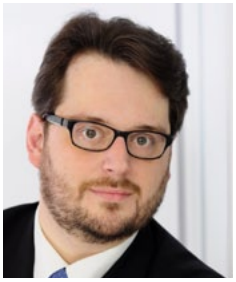
Jörn Koch
(joern.koch@treqs.org)
arbeitet seit 2001 als Coach bei der WPS – Workplace Solutions GmbH. Einer seiner Schwerpunkte ist, Unternehmen und Projekte dabei zu unterstützen, ihre Art des agilen Vorgehens zu finden und zu etablieren.



Sebastian Middeke
(sebastian.middeke@treqs.org)
ist Senior Consultant für Quality Engineering & Assurance bei der Cognizant Technology Solutions GmbH. Er berät seit 2005 Projekte zu Themen des Test- und Anforderungsmanagements und coacht die Einführung und Optimierung agiler Prozesse.

Effizientere agile Prozesse: Testfallbasierte Anforderungsdokumentation

Anforderungen in Prosaform bergen oft die Gefahr von Missverständnissen und Fehlentwicklungen, die auch agile Projekte in eine Schieflage bringen können. Hier hilft es, Anforderungen mithilfe von Testfällen zu dokumentieren. Gerade in großen oder verteilten Projekten, in denen eine effiziente Kommunikation häufig schwierig ist, erreichen wir mit diesem Ansatz immer wieder sehr gute Ergebnisse – die Resonanz der Beteiligten ist durchweg positiv. Dies hat uns in den letzten Jahren motiviert, unsere Erfahrungen im treqs-Ansatz zu konsolidieren. Dessen positiven Effekt wollen wir in diesem Artikel vorstellen und diskutieren.

Ein charakteristisches Merkmal agiler Prozesse ist die kontinuierliche und enge Zusammenarbeit von Kunde und Entwickler: Softwareentwicklung ist aus agiler Sicht ein gemeinsamer Lernprozess. Nicht nur die Software entwickeln wir weiter, auch unser Verständnis des Projekts vertiefen wir auf allen Ebenen und wir konkretisieren oder korrigieren unsere Zielvorstellung. Erkenntnisse und neue Ideen sind uns dabei willkommen und wir berücksichtigen sie im laufenden Prozess.

Dazu müssen sich Kunde und Entwickler in ihren unterschiedlichen Rollen Sprint für Sprint über das gemeinsame Ziel und die konkreten nächsten Aufgaben verständigen. Die zentralen Artefakte, über die sich Kunde und Entwickler dabei regelmäßig austauschen, sind vor allem die User-Storys, die das Soll beschreiben, sowie die laufende Software.

It's all about Feedback

Die kontinuierliche und systematische Abstimmung von Soll- und Ist-Stand ist ein weiteres charakteristisches Merkmal des agilen Vorgehens. Feedback ist in agilen Prozessen die entscheidende Kraft, die eine hohe Qualität und Effizienz in der Umsetzung ermöglicht und den Prozess in der Spur hält. Nur wenn die Qualität der Anforderungen und der Umsetzung stimmt und wir den Stand des Projekts exakt bestimmen können, sind wir in der Lage, im laufenden Prozess auf geänderte Anforderungen angemessen zu reagieren, ohne das Projektziel aus den Augen zu verlieren.

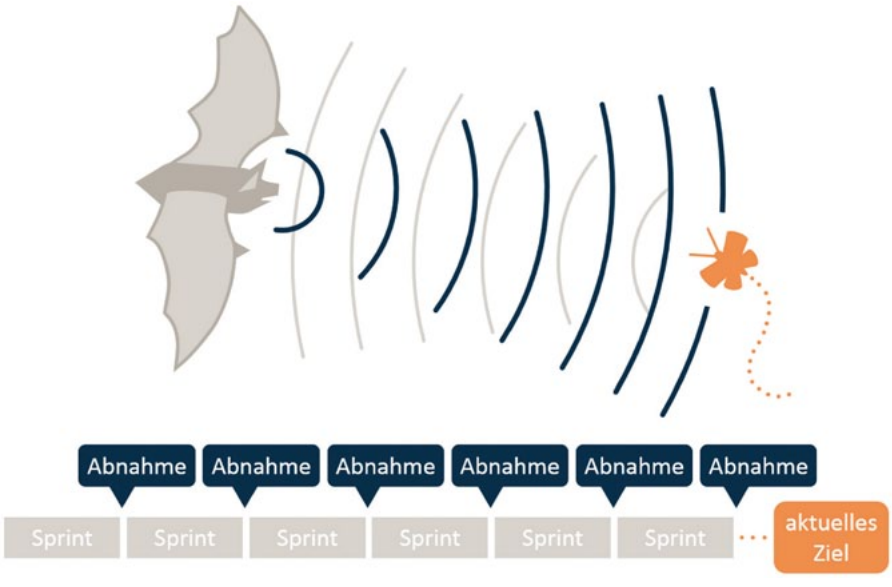


Abb. 1: It's all about Feedback: Sprintabnahmen geben regelmäßig Orientierung.

In der Praxis beobachten wir aber, dass auch in agilen Prozessen Qualität und Effizienz nicht garantiert sind. Falsch verstandene Anforderungen und langwierige Rückkopplungszyklen können agile Prozesse in eine Schieflage bringen. Eine effiziente Kommunikation ist essenziell für agile Projekte – aber nicht selbstverständlich. In vielen Projekten sind die Bedingungen nicht ideal und Kommunikationsprobleme sind wahrscheinlich: Große Teams, verteilte Teams, Fluktuation im Team (vgl. [Koc10]), Sprachbarrieren, Zeitzonen, Zeitdruck, Komplexität und viele weitere Faktoren, denen wir in der Praxis begegnen, machen Missverständnisse wahrscheinlich.

Sprintabnahmen geben regelmäßig Orientierung

In einem gut funktionierenden agilen Projekt stellen wir spätestens bei der Abnahme am Ende eines Sprints fest, ob Anforderungen richtig verstanden und korrekt umgesetzt wurden.

Diese regelmäßigen Abnahmen in kurzen Abständen machen agile Projekte deutlich risikoärmer im Vergleich zu klassischen Wasserfall-Projekten, in denen nur eine einzelne Gesamtabnahme am geplanten Projektende steht. Über diese Sprintabnahmen verschaffen wir uns Sprint für Sprint ein genaues Bild des Projektstandes (vgl. [Koc11]), auf dessen Basis wir unsere Planung weiter konkretisieren oder korrigieren können (siehe **Abbildung 1**). Diese Möglichkeit fehlt in klassischen Wasserfall-Projekten.

Natürlich wird auch in Wasserfall-Projekten die Qualität der Umsetzung im laufenden Projekt geprüft, z. B. über Unit-Tests oder manuelle Entwicklertests, die auch dort angewendet werden. Jedoch sind diese Tests immer unter Vorbehalt zu sehen, da sie nicht prüfen können, inwieweit die Anforderungen richtig verstanden worden sind – Anforderungen können eben nicht nur in der Umsetzung, sondern auch im Testdesign missverstanden werden.

Auch in agilen Projekten haben die Tests, die Entwickler im laufenden Sprint durchführen, nur dann eine verbindliche Aussage, wenn sie den Anforderungen exakt entsprechen.

Das letzte Wort hat der Kunde

Am Ende des Sprints sind wir in der Lage, Abweichungen vom Soll genau zu erkennen. Dabei können wir uns auf die fachliche Expertise des Kunden verlassen, der seine Abnahmetests durchführt. Sofern bei der Sprintabnahme Missverständnisse auftreten, können wir diese direkt klären. Bewährt haben sich kurze, intensive Abnahme-Workshops, in denen Entwickler und Kunden gemeinsam die Software testen und Probleme „auf dem kurzen Dienstweg“ klären.

Doch auch in einem agilen Projekt gilt: Wenn die Abnahme fehlschlägt, muss das Umgesetzte aufwändig nachgebessert werden. Das ist ineffizient und in der Summe letztendlich teuer, auch wenn die „Fallhöhe“ am Ende eines Sprints geringer ist als beim Scheitern der Gesamtabnahme am Ende eines Wasserfall-Projekts.

Nachfragen und Fehlinterpretationen wollen wir vermeiden. Wir benötigen daher eine Lösung, die es dem Kunden ermöglicht, seine Anforderungen angemessen detailliert, vollständig und unmissverständlich zu beschreiben. Weiterhin soll diese Lösung den Entwickler in die Lage versetzen, seine Ergebnisse frühzeitig verbindlich zu überprüfen und Fehlentwicklungen selbständig zu erkennen.

Der treqs-Ansatz

Am Ende des Sprints gelingt uns die Aufklärung von Missverständnissen oder Fehlern sehr gut. Wir haben uns daher gefragt, ob wir dies nicht auch schon vorher schaffen können, denn: *Wenn die Tests am Ende des Sprints helfen, Missverständnisse aufzuklären, dann können sie vorher helfen, Missverständnisse zu vermeiden.*

Zu den Aufgaben des agilen Teams gehört es, im Verlauf des Sprints Testfälle zu spezifizieren, zu dokumentieren und durchzuführen. Warum machen wir das dann nicht bereits vor dem Sprint, um bereits während des Sprints davon profitieren zu können?

Dabei soll der agile Charakter des Vorgehens nicht verloren gehen. Wir möchten vor Sprintbeginn nur jene Testfälle erstellen, die wir bei dessen Ende für die Sprintabnahme benötigen. Die Devise „No Big Upfront Design” bleibt weiterhin bestehen.

Die Geschichte

Die grundsätzliche Idee der testfallbasierten Anforderungen haben wir 2005 in einem verteilten agilen Projekt erarbeitet, in dem Verständigungsprobleme aufgrund von Sprachbarrieren immer wieder zu Missverständnissen führten. Das Entwicklerteam war als Folge davon erheblich mit Bug-Fixing beschäftigt und der Kunde mit langwierigen Sprintabnahmen sowie dem Erfassen und Nachtesten von Bugs. Beide Seiten wendeten zudem viel Zeit für das Klären von Anforderungen auf.

Wir bemerkten aber durch das intensive Bug-Fixing, dass die Klickpfade, die wir bei der Erfassung von Bugs angegeben haben, es den Entwicklern sehr leicht machten, die Anforderungen korrekt zu verstehen. Zudem machten sie das Nachtesten für den Kunden deutlich einfacher. Schließlich gingen wir dazu über, auch die User-Storys prophylaktisch um solche Klickpfade zu ergänzen, bis schließlich der größte Teil der Spezifikation aus solchen Pfaden bestand. Die Aufwände für Bug-Fixing und Abstimmungen sanken dadurch erheblich.

Aufgrund unserer Überlegungen und der immer wieder sehr positiven Erfahrungen haben wir diesen Ansatz (als agile Coachs oder produktive Projektmitglieder) auch in weiteren agilen Projekten verfolgt und ausgearbeitet.

Die Form

Eine klassische User-Story hat die Form: „Ich als Anwender in der Rolle … möchte … tun, um …“ Für ein Zeiterfassungs-System könnte eine User-Story also beispielsweise lauten: „Als Mitarbeiter möchte ich alle Zeiten, Aktivitäten und Kommentare eines Arbeitstages in den aktuellen Tag kopieren und dann anpassen können, um bei ähnlich strukturierten Arbeitstagen möglichst wenig eingeben zu müssen.“

Im *treqs*-Ansatz (*test-based requirements*, gesprochen „tracks“) definieren wir Anforderungen mithilfe von relevanten Akzeptanz-Testfällen aus Anwendersicht. Für die Zeiterfassungs-Story liegt ein zentraler Akzeptanztest nahe (siehe **Beispiel 1**: Der Mitarbeiter wählt einen Tag mit vorhandenen Einträgen aus).

- Der Mitarbeiter wählt einen Tag mit vorhandenen Einträgen aus.
- Das System zeigt eine Übersicht mit allen Zeiten, Aktivitäten und Kommentaren für diesen Tag.
- Der Mitarbeiter kopiert alle diese Einträge in den heutigen Tag.
- Das System zeigt eine Übersicht für den heutigen Tag. Die Übersicht enthält genau die Zeiten, Aktivitäten und Kommentare des zuvor gewählten Tages.
- Das System zeigt eine Meldung, wie viele Einträge von welchem Datum nach heute kopiert worden sind.

***Beispiel 1:** Ein erster Akzeptanztest für die Beispiel-Story.*

Weitere Akzeptanztestfälle könnten sein:

- Der Mitarbeiter wählt einen Tag ohne Einträge aus.
- Der Mitarbeiter wählt den heutigen Tag aus.
- Der Mitarbeiter wählt einen Tag mit fehlerhaften Einträgen aus.
- Der heutige Tag enthält bereits Einträge.

Der erste Akzeptanztestfall wird – wenn auch nicht in allen Details – bereits aus dem Prosatext der User-Story deutlich. Bei den weiteren Akzeptanztestfällen ist das nicht so. Es ist für den Kunden auch beim Schreiben eines Prosatextes unerlässlich, alle relevanten Anwendungsfälle abzudecken. Indem wir sie als Testfälle dokumentieren machen wir sie explizit.

Dem Kunden stehen zusätzliche Formen der Dokumentation weiterhin offen. Dies können beispielsweise Mockups sein oder Prosatexte, die Grundprinzipien und Überblickswissen vermitteln, Hintergründe erklären oder die fachliche Motivation beschreiben. Ebenso sind Diagramme, Tabellen, Skizzen oder eben auch die klassischen User-Story-Prosatexte möglich. Alles, was zur Klärung der Anforderungen beiträgt, ist weiterhin erlaubt. Unsere Devise beim treqs-Ansatz ist: ergänzen, nicht ersetzen!

treqs-Testfälle sind „Happy-Paths“

Im Rahmen eines Softwareentwicklungsprojekts trifft man normalerweise auf vier Arten von Testfällen:

1. Die *Positiv-Testfälle* bilden den gewünschten Normalablauf ab, üblicherweise, wie ihn sich der Kunde vorstellt. Diese „Happy-Paths“ können auch negative Ergebnisse beschreiben (z. B. Anzeige eines Warnhinweises), solange es das gewünschte Ergebnis aus Kundensicht ist. Der Kunde gibt seine zentralen Positiv-Testfälle vor, die von Testexperten um gegebenenfalls nötige Testfallvarianten ergänzt werden. Zumindest die zentralen Positiv-Testfälle sollten für eine Abnahme durchgeführt werden, da hierüber geprüft wird, ob die Software ihren Zweck erfüllt.
2. Die *Negativ-Testfälle* werden von den Testexperten im Projekt verfasst. Sie ergänzen die Positiv-Testfälle um alle Aspekte, die zum Aufdecken von Abweichungen nötig sind. Beispiele hierfür wären überschrittene Grenzwerte oder die Verwendung falscher Datentypen (z. B. Buchstaben anstelle von Zahlen).
3. Die Entwickler im Projekt spezifizieren die *technischen Testfälle*. Normalerweise sind dies Komponententests. Hier stehen die technischen Details der Umsetzung im Fokus, die dem Kunden oft unbewusst sind.
4. Die *nicht-funktionalen Testfälle* prüfen Aspekte außerhalb der funktionalen Eigenschaften, beispielsweise Performance oder Usability. Diese werden oft ebenfalls von den Testexperten verfasst.

Die letzten drei Testfall-Arten basieren auf der Anforderungsdokumentation des Kunden, werden aber nicht von ihm verfasst. Die Autoren dieser Testfälle können die Anforderungen natürlich missverstehen. Missverstandene Anforderungen sind im Testfall-Entwurf besonders tückisch, da Testfälle, die Anforderungen nicht korrekt wiedergeben, eine falsche Sicherheit vorspiegeln und damit in die Irre leiten. Werden die Anforderungen über treqs-Testfälle beschrieben, ist die korrekte Umsetzung der fachlichen Kernfunktionalität gut abgesichert und Irrtümer betreffen eher andere Aspekte (z. B. technische), die nicht im Kompetenzbereich des Kunden liegen.

Wie finde ich geeignete Testfälle?

Bei der Einführung von treqs in einem Projekt benötigen die Autoren der treqs-Testfälle anfangs Anleitung beim Schreiben. Das Schreiben von Anforderungen ist eine sehr anspruchsvolle Tätigkeit. Wenn es an das Formulieren geht, fehlt dem Kunden oft eine Vorstellung davon, welche Informationen der Entwickler im Einzelnen benötigt, und kann dies oft auch nicht nach eigenem Ermessen beurteilen.

Eine bewährte Daumenregel ist, sich beim Dokumentieren einer Anforderung die folgende Frage zu stellen: „Was würde ich am Ende des Sprints konkret ausprobieren, um mich davon zu überzeugen, dass diese Anforderung nach meinen Vorstellungen umgesetzt wurde?“

Diese Frage gibt dem Kunden eine sehr konkrete Orientierung bei der Formulierung der Anforderungen. Vor allem ist er in der Lage, diese Frage nach eigenem Ermessen zu beantworten. Gleichzeitig nennt er so die Details, die ihm wichtig sind und die der Entwickler für die korrekte Umsetzung benötigt.

Wenn der Kunde den Beispiel-Testfall zum Zeiterfassungssystem noch einmal anschaut und sich die obige Frage stellt, bemerkt er eventuell, dass Teile des Testfalls genauer gefasst werden müssen. Um die Benutzung möglichst einfach zu halten, möchte der Kunde Einträge z. B. nur in den heutigen Tag kopieren können und dazu nur auf einen Button drücken müssen. Also passt der Kunde den Testfall an (siehe Beispiel 2).

- Der Mitarbeiter wählt einen Tag mit vorhandenen Einträgen aus.
- Das System zeigt eine Übersicht mit allen Zeiten, Aktivitäten und Kommentaren für diesen Tag.
- Das System zeigt einen Button, um alle diese Einträge in den heutigen Tag zu kopieren, sofern der heutige Tag noch keine Einträge enthält, andernfalls ist der Button deaktiviert.
- Der Mitarbeiter drückt auf diesen Button.
- Das System zeigt eine Übersicht für den heutigen Tag. Die Übersicht enthält genau die Zeiten, Aktivitäten und Kommentare des zuvor gewählten Tages.
- Das System zeigt eine Meldung, wie viele Einträge von welchem Datum nach heute kopiert worden sind.

Beispiel 2: Eine genauer gefasste Version des Akzeptanztests für die Beispiel-Story.

So ist sichergestellt, dass Einträge nicht in einen beliebigen Tag kopiert werden können, dass nur ein einzelner Klick notwendig ist und dass keine problematischen Kollisionen von Einträgen auftreten können. Kompliziertere Lösungen, die in der Umsetzung, im Test und in der Wartung teurer wären, schließt der Kunde damit in diesem Punkt aus. Wäre die Spezifikation hier ungenauer, würden die Entwickler entweder selbst kreativ werden – und würden dabei auch fachlich möglicherweise irrelevante Sonderfälle berücksichtigen müssen – oder sie würden im laufenden Sprint versuchen, die Anforderungen mit dem Kunden zu konkretisieren.

Eigenschaften eines guten treqs-Testfalls

Ein Testfall beschreibt die Aktionen eines Benutzers und die darauf folgenden Systemreaktionen Schritt für Schritt. Ein guter treqs-Testfall (genau genommen jeder gute Testfall) erfüllt dabei folgende Eigenschaften:

- *Er ist relevant:* Die geprüfte Anforderung ist konkret und wichtig.
- *Er ist überprüfbar:* Benutzerinterakti-

on und Systemreaktion sind genau aufgeführt.
- *Er ist exemplarisch:* Die Schrittfolge ist eindeutig und verzichtet auf Verzweigungen.
- *Er ist durchführbar:* Er ist so detailliert wie nötig und so klar wie möglich.
- *Er ist wiederholbar:* Hierzu gehört, dass konkrete Testdaten angegeben werden.
- *Er ist einfach:* Er enthält weder Umwege noch unnötige Erklärungen.

Mit diesen Qualitätsmerkmalen im Hinterkopf stellt unser Beispielkunde schnell fest, dass der Testfall zum Zeiterfassungssystem noch weiter verbessert werden muss. Der Testfall ist relevant und auch einfach gehalten. Die Benutzeraktionen und Systemreaktionen sind aus funktionaler Sicht in den Teilen, die dem Kunden wichtig sind, genau beschrieben. Also ist der Test auch ausreichend überprüfbar.

Der Testfall ist jedoch noch nicht exemplarisch, da er im dritten Schritt eine Verzweigung enthält, für den Fall, dass für den heutigen Tag bereits Einträge existieren. Bei der Durchführbarkeit und Wiederholbarkeit fällt auf, dass die Frage offen bleibt, ob im ersten Schritt auch ungültige Einträge erlaubt sind. Der Kunde bessert seinen Testfall entsprechend nach (siehe Beispiel 3) und entscheidet sich für die Variante, in der der heutige Tag keine Einträge enthält. Die andere Variante beschreibt der Kunde dann in einem separaten Testfall.

Vorbedingung: Es existieren keine Einträge für den heutigen Tag. Es existiert mindestens ein Tag mit mindestens einem Eintrag. Alle Einträge dieses Tages sind gültig.

- Der Mitarbeiter wählt einen Tag mit ausschließlich gültigen Einträgen aus.
- Das System zeigt eine Übersicht mit allen Zeiten, Aktivitäten und Kommentaren diesen Tag.
- Das System zeigt einen Button, um alle diese Einträge in den heutigen Tag zu kopieren.
- Der Mitarbeiter drückt auf diesen Button.
- Das System zeigt eine Übersicht für den heutigen Tag. Die Übersicht enthält genau die Zeiten, Aktivitäten und Kommentare des zuvor gewählten Tages.
- Das System zeigt eine Meldung, wie viele Einträge von welchem Datum nach heute kopiert worden sind.

Beispiel 3: Die fertige Version des Akzeptanztests für die Beispiel-Story.

Im Beispiel sind keine exakten Testdaten angegeben. Da der Ablauf des Beispiel-Testfalls nicht von den konkreten Einträgen abhängt, ist das hier unkritisch. In anderen Fällen kann es notwendig sein, sehr genaue Daten anzugeben, z. B. in einem Testfall, der die korrekte Berechnung der Gesamtstunden eines Tages beschreibt.

Der Testfall ist durchführbar und wiederholbar, sofern die Vorbedingungen erfüllt sind. Diese Vorbedingungen herzustellen, kann bei der konkreten Durchführung schwierig sein. Direkt wiederholbar ist der Beispiel-Testfall z. B. nur, wenn eventuell vorhandene Einträge des heutigen Tages gelöscht werden können. Eine einfache Lösung kann sein, die Beispiel-User-Story erst nach der Umsetzung der User-Story zum Löschen von Einträgen einzuplanen.

Gegebenenfalls kann es aber sehr aufwändig sein, die Vorbedingungen eines Testfalls zu erfüllen, oder es ist über einfache Benutzeraktionen sogar unmöglich. So kann der Monatsabschluss in einem Zeiterfassungssystem z. B. nur über ein Zurücksetzen der Datenbank zurückgenommen werden.

Die Vorbedingungen für Tests herzustellen, ist nach unserer Erfahrung vor allem ein Problem in Projekten, in denen nur wenig manuell getestet wird und eine Infrastruktur dafür fehlt. In Projekten, in denen wir den treqs-Ansatz von Projektbeginn an verfolgt haben, konnten wir eine solche Infrastruktur nach Bedarf schrittweise aufbauen.

Bewertung des treqs-Ansatzes

Der treqs-Ansatz wirkt sich bei all seiner Einfachheit an vielen Stellen des Entwicklungsprozesses und für alle Projektbeteiligten sehr positiv aus.

Kommunikation

Es hat sich gezeigt, dass Testfälle von allen beteiligten Rollen gleichermaßen gut verstanden werden – nicht zuletzt, weil sie von der Form her vertraut sind.

Anhand der Testfälle entsteht frühzeitig ein gemeinsames Verständnis der Fachlichkeit des zu bauenden Systems und eine gemeinsame Sprache kann sich etablieren. Diskussionen über die Umsetzung sind bereits mit relevanten Ergebnissen möglich, noch bevor die Umsetzung einer User-Story begonnen hat. Auch spät im Prozess Beteiligte, wie z. B. Tester, können zu einem sehr frühen Zeitpunkt Feedback geben und sich in die Konzeption einbringen.

Zudem ist die Umsetzung einer User-Story anhand der Testfälle von jedem unmittelbar überprüfbar. Selbst Personen, die nicht an der Konstruktion beteiligt sind, können sich so kompetent zum Ergebnis äußern. Das Team kommuniziert auf Basis der Testfälle wirkungsvoller und reibungsloser und rückt näher zusammen. Die Effizienz des Ansatzes zeigt sich im laufenden Prozess.

Detaillierungsgrad der Anforderungsdokumentation

Dadurch, dass treqs-Testfälle am System durchführbar sein sollen, liegt der Fokus des Autors der User-Story beim Schreiben stark auf dem Umgang des Anwenders mit dem System und den konkreten funktionalen (und zum Teil auch nicht-funktionalen) Details. Diese werden entsprechend genau beschrieben und fallen in der Regel auch nicht unbeabsichtigt unter den Tisch. Dies gilt auch für selbstverständliche Basis-Features, die gegenüber Leistungs- und Begeisterungs-Features (vgl. [Kan84]) gerne übersehen werden.

Das Ergebnis sind User-Storys, deren Detaillierungsgrad den Informationsbedarf bei Entwurf, Umsetzung und Abnahme des Systems genau trifft. Dadurch sinkt in der Folge der Kommunikationsbedarf, der sonst anfiele, um Unklarheiten oder Missverständnisse bezüglich der Anforderungen im Nachhinein zu beseitigen.

Aus den treqs-Testfällen wird der konkrete Einsatzkontext von System-Features deutlich und ermöglicht es allen Beteiligten, den fachlichen Bedarf hinter den Features ausreichend zu verstehen und gegebenenfalls ungenaue Formulierungen korrekt zu interpretieren. All das reduziert Nachfragen und Fehlentwicklungen.

Qualitätssicherung

Der treqs-Ansatz entlastet Entwickler und Tester darin, Anforderungen richtig zu deuten oder über Rückfragen ermitteln zu müssen, um geeignete Akzeptanzkriterien abzuleiten. Die Autorenschaft für die zentralen Testfälle des Systems liegt im treqs-Ansatz grundsätzlich auf der Fachseite, die ihre eigenen Anforderungen und Ak-

zeptanzkriterien am besten kennt. Seitens der Entwickler und Tester ist keine Interpretation bezüglich der Akzeptanzkriterien nötig.

Gleichzeitig lassen sich die treqs-Testfälle aufgrund ihrer Testfallstruktur gut qualitätssichern, da formale Mängel daran meist für alle Beteiligten offensichtlich sind.

Der treqs-Ansatz garantiert die Testbarkeit der Anforderungen, da die wichtigsten Basis-Testfälle von der Fachseite schon so früh wie möglich vorgegeben werden. Sie können dann von Entwicklern und Testern direkt verwendet werden und vergleichsweise leicht um technische Tests, Testvariationen und Negativ-Tests ergänzt werden. Der nötige Aufwand für die Testfallerstellung sinkt hierdurch erheblich.

Auch die Dokumentation von Bugs im Rahmen von Tests wird durch den treqs-Ansatz deutlich einfacher. Bugs sind für alle Beteiligten offensichtlich – zu ihrer Rekonstruktion muss in der Regel nur auf den betroffenen treqs-Testfall verwiesen werden.

Bei der späteren Abnahme durch den Auftraggeber helfen die konkreten treqs-Testfälle, da über sie der Zustand des abgelieferten Systems direkt überprüft werden kann. Anhand von Prosatexten ist es nach unserer Erfahrung sehr schwierig zu entscheiden, inwieweit eine Umsetzung der Spezifikation entspricht. Im Extremfall kann dies eine subjektive Entscheidung sein. treqs-Testfälle ermöglichen eine objektive Bewertung des aktuellen Entwicklungsstands gegenüber den Anforderungen.

Der Fokus von treqs liegt auf den Anforderungen für den aktuellen Sprint. Dennoch können treqs-Testfälle weiter verwendet und nach der Abnahme in einen Bestand von Regressionstestfällen aufgenommen werden.

Häufige Kritikpunkte

Wie positiv sich treqs auf die Projektarbeit auswirkt, spürt das gesamte Team sehr schnell, sobald die Fachseite damit beginnt, Anforderungen Testfall-basiert aufzuschreiben.

Bevor es soweit ist und treqs im praktischen Einsatz für sich selbst sprechen kann, müssen wir im Vorfeld auf ein paar typische Vorbehalte genauer schauen.

Extra-Aufwand

Der Ansatz wirkt zunächst aufwändig und ist es – im Vergleich mit der oft gelebten Praxis – sicher auch auf den ersten Blick, insbesondere weil nach unserer Beobachtung von der Fachseite oftmals vollständig auf die Formulierung von expliziten Akzeptanzkriterien verzichtet wird.

Die Anforderungsdokumentation gestaltet sich subjektiv aufwändiger, da die Definition der Testfälle bereits beim Schreiben der User-Storys hinzukommt. Dies wird anfangs häufig kritisiert, zumal das Schreiben der Testfälle eine anspruchsvolle und entsprechend aufwändige Tätigkeit ist.

Dennoch ist dieses Vorgehen sinnvoll und erzeugt bei genauerem Hinsehen keine zusätzlichen Aufwände: Unabhängig vom treqs-Ansatz muss eine gute Anforderungsdokumentation alle relevanten Anwendungsfälle berücksichtigen. Eine gute Qualitätskontrolle erfordert die Definition von weiteren, viel genaueren und umfassenderen Tests, die zudem von der Fachseite validiert werden müssen. Spätestens zur Abnahme muss sich die Fachseite überlegen, durch welche Tests sie sich davon überzeugen will, dass die Umsetzung den Anforderungen entspricht. Die Definition genau dieser Tests ziehen wir im treqs-Ansatz nach vorne, also vor die Umsetzung.

Die schiere Menge der Testfälle

Ein weiterer Kritikpunkt ist, dass die schiere Menge der Testfälle ab einem Punkt unübersichtlich wird. Dies ist jedoch nur der Fall, wenn treqs-Testfälle als Regressionstestfälle weiterverwendet werden. Dasselbe Problem tritt aber auf, wenn wir diese Regressionstestfälle im Nachhinein erstellen. Ein geeignetes Testmanagement ist immer unerlässlich – unabhängig vom treqs-Ansatz.

Hoher Pflegeaufwand der Testfälle

Änderungs- und Ergänzungsbedarf an den Testfällen fällt jeweils nur für den nächsten Sprint an. Es geht im treqs-Ansatz nicht darum, den gesamten Bestand an Testfällen zu pflegen und aktuell zu halten. Eine vollständige Spezifikation ist im Rahmen eines agilen Prozesses bekanntermaßen nicht notwendig und auch nicht erstrebenswert.

Dennoch gilt: Testfälle sind im Gegensatz zu Prosatext aufgrund ihrer übersichtlichen Aufteilung in einzelne Schritte vergleichsweise änderungsfreundlich und damit nicht schlechter zu pflegen. Im Laufe des Projekts kann der Kunde vorhandene Testfälle ganz pragmatisch als Vorlage für ähnliche Anforderungen wiederverwenden. Falls er Erweiterungen oder Sonderfälle benötigt, so kann er diese später über zusätzliche Testfälle leicht nachpflegen.

Frühes Commitment der Fachseite unrealistisch

Ein Softwareprojekt kann nur erfolgreich sein, wenn die Fachseite verbindlich zu ihren Anforderungen steht. Bei agilen Projekten ist dies minimal für die Anforderungen des nächsten Sprints nötig. Im Falle des treqs-Ansatzes ist zudem die Bereitschaft der Fachseite notwendig, sich frühzeitig auf die Testfall-Spezifikation einzulassen.

Falls die Fachseite nicht verbindlich zu ihren Anforderungen steht oder stehen kann, merken wir es im treqs-Ansatz also sehr viel früher als in anderen Ansätzen. Damit können wir ein wichtiges Projektrisiko vielleicht nicht einfacher lösen, aber frühzeitig angehen.

Testfälle werden schnell komplex

Das Schreiben (und später das Durchführen) von Testfällen macht Mühe. Es gilt: Je einfacher das System, desto einfacher sind die Testfälle. Eine Anforderung, die eine aufwändige Lösung erfordert, benötigt meist auch aufwändige Testfälle. Das motiviert den Kunden stark, das Design einfach zu halten.

Komplizierte Lösungen, wie sie z. B. durch Altsysteme vorgegeben werden, erscheinen wenig attraktiv, wenn sie durch aufwändige Testfälle beschrieben werden müssen. Die bloße Nennung einzelner Features mit Verweis auf das Altsystem ist im treqs-Ansatz keine Option. Das ist ein wichtiger psychologischer Punkt, der sehr zur Effizienz des treqs-Ansatzes beiträgt. Einfachheit lohnt sich früh für alle Beteiligte.

Abgrenzung zu ähnlichen Ansätzen

Der Ansatz, User-Storys bereits vor der Umsetzung mit Abnahmekriterien zu versehen, ist grundsätzlich nicht neu. Die Form solcher Akzeptanzkriterien bewegt sich dabei von freien Prosaformulierungen bis hin zu formalen Testfall-Spezifikationen, die z. B. durch Tools wie FitNesse ausführbar sind.

Der treqs-Ansatz definiert einen Formalisierungsgrad, der dazwischen liegt: Er ist formal genug, um unmissverständlich zu sein, und frei genug, um in der Sprache der Anwender und bereits im Vorfeld ei-

ner konkreten Umsetzung formuliert werden zu können. Das ermöglicht es, treqs nicht nur für einzelne Aspekte des Prozesses (wie z. B. die Qualitätskontrolle) zu verwenden, sondern an allen Stellen des Prozesses sinnvoll einzusetzen.

Test-Driven-Development

Der treqs-Ansatz ähnelt dem *Test-Driven-Development* (*TDD*), da er ebenfalls mit dem Schreiben der Tests beginnt. Allerdings geht er in einem Punkt wesentlich darüber hinaus: Der treqs-Ansatz setzt bereits im Requirements Engineering an und nicht wie TDD erst im Rahmen der softwaretechnischen Umsetzung.

Zudem beschreiben treqs-Testfälle die fachlichen Anforderungen an das Gesamtsystem und nicht die Anforderungen an eine einzelne technische Komponente (Unit) des Systems.

treqs-Testfälle enthalten bereits viele hilfreiche Informationen, die TDD oder ganz allgemein das Schreiben von Unit-Tests erleichtern: So kann das Verhalten einzelner Komponenten im Rahmen der in treqs beschriebenen Testszenarios betrachtet werden, um daraus wesentliche Teile der konkreten Unit-Tests inklusive Testdaten abzuleiten.

Manuelle Entwicklertests werden durch treqs deutlich erleichtert und garantieren gleichzeitig ein verbindliches Feedback. Zudem können die Testfälle aus den treqs von den Entwicklern unmittelbar und selbständig durchgeführt werden.

Ein eng an TDD angelehntes Verfahren des treqs-Ansatzes ist es zum Beispiel, die treqs-Testfälle Schritt für Schritt zum Laufen zu bringen. Die technische Umsetzung ist erfolgreich, wenn der Entwickler alle treqs-Testfälle erfolgreich durchführen konnte. Das ist eine wichtige Qualitätssicherungsmaßnahme im laufenden Sprint, welche die fachliche Abnahme am Ende des Sprints erleichtert.

Use-Cases

Use-Cases gehören zu den üblichen Wegen, Anforderungen in einem Softwareprojekt zu definieren. treqs-Testfälle stellen wie Use-Cases eine Auswahl von repräsentativen Szenarios bezüglich der dokumentierten Anforderung dar. Beide sind in der Sprache der Anwender verfasst. Allerdings sind treqs-Testfälle konkreter als Use-Cases – es sind quasi „am System durchführbare Use-Cases“, in denen die erwartete Reaktion des Systems genau spezifiziert wird.

Sonstige Ansätze

Darüber hinaus gibt es noch zahlreiche weitere Ansätze, die sich ebenfalls mit der frühen Verwendung von Akzeptanzmerkmalen beschäftigen. Aufgrund der Namen liegt ein Vergleich mit *Acceptance Test Driven Development (ATDD), Behavior Driven Development (BDD)* und *Specification by Example* nahe.

Alle drei Ansätze weisen zwar auf die Eignung von Testfällen zur Spezifikation hin, legen den Fokus aber auf die Automatisierung der Testfälle. Im treqs-Ansatz schließen wir die Automatisierung von Akzeptanztests als zusätzliche Option nicht aus, denn wir finden Testautomatisierung gut und wichtig. Wir stellen sie aber aus zwei Gründen explizit nicht in den Fokus: Zum einen sind maschinenlesbare Testfälle nach unserer Erfahrung nicht für jeden menschlichen Autor oder Leser gleichermaßen geeignet, insbesondere nicht von der Fachseite, der die Verwendung von Schlüsselwörtern oft komplizierter erscheint als der Entwicklerseite.

Zum anderen – und das wiegt noch schwerer – sollen treqs-Testfälle in Bereichen vage bleiben dürfen, die der Kunde nicht exakt spezifizieren möchte – sei es, um der Kreativität und Expertise der Entwickler Raum zu geben oder um Altbekanntes nicht unnötig genau beschreiben zu müssen. Automatisierte Testfälle lassen diese wichtige „menschengerechte Ungenauigkeit“ nicht zu.

Wir möchten der Fachseite mit treqs ein Mittel an die Hand geben, mit dem es ihr leicht fällt, inhaltlich sehr gute Anforderungstexte zu schreiben. Zumeist tun sie es dann auch gerne.

Fazit

Die Dokumentation von Anforderungen und die Verständigung zwischen Kunde und Entwickler gehören zu den größten Herausforderungen in einem Softwareprojekt. Die meisten Anforderungen sind funktional und basieren auf Benutzerinteraktionen. Derartige Anforderungen lassen sich sehr gut als Testfälle beschreiben, selbst wenn die Anforderungen als Prosatext schwer zu fassen wären.

Der treqs-Ansatz ist ein einfacher und bewährter Weg, um Anforderungen als Testfälle zu dokumentieren und die Kommunikation und Kooperation im Team frühzeitig zu unterstützen. Er führt zu weniger Fehlentwicklungen und Rückfragen an die Fachseite, ermöglicht verbindliche Entwicklertests und damit eine erwartungskonforme Umsetzung. Dies spart Aufwände, gerade in großen, unübersichtlichen Projekten und in Teams, in denen die Kommunikation schwierig ist.

Wenn Sie neugierig geworden sind, dann ergänzen Sie doch eine User-Story einmal mit einem ersten „Happy-Path“. Fangen Sie klein an und schauen Sie auf den positiven Effekt. ■

Der Artikel wurde ebenfalls in OBJEKTspektrum 02/2014 veröffentlicht.

Literatur & Links

[Koc10] J. Koch, J. Sauer, A Task-Driven Approach on Agile Knowledge Transfer, in: Agility Across Time and Space, Springer-Verlag 2010, siehe: http://www.springerlink.com/content/q4q22061184266x1/

[Koc11] J. Koch, S. Middeke, Transparenz und Qualitätskontrolle in agilen Projekten: Agile Selbstheilungsmechanismen, in: Javamagazin, 7/2011, siehe: http://it-republik.de/jaxenter/java-magazin-ausgaben/Java-7-000453.html

[Kan84] N. Kano, Attractive Quality and Must-be Quality, in: Journal of the Japanese Society for Quality Control, H. 4, 1984