informationen enthalten:

- Akteur: der vorgesehene Nutzer des Systems
- Zweck: Angabe von Testziel und Testbedingungen
- Setup: Was muss vorhanden sein, um die Testdurchführung zu beginnen
- Priorität: der relative Stellenwert dieser Charta, auf Grundlage der Priorität der zugehörigen User-Story oder Risikostufe
- Referenz: Spezifikationen (z. B. User-Story), Risiken, oder andere Informationsquellen
- Daten: Jegliche Daten, die benötigt werden, um die Charta durchzuführen
- Aktivitäten: Eine Liste von Ideen, was der Akteur mit dem System vielleicht tun will (z. B. als „Super User“ ins System einloggen) und was für Tests interessant wären (sowohl Positiv- als auch Negativtests)
- Orakel Notizen: Wie soll das Produkt beurteilt werden, um zu bestimmen, was korrekte Ergebnisse sind (z. B. um zu erfassen, was auf dem Bildschirm passiert und dies mit dem zu vergleichen, was im Nutzerhandbuch steht)
- Variationen: alternative Aktionen und Auswertungen, um die Ideen zu ergänzen, die unter Aktivitäten beschrieben sind.

Für die Organisation des explorativen Testens kann eine Methode genutzt werden, die sich sitzungsbasiertes Testmanagement (session-based testing) nennt. Eine Sitzung ist definiert als eine ununterbrochene Zeitspanne des Testens, die z. B. 60 Minuten lang sein kann.

Testsitzungen können folgendes beinhalten:

- Überblickssitzung (um zu lernen, wie es funktioniert)
- Analysesitzung (Bewertung der Funktionalität oder Eigenschaften)
- Genaue Überdeckung (Ausnahmefälle, Szenarien, Interaktionen)

Die Qualität der Tests hängt von der Fähigkeit des Testers ab, relevante Fragen darüber zu stellen, was getestet werden soll. Beispiele könnten die folgenden sein:

- Was ist das Wichtigste, das über das System herauszufinden ist?
- Auf welche Art und Weise kann das System versagen?
- Was passiert, wenn…?
- Was sollte passieren, wenn…?
- Werden die Bedürfnisse, Anforderungen und Erwartungen des Kunden erfüllt?
- Ist das System installationsfähig (und wenn nötig deinstallationsfähig) für alle unterstützten Upgrades

Während der Testdurchführung nutzt der Tester Kreativität, Intuition, Erkenntnisvermögen und Fachkenntnisse, um mögliche Probleme der Software zu finden. Der Tester muss auch ein gutes Verständnis der Software unter Testbedingungen besitzen sowie Kenntnisse über den Fachbereich, darüber wie die Software genutzt wird und wie zu bestimmen ist, wann die Software fehlschlägt.

Eine Reihe von Heuristiken kann für die Tests genutzt werden. Eine Heuristik kann dem Tester eine Anleitung geben, wie die Tests durchgeführt werden und wie die Ergebnisse zu bewerten sind [Hendrickson]. Beispiele hierfür sind:

- Grenzen
- CRUD (Create, Read, Update, Delete = Erstellen, Lesen, Aktualisieren, Löschen)

- Konfigurationsvariationen
- Unterbrechungen (z. B. Abmelden, Schliessen oder Neustarten)

Es ist wichtig, dass der Tester den Prozess so genau wie möglich dokumentiert. Die folgende Liste gibt Beispiele für Informationen, die dokumentiert werden sollten:

- Testüberdeckung: Welche Eingabewerte wurden genutzt, wie viel wurde abgedeckt, und wie viel muss noch getestet werden.
- Bewertungsnotizen: Beobachtungen während des Testens, sind das System und das getestete Feature stabil, wurden Fehler gefunden, was ist als nächster Schritt als Folge der aktuellen Beobachtungen geplant und gibt es weitere Ideen?
- Risiko-/Strategieliste: Welche Risiken wurden abgedeckt und welche verbleiben von den wichtigsten, wird die ursprüngliche Strategie weiterverfolgt, sind Änderungen nötig?
- Probleme, Fragen und Anomalien: Jegliches unerwartetes Verhalten, jegliche Fragen bezüglich der Effizienz des Ansatzes, jegliche Bedenken bezüglich der Ideen/Testversuche, Testumgebung, Testdaten, Missverständnisse bezüglich der Funktion, des Testskripts oder des Systems unter Testbedingungen.
- Tatsächliches Verhalten: Aufzeichnung des tatsächlichen Verhaltens des Systems, das gespeichert werden muss (z. B. Video, Screenshots, Ergebnisdateien)

Die aufgezeichneten Informationen sollten in einem Statusmanagementwerkzeug erfasst und/oder zusammengefasst werden (z. B. Testmanagementwerkzeuge, Taskmanagementwerkzeuge, das Task-Board). Dies sollte in einer Art und Weise geschehen, die es Interessenvertretern erleichtert, den aktuellen Status aller durchgeführten Tests zu verstehen.

3.4 Werkzeuge in agilen Projekten

Die im Lehrplan zum Foundation Level [ISTQB_FL_SYL], Kap. 6, beschriebenen Werkzeuge sind relevant und werden von Testern in agilen Teams ebenfalls genutzt. Nicht alle Werkzeuge werden auf dieselbe Art und Weise genutzt und einige Werkzeuge haben eine größere Relevanz in agilen als in traditionellen Projekten.

Ein Beispiel: Obwohl die Werkzeuge für Testmanagement, Anforderungsmanagement und Fehler- und Abweichungsmanagement (Fehlernachverfolgungswerkzeuge) von agilen Teams genutzt werden können, entscheiden sich einige agile Teams für ein allumfassendes Werkzeug (z. B. Anwendungslebenszyklusmanagement oder Aufgabenmanagement), das Features bereitstellt, die für die agile Entwicklung relevant sind, wie z. B. Task-Boards, Burndown-Charts und User-Stories.

Konfigurationsmanagement-Werkzeuge sind wichtig für Tester in agilen Teams wegen der großen Zahl automatisierter Tests auf allen Stufen und der Notwendigkeit, die zugehörigen Testartefakte zu speichern und zu verwalten.

Zusätzlich zu den oben erwähnten Werkzeugen können Tester in agilen Projekten auch die Werkzeuge nutzen, die in den folgenden Abschnitten beschrieben werden. Diese werden vom gesamten Team genutzt, um die Zusammenarbeit und Informationsweitergabe zu unterstützen, was von zentraler Bedeutung für die agile Vorgehensweise ist.

3.4.1 Aufgabenmanagement- und Nachverfolgungswerkzeuge

Manche agile Teams nutzen physische Story/Task-Boards (z. B. Whiteboard, Pinnwand), um User-Stories, Tests und andere Aufgaben über jeden Sprint hinweg zu verwalten und nachzuverfolgen. Andere Teams nutzen Anwendungslebenszyklusmanagement- und Aufgabenmanagement-Software, darunter auch elektronische Task-Boards.

Diese Werkzeuge dienen den folgenden Zwecken:

- Aufzeichnung der Stories, ihrer relevanten Entwicklungs- und Testaufgaben, um sicherzustellen, dass während des Sprints nichts verloren geht.
- Erfassen der Aufwandschätzungen der Teammitglieder und automatische Berechnung des notwendigen Gesamtaufwands für die Implementation der Story, um effiziente Iterationsplanungssitzungen zu unterstützen.
- Verbindung von Entwicklungs- und Testaufgaben derselben Story, um ein vollständiges Bild des notwendigen Aufwands des Teams für die Implementierung der Story bereitzustellen.
- Erfassen des Aufgabenstatus nach jedem Update der Entwickler und Tester, was automatisch einen aktuell berechneten Snapshot des Status jeder Story, der Iteration und des gesamten Releases liefert.
- Bereitstellung einer visuellen Darstellung via Schaubildern und Dashboards des aktuellen Status jeder User-Story, der Iteration und des Releases. Auch in geografisch dezentralisierten Teams ermöglicht dies allen Interessenvertretern, den Status schnell zu überblicken und zu prüfen.
- Integration mit Konfigurationsmanagement-Werkzeugen, was die automatische Aufzeichnung von Code Check-ins und Builds gegenüber Aufgaben und, in einigen Fällen automatisierte Statusupdates für Aufgaben ermöglicht.

3.4.2 Kommunikations- und Informationsweitergabe-Werkzeuge

Neben E-Mail, Dokumenten und verbaler Kommunikation nutzen agile Teams oft drei weitere Werkzeuge, um die Kommunikation und Informationsweitergabe zu unterstützen: Wikis, Instant Messenger und Desktop Sharing.

Wikis ermöglichen es Teams, eine online Wissensbasis zu verschiedenen Projektaspekten zu schaffen und zu teilen, darunter die folgenden:

- Produktfeature-Diagramme, Feature-Diskussionen, Prototypdiagramme, Fotos von Whiteboard-Diskussionen und andere Informationen.
- Werkzeuge und/oder Techniken für Entwicklung und Test, die andere Teammitglieder nützlich finden.
- Metriken, Tabellen und Dashboards zum Produktstatus, die insbesondere dann wertvoll sind, wenn das Wiki mit anderen Werkzeugen verbunden ist, wie z. B. mit dem Build-Server und dem Aufgabenmanagementsystem, da das Werkzeug den Produktstatus automatisch aktualisieren kann.
- Besprechungen zwischen Teammitgliedern, ähnlich denen per Instant Messenger oder E-Mail, aber so, dass sie mit allen anderen Teammitgliedern geteilt werden.

Instant-Messenger, Telefonkonferenzen und Video-Chat-Werkzeuge haben die folgenden Vorteile:

- Sie ermöglichen direkte Kommunikation in Echtzeit zwischen den Teammitgliedern, insbesondere bei dezentralisierten Teams.
- Sie beziehen dezentralisierte Teams in Stand-up-Meetings ein.
- Sie vermindern Telefonrechnungen durch die Verwendung von Voice-over-IP-Technologie und beseitigen Kostenbeschränkungen, welche die Kommunikation zwischen Teammitgliedern in verteilten Umgebungen verringern könnten.

Desktop-Sharing und Erfassungswerkzeuge haben folgende Vorteile:

- In dezentralen Teams können damit Produktdemonstrationen, Code Reviews und sogar Pairing stattfinden.

- Aufzeichnen der Produktdemonstrationen am Ende jeder Iteration, welche dann in das Wiki des Teams eingefügt werden können.

Diese Werkzeuge sollten genutzt werden, um die direkte Kommunikation von Angesicht zu Angesicht in agilen Teams zu ergänzen und zu erweitern, nicht um sie zu ersetzen.

3.4.3 Werkzeuge für Build und Distribution

Wie bereits zuvor in diesem Lehrplan beschrieben, ist das tägliche Builden und die Distribution der Software eine Schlüsselmethode in agilen Teams. Das erfordert die Nutzung von Werkzeugen zur kontinuierlichen Integration (continuous integration) und zur Build-Distribution.

Die Nutzung, die Vorteile und die Risiken dieser Werkzeuge wurden in Abschnitt 1.2.4 bereits beschrieben.

3.4.4 Werkzeuge für das Konfigurationsmanagement

In agilen Teams können Konfigurationsmanagementwerkzeuge nicht nur zur Speicherung von Quellcode und automatisierten Testskripts verwendet werden, sondern auch zur Speicherung manueller Tests und anderer Arbeitsergebnisse. Sie werden häufig im selben Repository hinterlegt wie der Quellcode. Die Nachverfolgung, welche Versionen der Software mit genau welchen Testversionen getestet wurde ist somit möglich. Auch ermöglicht dies schnelle Änderungen ohne den Verlust historischer Informationen.

Die Versionsverwaltung erfolgt über Versionskontrollsysteme (Version Control System, VCS). Dabei werden die zentrale Versionsverwaltung (centralized VCS oder CVCS) und die verteilte Versionsverwaltung (distributed VCS oder DVCS) unterschieden.

Die Größe des Teams, seine Struktur, sein Einsatzort und die Anforderungen zur Integration mit anderen Werkzeugen bestimmen, welches Versionskontrollsystem für ein bestimmtes agiles Projekt geeignet ist.

3.4.5 Werkzeuge für Testentwurf, Implementierung und Durchführung

Die meisten der in agilen Entwicklungen eingesetzten Werkzeuge sind nicht neu oder speziell für agile Entwicklung gemacht. Doch besitzen sie wichtige Funktionen in Anbetracht der schnellen Veränderungen in agilen Projekten.

- Werkzeuge für den Testentwurf: Die Nutzung von Werkzeugen wie Mind-Maps sind beliebt, um schnell Tests für ein neues Feature zu umreißen und zu entwerfen.
- Testfallmanagement-Werkzeuge: Die Art der Testfallmanagement-Werkzeuge, die in agilen Projekten genutzt werden, können Teil des Werkzeugs für Anwendungslebenszyklusmanagement oder Aufgabenmanagement des gesamten Teams sein.
- Werkzeuge zur Testdatenvorbereitung und –erstellung: Werkzeuge, die Daten generieren, um die Datenbank einer Anwendung zu bestücken, sind sehr von Vorteil, wenn viele Daten und Datenkombinationen nötig sind, um die Anwendung zu testen. Diese Werkzeuge können auch dabei helfen, die Datenbankstruktur neu festzulegen, wenn das Produkt geändert wird, und um die Skripts zu Generierung der Daten zu refaktorieren. Dies ermöglicht eine schnelle Aktualisierung der Testdaten im Fall von Änderungen. Einige Werkzeuge zur Testdatenvorbereitung nutzen Produktionsdatenquellen als Rohmaterial und verwenden Skripts, um sensible Daten zu entfernen oder zu anonymisieren. Andere Werkzeuge können dabei helfen, große Daten ein- oder -ausgaben zu bewerten.

- Testdatenladewerkzeuge: Nachdem die Daten für die Tests erstellt sind, müssen sie in die Anwendung geladen werden. Manuelle Dateneingabe ist oft zeitaufwändig und fehleranfällig, aber es gibt Datenladewerkzeuge, die den Prozess verlässlich und effizient machen. Viele Datengenerierungswerkzeuge enthalten sogar bereits eine integrierte Komponente zum Laden der Daten. Es ist manchmal auch möglich, große Datenmengen über das Datenbankmanagementsystem hochzuladen.
- Automatisierte Testdurchführungswerkzeuge: Es gibt Testdurchführungswerkzeuge, die tendenziell besser für agiles Testen geeignet sind. Spezielle Werkzeuge für Test First Ansätze, wie verhaltensgetriebene Entwicklung, testgetriebene Entwicklung und Abnahmetestgetriebene Entwicklung gibt es sowohl von kommerziellen Anbietern als auch als Open Source Lösungen. Diese Werkzeuge ermöglichen es Testern und Fachbereichsspezialisten das erwartete Systemverhalten in Tabellen oder in einfacher Sprache unter Nutzung von Schlüsselwörtern zu formulieren.
- Werkzeuge für exploratives Testen: Für Tester und Entwickler nützlich sind Werkzeuge, die alle Aktivitäten in einer explorativen Testsitzung aufzeichnen und speichern. Wird ein Fehler gefunden, so ist ein solches Werkzeug von Nutzen, da alle Aktivitäten vor dem Auftreten des Fehlers nachvollziehbar aufgezeichnet sind. Dies ist für das Berichten des Fehlers an die Entwickler sinnvoll. Die Protokollierung der in einer explorativen Testsitzung durchgeführten Schritte kann von Vorteil sein, wenn der Test schließlich in der automatisierten Regressionstestsuite integriert ist.

3.4.6 Cloud Computing und Virtualisierungswerkzeuge

Virtualisierung ermöglicht es einer einzelnen physischen Ressource (z. B. einem Server), sich als viele separate, kleinere Ressourcen darzustellen. Wenn virtuelle Maschinen oder Cloud Realisierungen genutzt werden, stehen Teams eine größere Anzahl an Ressourcen (z. B. Servern) für Entwicklung und Tests zur Verfügung. Dies kann dazu beitragen, Verzögerungen zu verhindern, die sich aus der Tatsache ergeben können, dass auf physische Server gewartet werden muss. Einen neuen Server zur Verfügung zu stellen oder einen Server wiederherzustellen ist effizienter durch die Snapshot-Funktion, die in den meisten Virtualisierungswerkzeugen eingebaut ist. Ein Snapshot ist die Abspeicherung einer aktuellen Situation. Dieser kann jederzeit wiederhergestellt werden.

Einige Testmanagementwerkzeuge nutzen Virtualisierungstechniken, um einen Snapshot eines Servers in dem Moment anzufertigen, in dem ein Fehler entdeckt wird. Das ermöglicht es Testern, diesen Snapshot den Entwicklern, die den Fehler untersuchen, zur Verfügung zu stellen.

4. Referenzen

4.1 Standards

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
- [ISO25000] ISO/IEC 25000:2005, Software Engineering – Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2 ISTQB-Dokumente

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FA_OVIEW] ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011

4.3 Literatur

- [Aalst13] Leo van der Aalst, Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.
- [Anderson13] David Anderson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.
- [Baumgartner13] Manfred Baumgartner et al, "Agile Testing - Der agile Weg zur Qualität", Carl Hanser Verlag, 2013
- [Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.
- [Beck04] Kent Beck, Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.
- [Black07] Rex Black, “Pragmatic Software Testing,” John Wiley and Sons, 2007.
- [Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e,” Wiley, 2009.
- [Bucsics14] Thomas Bucsics et al, "Basiswissen Testautomatisierung - Konzepte, Methoden und Techniken", dpunkt Verlag, 2014
- [Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.
- [Cohn04] Mike Cohn, “User Stories Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.
- [Crispin08] Lisa Crispin, Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams,” Addison-Wesley Professional, 2008.
- [Gärtner13] Markus Gärtner, „ATDD in der Praxis: Eine praktische Einführung in die Akzeptanztest-getriebene Softwareentwicklung mit Cucumber, Selenium und FitNess“, dpunkt Verlag, 2013
- [Goucher09] Adam Goucher, Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O’Reilly Media, 2009.
- [Jeffries00] Ron Jeffries, Ann Anderson, Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.
- [Jones11] Capers Jones, Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.
- [Linz14] Tilo Linz, Testing in Scrum: A Guide for Software Quality Assurance in the Agile World, Rocky Nook, 2014.
- [Schwaber01] Ken Schwaber, Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.

- [vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.
- [Wiegers13] Karl Weigers, Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.

4.4 Agile Terminologie

Schlüsselwörter, die im ISTQB-Glossar zu finden sind, sind zu Beginn jedes Kapitels erwähnt. Für die gebräuchlichen agilen Begriffe haben wir auf die folgenden verlässlichen Internetquellen zurückgegriffen, die Definitionen anbieten.

http://guide.Agilealliance.org/
http://searchsoftwarequality.techtarget.com
http://whatis.techtargetcom/glossary
http://www.scrumalliance.org/

Wir empfehlen dem Leser, diese Seiten zu besuchen, wenn unbekannte Agile-bezogene Begriffe in diesem Dokument vorkommen. Diese Links waren zum Zeitpunkt der Veröffentlichung dieses Lehrplans aktiv.

4.5 Andere Referenzen

Die folgenden Referenzen weisen auf Informationen hin, die im Internet und an anderen Stellen verfügbar sind. Obwohl diese Referenzen zum Zeitpunkt der Veröffentlichung dieses Lehrplans geprüft wurden, kann das ISTQB keine Verantwortung dafür übernehmen, dass diese Referenzen auch weiterhin verfügbar sind.

- [Agile Alliance Guide] Diverse Mitwirkende, http://guide.Agilealliance.org/.
- [Agilemanifesto] Diverse Mitwirkende, www.agilemanifesto.org.
- [agileManifesto Prinzipien] http://agilemanifesto.org/iso/de/principles.html
- [Hendrickson]: Elisabeth Hendrickson, “Acceptance Test-driven Development,” testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview.
- [INVEST] Bill Wake, “INVEST in Good Stories, and SMART Tasks,” xp123.com/articles/invest-in-good-stories-and-smart-tasks.
- [Kubaczkowski] Greg Kubaczkowski, Rex Black, “Mission Made Possible,” www.rbcs-us.com/images/documents/Mission-Made-Possible.pdf.
- [Scrum Guide] Ken Schwaber, Jeff Sutherland, Herausgeber, “The Scrum Guide,” www.scrum.org.
- [Sourceforge] Diverse Mitwirkende, www.sourceforge.net.
- [Bolton] http://www.developsense.com/resources.html

Index