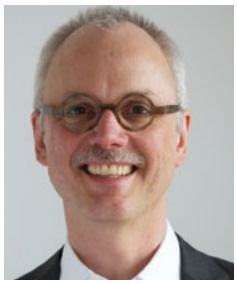

Matthias Bohlen
(mbohlen@mbohlen.de)
arbeitet als Experte für effektive Produktentwicklung. Teams engagieren ihn als Externen in vielen Projekten, damit er ihnen dabei hilft, sich zu organisieren und Prozesse für sich selbst zu finden und aufzustellen.

Anforderungen klein gehackt: Software mit mehr Feedback und weniger Risiko entwickeln

Kunden fordern häufig Features, die zu umfangreich und komplex sind, als dass ein Entwicklungsteam sie innerhalb kurzer Zeit realisieren könnte. Die Folge: Es dauert Wochen und Monate, bis das Team mit seinem Ergebnis zurückkommt. Der Kunde ist inzwischen gedanklich und geschäftlich schon weiter. Das Feedback kommt zu selten und fällt negativ aus, das Team hat am Ziel vorbei gearbeitet. Im Sinne der Business-Agilität ist es besser, kleine Features und Storys zu haben, die dem Kunden trotzdem Wert liefern oder ihn zumindest befähigen, den Teams Feedback zu geben. Kunden schaffen dies in der Regel jedoch nicht allein. Also muss die Fähigkeit, dem Kunden beim Kleinschneiden der Anforderungen zu helfen, zum Standard-Repertoire eines jeden Entwicklungsteams gehören. Für diese Aufgabe haben sich in den letzten Jahren sehr viele bewährte Techniken etabliert – doch jeder macht es ein bisschen anders. In diesem Artikel ist der „State of the Art" von Methoden zum Zerlegen großer Features in kleine, handliche Einheiten zusammengestellt. Sie erhalten so einen Überblick und sind in der Lage, die Methoden auszuwählen, die für Ihre Kunden und Ihre Teams am besten passen.

Lotteriespiel

Stellen Sie sich vor, ich spiele mit Ihnen Lotterie. Ich denke mir eine zweistellige Zahl, Sie setzen 2 Euro und versuchen zu erraten, welche Zahl ich mir gedacht habe. Wenn Sie richtig geraten haben, bekommen Sie 200 Euro. Ich biete Ihnen zwei verschiedene Versionen dieser Lotterie an:

- Bei der ersten Version setzen Sie 2 Euro und raten sofort beide Ziffern. Wenn es die Falschen waren, verlieren Sie 2 Euro, sonst bekommen Sie 200 Euro.
- Bei der zweiten Version setzen Sie zunächst nur 1 Euro und raten die erste Ziffer. Wenn es die Falsche war, verlieren Sie 1 Euro und das Spiel ist aus. Wenn es die Richtige war, dürfen

Spiele	Einsatz	Gewinn	Ergebnis
99	2 €	0 €	- 198 €
1	2 €	200 €	+ 198 €
		Summe	**0 €**

Tabelle 1: Lotterie ohne Feedback.

Spiele	Einsatz	Gewinn	Ergebnis
90	1 €	0 €	– 90 €
9	2 €	0 €	– 18 €
1	2 €	200 €	+ 198 €
		Summe	**+ 90 €**

Tabelle 2: Lotterie mit Feedback.

Sie noch 1 Euro setzen und die zweite Ziffer raten. Wenn Sie auch diese richtig erraten, bekommen Sie wiederum 200 Euro.

Welche der beiden Lotterien würden Sie lieber spielen? Risiko und Chance erscheint in beiden Fällen gleich: 2 Euro verlieren und 200 Euro gewinnen. Trotzdem gibt es einen Unterschied, oder?

Rechnen wir es sorgfältig durch (siehe Tabellen 1 und 2): Bei der ersten Lotterie kommen Sie mit 0 Euro heraus, wenn Sie sie 100 Mal spielen. In 100 Spielen der zweiten Lotterie machen Sie stattdessen 90 Euro Gewinn. Die zweite Lotterie -- die mit Feedback -- ist also deutlich attraktiver.

Softwareentwicklung

Warum erzähle ich Ihnen das? Das Feedback hat nicht nur in der Lotterie einen Wert, sondern auch in der Softwareentwicklung. Wenn ein Unternehmen Geld in die Entwicklung von Software steckt, ist es im Vorhinein nicht klar, ob die Entwickler die „Zahl im Kopf des Auftraggebers" richtig erraten werden, sprich: Bekommt der Auftraggeber auch die Software, die er haben möchte? Wenn die Entwickler einen großen Auftrag bekommen und längere Zeit kein Feedback, so ist das Risiko groß, dass am Ende etwas signifikant anderes herauskommt, als der Auftraggeber wollte. Es droht wegen der dann notwendigen Nacharbeit Zeit- und damit Geldverlust, z. B. durch verpasste Marktchancen.

Also ist die Überlegung der Agilen folgerichtig: *Wenn wir eng mit dem Kunden zusammenarbeiten und er uns häufig Feedback gibt, vergrößern wir damit deutlich die Chancen, ihm das liefern zu können, was er wirklich meint.*

User Stories

In agilen Methoden nutzen Teams oft Benutzergeschichten (*User Stories*), um mit dem Kunden zu kommunizieren. Eine User Story ist eine Idee für ein in sich abgeschlossene Aufgabe, die der Benutzer mithilfe des Systems erledigen kann. Die Story schreibt man als einen einzelnen Satz auf eine Karte und hängt sie zum Sortieren an die Wand. Anfänglich ist jede solche Story wie eine Eintrittskarte für ein Gespräch, das erst noch geführt werden muss.

Kunde und Team besprechen die Story (siehe Abbildung 1) und erarbeiten dabei pro Story:

- einen Anwendungsfall
- mehrere Akzeptanztests für diesen Fall
- Beispiele und Formate für Ein- und Ausgabedaten
- die geforderte Qualität

Die gewünschten Qualitätseigenschaften dieser Story sollte man ebenfalls in diesem Gespräch klären, zum Beispiel:

- Performance in Form von Antwortzeit nach der Eingabe, zu verarbeitende Datenmenge pro Zeiteinheit, Speicherverbrauch im Server
- Erlernbarkeit durch klare und systematische Gestaltung der Benutzungsoberflächen
- Revisionssicherheit durch Protokollierung während des Ablaufs.

Das Gespräch zur Anforderungsanalyse hat also für *eine* Story *mehrere* Ergebnisse, eine wichtige Erkenntnis, die wir gleich noch benötigen werden.

Kleingehacktes

Bei der Ideenfindung für eine Story, bei der Planung der Storys für ein Release oder beim Story-Analysegespräch zwischen Kunde und Team kann herauskommen: Diese Story ist viel zu groß -- das Team würde Wochen brauchen, bevor es dem Kunden etwas Lauffähiges zeigen und Feedback einholen kann. Damit stehen Kunde und Team vor einem Risiko, denn lange Bearbeitungszeiten bedeuten wenig Feedback. Wenig Feedback bedeutet, dass das Team möglicherweise Software bauen wird, die der Kunde nicht haben wollte.

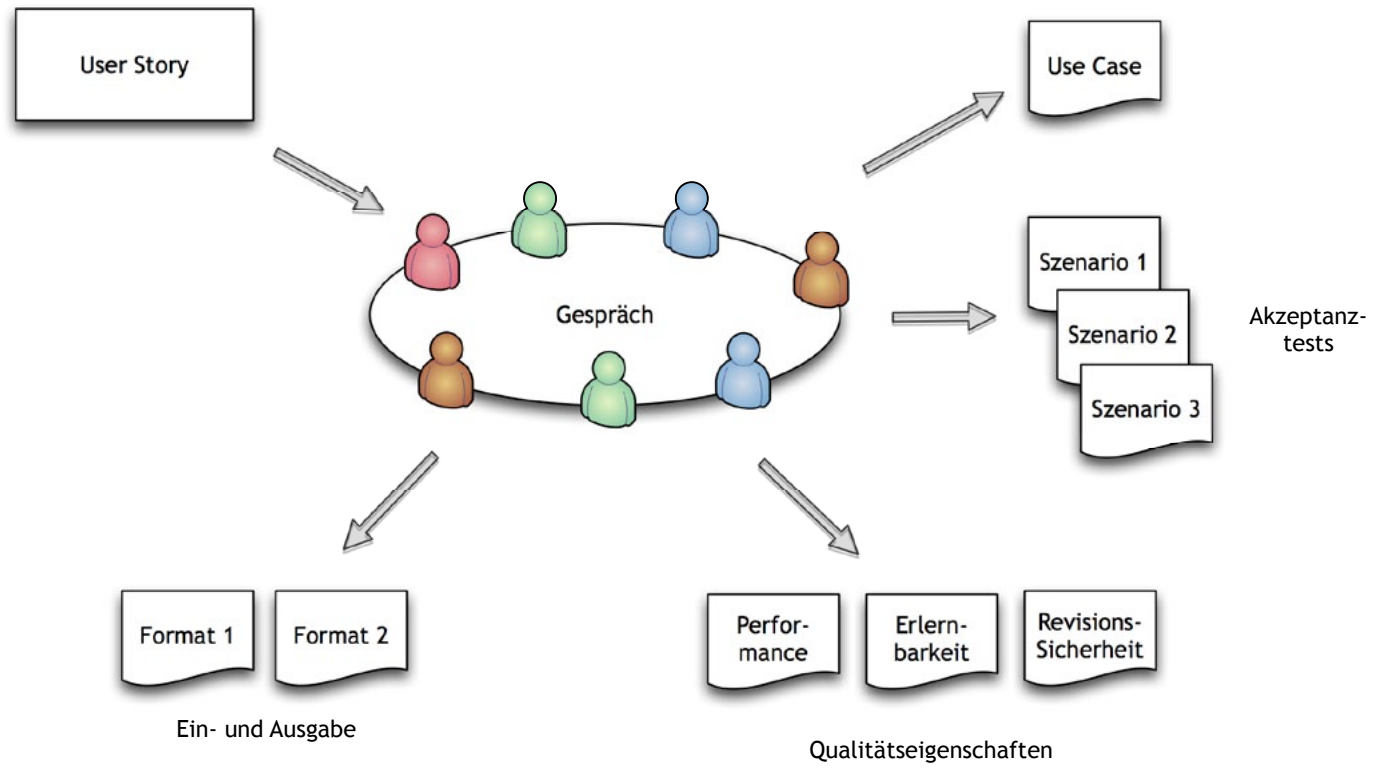

Abb. 1: Analysegespräch zu einer User Story.

Also wird das Team dem Kunden empfehlen, mit kleineren Storys zu arbeiten. Man braucht also ein Maß für die Größe einer Story.

Manche Teams arbeiten dazu mit Schätzungen in *Story-Points*, um herauszufinden, wie komplex eine Story sein wird. Typische Werte sind z. B. die Fibonacci-Zahlen mit 1, 2, 3, 5, 8, 13 … Story-Points. Teams sagen dann: „Wir wissen, dass wir Storys bis 5 Punkte gut schaffen. Wir wissen, dass es mit 8 schwierig wird und dass wir mit 13 definitiv ins Risiko laufen."

Andere Teams verzichten auf detaillierte Schätzungen und legen einfach fest: „Wir möchten, dass eine Story in einer Woche zu entwickeln sein kann." Solche Teams haben also nur zwei Story-Größen: „klein genug" und „zu groß". Letztere Methode hat bei reifen Teams deutliche Vorteile (vgl. [Boh12]).

Alles mit Methode

Würde das Team nun einfach den Kunden bitten, er möge seine Storys kleiner schneiden, sodass das Team sie schneller realisieren kann, wird der Kunde in vielen Fällen nicht wissen, wie er das machen soll. Agile Teams müssen also Methoden beherrschen, um dem Kunden beim Kleinschneiden (*Story Splitting*) zu helfen.

Dafür gibt es verschiedene Ansätze, die hier vorgestellt werden sollen. Sie können sich die heraussuchen, die für Ihren Kunden und/oder Ihre Teams passen.

Akzeptanztests

Betrachten Sie einmal, wie viele Akzeptanztests Ihre Story hat. Stellt sich heraus, dass man viele verschiedene Testfälle braucht, um zu definieren, was es bedeutet, wenn die Story „fertig" oder „erfolgreich umgesetzt" ist? Dann kann es sein, dass die Story einfach zu groß ist. In diesem Fall kann das Team dem Kunden Folgendes vorschlagen:

- Lass uns in einer ersten Version die Story so entwickeln, dass deine drei wichtigsten Akzeptanztests erfüllt werden.
- Gib uns dann Feedback.
- Wir entwickeln danach weiter und erfüllen die nächsten drei Tests usw.

Bei einer Story „Geld überweisen" würde man z. B. zuerst den Testfall „Genug Geld vorhanden" implementieren und danach erst den Fall „Konto überzogen".

Dadurch kann das Team unter Umständen recht schnell etwas bereitstellen und der Kunde merkt, ob die Menge der Akzeptanztests richtig ist, ob er welche streichen kann oder neue hinzufügen muss.

Qualitätseigenschaften

Was hat Ihr Gespräch über die Story ergeben? Muss das Team sehr viele Qualitätseigenschaften für diese Story gleichzeitig bereitstellen? Ein Beispiel:

- Die Eingabe von Kundendaten soll für den Benutzer in 10 Sekunden erfolgen können.
- Bei der Eingabe der Adresse soll das System den Ort sofort aus der Postleitzahl automatisch vervollständigen.
- Das System soll nach einer Sekunde die Eingabe akzeptiert haben.
- Der neue Kunde soll „sofort" zur Weiterverarbeitung „weltweit" zur Verfügung stehen.

Ziemlich hohe Anforderungen für eine Funktion zur Kundendatenerfassung, oder? Nicht etwa, dass sie nicht wichtig wären, doch: Werden sie wirklich alle sofort benötigt?

Das Team könnte in diesem Fall vorschlagen, zuerst eine Version zu entwickeln, die nur die erste Eigenschaft erfüllt. Dann gibt der Kunde Feedback dazu. Danach stellt das Team weitere Versionen bereit, die nach und nach auch die weiteren geforderten Eigenschaften aufweisen. Eventuell stellt sich unterwegs heraus, dass weniger oder mehr Qualität benötigt wird. Dann kann man das weitere Vorgehen anpassen.

Input, Verarbeitung, Output

Wenn eine Story im System ablauffähig ist, wird die zugehörige Funktion zuerst Daten einlesen, sie verarbeiten (mit Algorithmen oder Geschäftsregeln) und sie dann ausgeben. Alle drei Vorgänge – also Eingabe, Verarbeitung und Ausgabe – können als Ansatzpunkte dienen, um Storys klein zu schneiden. Liz Keogh hat das in einem Blog-Post sehr schön dargestellt (vgl. [Keo08]). Ich lasse mich von den Beispielen, die sie dort bringt, inspirieren und füge Erfahrungen aus eigenen Projekten hinzu.

Split nach Geschäftsregeln

Vielfach fordert der Kunde, dass die Story sehr viele verschiedene Arten von Objekten verarbeiten muss, für die jedes Mal andere Verarbeitungsmethoden oder Geschäftsregeln erforderlich sind.

In einem Payroll-System soll das Gehalt von Mitarbeitern berechnet werden, und zwar sowohl von Vollzeit- und Teilzeit-Angestellten als auch von Bonus-Empfängern. Eigentlich durchläuft die Story ganz verschiedene Berechnungsverfahren. Der Benutzer merkt nichts davon, für ihn sind alles einfach nur Mitarbeiter.

Das Team könnte jetzt den Kunden fragen: „Wie wäre es, wenn wir die Funktion erst einmal für Vollzeit-Angestellte bauen

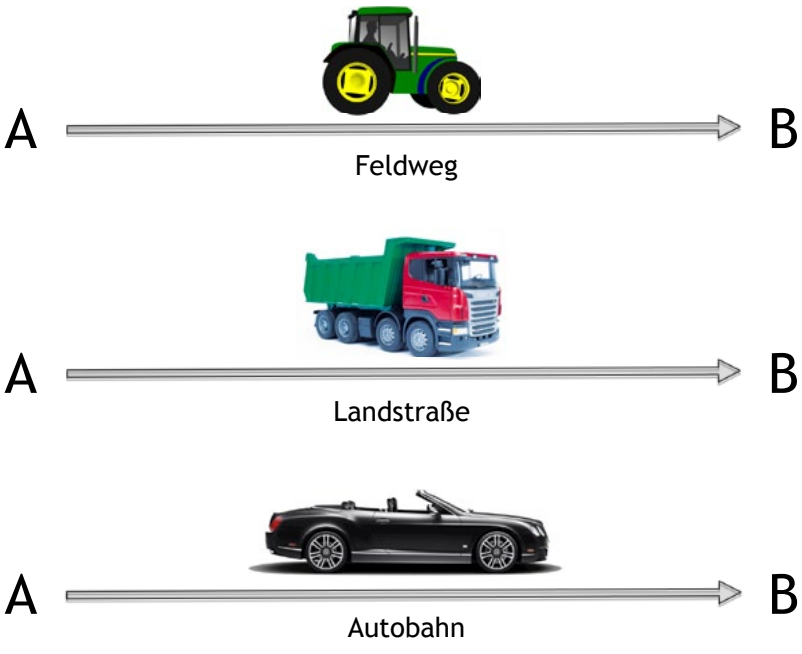

Abb. 2: Ausbaustufen der gewünschten Lösung.

und dafür von Ihnen Feedback bekommen? Danach bauen wir sie auch für Externe und für Partner, einverstanden?"

Die Storys, die das Team im Laufe der Zeit baut, könnten nach und nach funktionieren für:

- Vollzeit-Angestellte
- Teilzeit-Angestellte
- Erst kürzlich Eingestellte
- Angestellte, die weggehen
- Überstunden
- Altersversorgungsbeiträge
- Ausgaben
- Boni

Bei wissenschaftlichen Anwendungen, z. B. in der Bildverarbeitung, gibt es bei gleichbleibender Ein- und Ausgabe unterschiedlich genaue Verarbeitungsmethoden. Das Team kann vorschlagen, zunächst eine sehr einfache Methode mit geringer Genauigkeit zu implementieren und danach erst den Vollausbau zur Verfügung zu stellen.

Split nach Ausgabe

Innerhalb derselben Story kommt es vor, dass der Kunde mehr als eine Ausgabemöglichkeit fordert, zum Beispiel:

- auf den Bildschirm
- auf den Drucker
- in ein PDF
- als Datentransfer zu einem externen SAP-System

In diesem Fall kann das Team vorschlagen, immer nur eine Möglichkeit auf einmal zu entwickeln und dann nach Feedback zu fragen. So kann unnötige Arbeit eingespart werden, wenn sich herausstellt, dass die erste Version schon Missverständnisse enthält, die erst ausgeräumt werden müssen.

Split nach Eingabe

Es gibt Storys, deren Verhalten durch Optionen gesteuert verändert wird. Je nach Eingabe reagiert das System ganz unterschiedlich. Ein Beispiel hierfür ist die erweiterte Google-Suche (vgl. [Goo]). Teams könnten vorschlagen, die folgenden Such-Storys separat zu implementieren:

- Suche alle diese Wörter.
- Suche genau diese Wörter.
- Suche Keines dieser Wörter
- Suche Ergebnisse aus bestimmten Ländern.
- Suche in bestimmten Sprachen.
- Suche aus bestimmten Domänen.

Einfach	Schwieriger	Gedanken dazu
Forschen	Implementieren	Was haben andere getan?
Ausprobieren	Implementieren	Schnelle Lösung erforschen
Manuell	Automatisiert	Meist müssen wir die manuelle Lösung ohnehin behalten.
Kaufen	Bauen	Kann so oder umgekehrt sein: rechne Customizing-Kosten.
Bauen	Kaufen	... gegen die Kosten, es selbst zu tun.
Single-User	Multi-User	Weniger Sorgen über Skalierung, Benutzerkonten.
Nur API	Benutzungsschnittstelle	Tests könnten auch ohne Benutzerschnittstelle funktionieren.
Text- oder Skript-Oberfläche	GUI	Einfaches Interface kann bereits Ideen belegen.
Generisches UI	Spezielles UI	„Naked Objects"-Ansatz kann billiger sein.
Statisch	Dynamisch	Nur einmal machen und Updates ignorieren.
Fehler ignorieren	Fehler behandeln	Minimale Fehlerbehandlung
Transient	Persistent	Fokus auf Verhalten statt auf Persistenz.
Low-Fi	Hi-Fi	Qualität des Ergebnisses (z. B. Pixeltiefe).
Unzuverlässig	Zuverlässig	Perfekte Uptime ist sehr teuer.
Im kleinen Rahmen	Im großen Stil	Baue Lastkapazität erst langsam auf.
Weniger „-barkeiten"	mehr „-barkeiten"	Gehe Nicht-funktionales erst später an.
Wenige Features	Viele Features	Weniger Features sind leichter zu machen.
Haupt-Durchlauf	Alternative Durchläufe	Den grünen Pfad versus alle möglichen Pfade.
0	1	Nichts ist einfacher als etwas
1	Viele	Eins ist einfacher als eine Menge.
Eine Ebene	Viele Ebenen	Eine Ebene ist die Basis für alle Ebenen.
Basis-Fall	Allgemeiner Fall	Basis-Fall muss gemacht werden, die anderen nicht.

Tabelle 3: 20 Wege, eine Story zu zerteilen.

Feldweg, Landstraße, Autobahn

Allgemein kann das Team einfach den Kunden sehr bildhaft fragen: „In welcher Qualität und zu welchem Preis möchten Sie die Lösung geliefert bekommen?“ Möchten Sie …

- … einen Feldweg, sodass Sie mit einem Traktor die wichtigen Dinge von A nach B bringen können?
- … eine Landstraße, auf der Sie mit Lastwagen große Mengen transportieren können?
- … eine Autobahn, auf der das Ganze dazu noch schnell geht?

Logischerweise steigt in diesen drei Stufen (siehe Abbildung 2) der Preis an, sodass der Kunde sich aussuchen kann, wie viel er ausgeben möchte. Das Team muss dabei klar herausarbeiten, was denn nun Feldweg, Landstraße und Autobahn technisch und fachlich bedeuten.

20 sonstige Methoden

Bill Wake hat in einem Artikel (vgl. [Wak09]) zu diesem Thema 20 Ideen zusammengetragen, wie man User Stories zerkleinern kann. Der Grundansatz ist dabei immer:

1. Story zerteilen in einen einfachen und einen schwierig zu realisierenden Teil.
2. Den infachen Teil realisieren.
3. Feedback dazu einholen und besprechen.
4. Den schwierigen Teil möglicherweise inzwischen weglassen können.

In Tabelle 3 sehen Sie die Ideen, die Bill Wake dazu gesammelt hat.

Desaster-Rezepte

Die Methoden, die ich soeben beschrieben habe, klingen attraktiv, besonders für das Entwicklungsteam. Seien Sie sich jedoch im Klaren, dass sie für den Kunden unter Umständen Mehraufwand bedeuten und er deshalb nur dann begeistert sein wird, wenn er sieht, die Methoden ihm ermöglichen, das lauffähige System häufiger zu inspizieren und mit seinem Feedback das Vorhaben in Richtung Erfolg zu steuern.

Wenn Sie also mit diesem Konzept scheitern wollen, dann zerkleinern Sie die Storys immer so, dass der Kunde *nicht* sieht, was er davon hat. Splitten Sie zum Beispiel nach Architekturebenen oder technischen Eigenschaften:

- Wir haben schon mal die Datenbank richtig angelegt.
- Die Eingabevalidierung funktioniert bereits, doch die Funktion selbst tut noch nichts.
- Nur bestimmte Benutzergruppen können auf die Funktion, die noch nichts tut, zugreifen.

Ein weiterer Garant für zuverlässiges Scheitern ist es, wenn Sie beim Kunden den Eindruck erwecken, Sie wollten sich um seine Anforderungen drücken. Geben Sie dem Kunden am besten das Gefühl, dass der schwierige Teil, den Sie beim Zerkleinern herausschneiden, später gar nicht mehr gemacht werden soll. Besonders gut wirkt es, wenn Sie z. B. vorschlagen, die Fehlerbehandlung oder die ausführliche Validierung der Eingabedaten wegzulassen.

Zusammenfassung

Das Kleinschneiden von Anforderungen ist eine wichtige Fähigkeit für Entwicklungsteams. Sie können als Team dem Kunden dabei helfen, kleine übersichtliche Storys zu fordern, die Sie schnell bereitstellen können. Der Kunde gibt Ihnen Feedback und kann das Vorhaben dadurch leichter steuern. Dadurch reduzieren Sie das Risiko ganz entscheidend und vergrößern die Erfolgschancen. Es gibt dafür bewährte Methoden, die Sie einsetzen können:

- Bei zu vielen Akzeptanztests pro Story entwickeln Sie die Story schrittweise, sodass immer mehr Akzeptanztests erfüllt werden.
- Bei zu vielen Qualitätseigenschaften pro Story bringen Sie schrittweise mehr Qualität in die Story.
- Bei zu vielen verschiedenen Inputs, Verarbeitungsvorschriften oder Outputs entwickeln Sie so, dass die Story zunächst für wenige, dann für immer mehr Kombinationen von Input, Verarbeitung und Output funktioniert.

Wenn Sie einen sehr kostenbewussten Kunden haben, wählen Sie die Methode „Feldweg, Landstraße, Autobahn“ und überlassen ihm die Entscheidung, wann er abbricht. Aus dem bunten Strauß der 20 sonstigen Ideen wählen Sie aus, falls keine der obigen, eher systematischen Verfahren so richtig passt.

Erzählen Sie mir davon, wenn Sie diese Methoden in eigenen Projekten angewendet und Storys damit zerteilt haben. Viel Erfolg! ■

Der Beitrag wurde ebenfalls in der Printausgabe von OBJEKTspektrum 04/2015 veröffentlicht.

Literatur und Links

[Boh12] M. Bohlen, Liefern, schon vor dem Schätzen!, E-Book, siehe: www.amazon.de

[Goo] Googles erweiterte Suche, siehe: http://www.google.de/advanced_search

[Keo08] L. Keogh, Splitting up stories, 2008, siehe: http://lizkeogh.com/2008/09/11/splitting-up-stories/

[Wak09] William C. Wake: Twenty ways to split stories, siehe: http://xp123.com/articles/twenty-ways-to-split-stories/