www.bt-magazin.de 1.2012 Heft 8 Deutschland: € 14,90 Österreich: € 16,80 Schweiz: sFr 29,80 Luxemburg: € 16,90

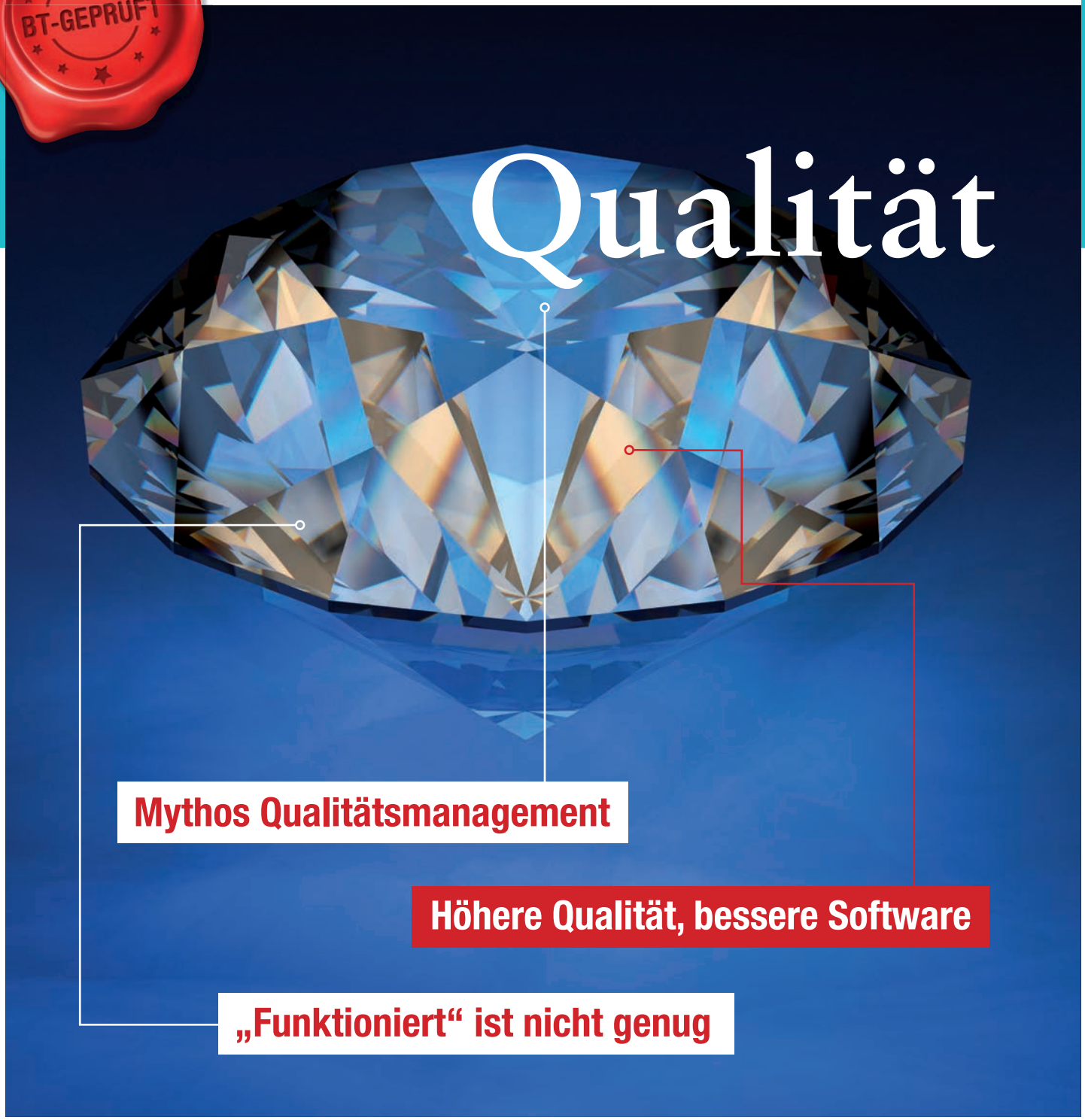
Business Technology

Business Technology

Architektur & Management Magazin

BT-GEPRÜFT

Expertenwissen für IT-Architekten, Projektleiter und Berater



Qualität

Mythos Qualitätsmanagement

Höhere Qualität, bessere Software

„Funktioniert“ ist nicht genug

4 191858 414906 01

Qualität intern und extern gewährleisten

Auf der Suche nach dem Qualitäter

Klar, jeder möchte Qualität in seiner Software. Und zwar immer und möglichst viel. Aber was genau ist eigentlich Qualität? Wie stellt man sie her? Und wer entscheidet, ob eine Software genug Qualität hat oder nicht? Dieser Artikel stellt der klassischen Vorstellung von Qualität eine alternative Sichtweise entgegen, die den Ideen der agilen Softwareentwicklung und des Lean Thinking entstammt.

AUTOREN: STEFAN ROOCK UND ARNE ROOCK

Go slow to go fast! So lautet ein Motto aus dem Lean Management. Hinter diesem scheinbaren Paradoxon steckt eine fundamental wichtige Erkenntnis: Wenn wir unseren Fokus nur auf Geschwindigkeit, also effizientes Arbeiten richten (Effizienz = „die Dinge richtig tun"), werden wir uns damit früher oder später (meistens früher, als wir denken) Probleme einhandeln, die unsere Geschwindigkeit massiv drosseln. Dieses Problem spiegelt sich auch in der klassischen Definition von Qualität wider, wie wir sie zum Beispiel in der Norm EN ISO 9000:2008 (der gültigen Norm zum Qualitätsmanagement) finden. Der zufolge ist Qualität der „Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt". Wenn wir also Software in hoher Qualität bauen wollen, müssen wir laut der ISO-Norm nur dafür sorgen, dass wir die Anforderungen an das neue System richtig verstanden haben, um es dann entsprechend dieser Anforderungen zu entwickeln und zwar möglichst effizient. Dass dieser Ansatz problematisch ist, zeigen Untersuchungen über die Benutzung von Softwaresystemen. Demnach werden 60 bis 70 Prozent der Funktionen selten oder gar nicht verwendet. Ob diese Funktionen gemäß den Anforderungen korrekt implementiert wurden, ist völlig unerheblich. Dass sie überhaupt implementiert wurden, ist das eigentliche Problem. Versuchen wir es also mit einer anderen Sichtweise: Jerry Weinberg schreibt „Quality is value to some person." [1] Wie hoch die Qualität unserer Software ist, hat also zunächst einmal gar nichts mit den Anforderungen zu tun. Entscheidend ist viel mehr, ob bestimmte Personen, nennen wir sie Kunden, die Software wertvoll finden. Im besten Fall gibt es natürlich eine Übereinstimmung zwischen beiden Ansätzen: Der Wert für den Kunden wird geliefert, indem seine definierten Anforderungen in Software überführt werden. Das Problem dabei ist, dass der Kunde in der klassischen Sichtweise nur einmal kurz am Anfang und am Ende des Entwicklungsprozesses beteiligt ist. Und das führt zu einer Reihe von Schwierigkeiten:

- Wir können am Anfang nie sicher sein, ob wir die Bedürfnisse der Kunden richtig verstanden haben. Weil

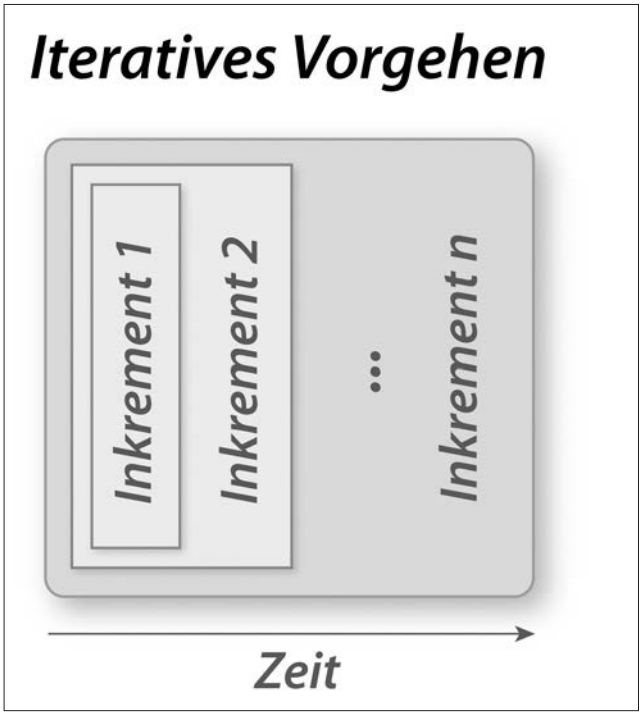


Abb. 1: In Scrum und Kanban wird potenziell immer wieder über dieselben Codestellen iteriert, weil ständig neue Inkremente erstellt werden, die das Gesamtsystem beeinflussen

Iteratives Vorgehen

Mit dem Wort „iterativ" sind an dieser Stelle nicht Iterationen fester Länge gemeint, wie sie in Scrum unter dem Namen Sprints obligatorisch sind (in Kanban jedoch eher selten verwendet werden). Stattdessen bedeutet „iterativ" hier zunächst nur, dass derselbe Anteil des Systems potenziell mehrfach überarbeitet wird, dass das Team also über den Code iteriert (**Abb. 1**). Das Einbeziehen des Kunden führt zwangsläufig zu einem iterativen Vorgehen. Existierende Funktionalität wird angepasst, geplante Funktionalität wird geändert, entfernt oder neu hinzugefügt. Das bedeutet, dass existierender Code immer wieder geändert werden muss, wenn wir eine hohe externe Qualität erreichen möchten.

Die Entwickler eines Systems sind auch immer ihre eigenen Kunden.

wir den Kunden nach der Anforderungsdefinition aber ziemlich lange nicht zu Gesicht bekommen, können wir unsere Annahmen nicht wirklich validieren. Stattdessen setzen wir sie in Software um und hoffen, dass wir richtig lagen, was aber leider nicht immer zutrifft.

- Was für unsere Kunden Wert schafft und was nicht, ist keine feste Größe, sondern ändert sich mit der Zeit. Nur weil ein Kunde zu einem bestimmten Zeitpunkt einen bestimmten Satz an Anforderungen für besonders wertvoll hielt, heißt das noch lange nicht, dass dies zu einem späteren Zeitpunkt noch so ist. Der Markt ändert sich, die Konkurrenz schläft nicht, neue technische Chancen ergeben sich und neue Ideen entstehen. Unter diesen Gesichtspunkten wäre es am Besten, wir könnten unseren Kunden möglichst oft die Chance geben, Einfluss auf die Weiterentwicklung des Systems zu nehmen.
- Mit den ersten beiden Punkten hängt die Frage nach der Wichtigkeit der einzelnen Anforderungen eng zusammen. Weil ich nicht genau weiß, welche Teile der Software den größten Wert für den Kunden schaffen, versuche ich natürlich, alles mit der höchsten Priorität umzusetzen. Wenn dann der Zeitplan durcheinander gerät (also eigentlich immer), kann ich nicht vernünftig entscheiden, welche Anforderungen die wichtigsten sind.

EXTERNE QUALITÄT MACHT INTERNE QUALITÄT ZUR HERAUSFORDERUNG

Theoretisch wissen wir also, wie wir eine hohe Qualität unserer Software erreichen können: Wir müssen den Kunden so in unseren Entwicklungsprozess einbeziehen, dass wir regelmäßig Feedback darüber bekommen, ob unsere Annahmen zutreffen oder nicht. Und wir müssen dem Kunden die Möglichkeit geben, sich immer wieder umentscheiden zu können, wie genau das neue System aussehen soll. Dies alles betrifft die externe Qualität unserer Software, also die Qualitätsaspekte, die für einen Außenstehenden (z. B. Anwender) sichtbar und beurteilbar sind.

Diese höhere Änderungsfrequenz des Codes stellt eine große Herausforderung für die interne Qualität dar, also die Qualität der Softwarearchitektur, des Entwurfs und des Codes. Für die Frage, was interne Qualität ist, hilft uns wieder Weinbergs Zitat. Interne Qualität ist nicht etwa dann vorhanden, wenn bestimmte Metriken, wie Testabdeckung oder zyklomatische Komplexität, gute Werte liefern, sondern sie liegt dann vor, wenn bestimmte Menschen sagen, dass Architektur, Entwurf und Code eine hohe Qualität haben. Um herauszufinden wer diese Menschen wohl sein können, bemühen wir Ishikawa Kaoru, den Vater der japanischen Qualitätskontrolle. Von ihm stammt das Zitat „The next process is your customer“. Dieser Ausspruch geht auf eine Situation in einem japanischen Stahlwerk zurück, in dem Mitarbeiter ihre Kollegen in einem nachgelagerten Prozessschritt als „ihre Feinde“ bezeichnet hatten. Ishikawa Kaoru wollte mit seiner Aussage eine Sichtweise etablieren, in der die Kollegen nachgelagerter Prozesse als Kunden der eigenen Arbeit verstanden werden. Es geht dann darum, auch diese Kunden durch die eigenen Arbeitsergebnisse zu begeistern. Spinnen wir diesen Gedanken zu Ende, dann ist klar, wer über die interne Qualität unserer Software entscheidet: nämlich diejenigen, die sie als Nächstes nutzen. Folgt der Entwicklung eine nachgelagerte Qualitätssicherung, so sind auch die Mitarbeiter der Qualitätssicherung die Kunden der Entwickler. Sind sie mit den Arbeitsergebnissen der Entwickler unzufrieden, haben die Entwickler keine hohe Qualität geliefert – ganz egal, was zu einem früheren Zeitpunkt mal in einem Dokument festgehalten wurde. Und auf jeden Fall sind die Entwickler eines Systems auch immer ihre eigenen Kunden, weil sie den existierenden Code später (wahrscheinlich) wieder ändern müssen.

HERAUSFORDERUNGEN BEI SCRUM UND KANBAN

Iterative Ansätze wie Scrum und Kanban (Kasten: „Iteratives Vorgehen“) tun sich viel leichter mit der externen Qualität als klassische Vorgehensmodelle, und zwar aus verschiedenen Gründen:

1. Sowohl Scrum als auch Kanban begrenzen den Work in Progress, kurz WIP (also die Menge an paralleler Arbeit). In Scrum geschieht diese Limitierung über die Timeboxes: Das Team nimmt nur so viele Auf-

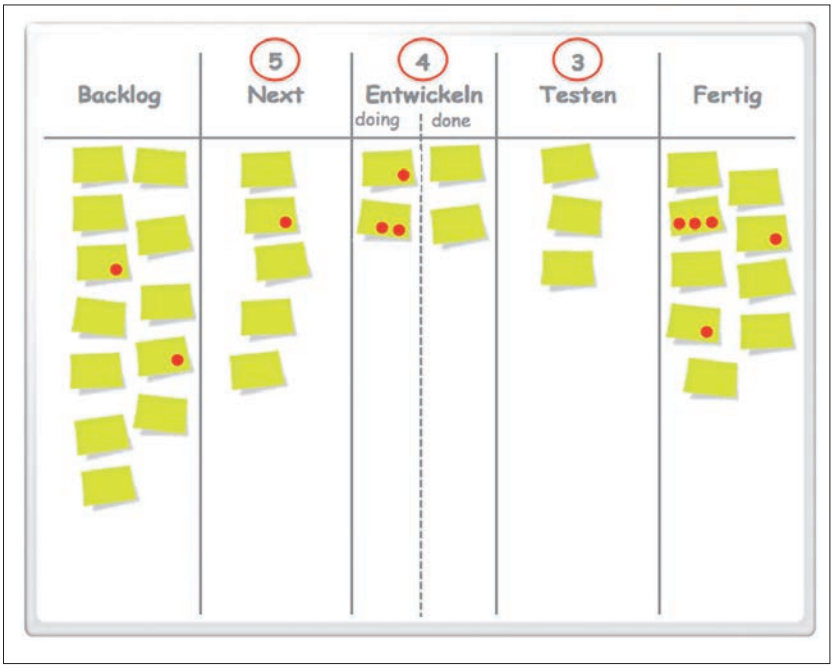


Abb. 2: Kanban-Board, auf dem Rückläufer durch rote Punkte visualisiert wurden

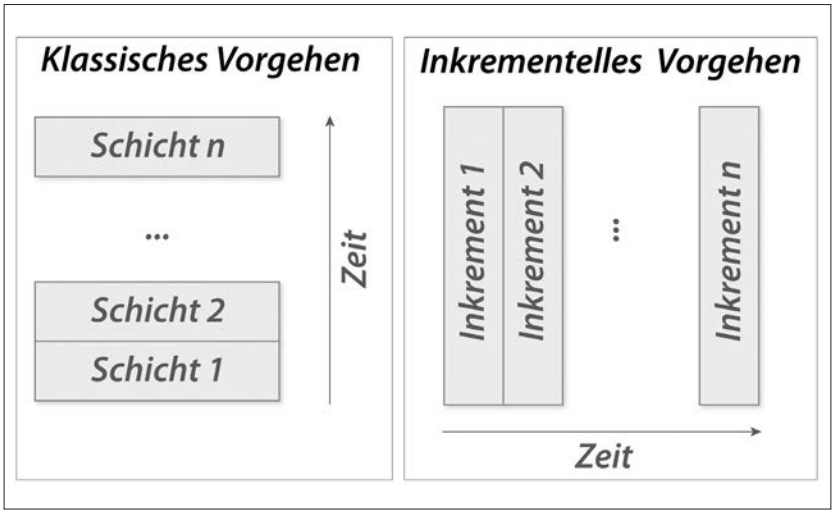


Abb. 3: Anders als beim klassischen Vorgehen werden beim inkrementellen Vorgehen immer komplette Durchstiche durch alle Schichten des Systems implementiert

gaben für den nächsten Sprint an, wie es meint auch wirklich fertig stellen zu können. In Kanban werden WIP-Limits meistens für jeden einzelnen Prozessschritt (z. B. Entwicklung, Test, Deployment usw.) definiert und in die jeweilige Spalte des Kanban-Boards geschrieben. Darüber hinaus arbeiten viele Teams mit persönlichen Avataren, die ebenfalls begrenzt sind. Wenn beispielsweise jedes Teammitglied nur drei Avatare zur Verfügung hat, kann es maximal an drei Aufgaben gleichzeitig arbeiten. Auch wenn der Zusammenhang auf den ersten Blick nicht unbedingt offensichtlich ist: Es zeigt sich immer wieder aufs Neue, dass zu viel WIP zu schlechterer Qualität führt. Ein Grund dafür sind häufige Kontextwechsel durch Multitasking. Der Entwicklungsbiologe John Medina schreibt, dass Menschen 50 Prozent länger für die Beendigung einer Aufgabe benötigen, wenn sie dabei unterbrochen werden. Und sie machen 50 Prozent mehr Fehler [2]. Niedriger WIP hingegen führt zu mehr Fokus und konzentrierterem Arbeiten. Darüber hinaus bedeutet hoher WIP längere Bearbeitungs- und Wartezeiten der einzelnen Aufgaben. Dadurch wird das Feedback der Tester, Kunden, Anwender usw. verzögert (vgl. Punkt 3).

2. Scrum und Kanban legen großen Wert auf Visualisierung. Wie es um die Qualität unseres Systems bestellt ist, sollte möglichst schnell sichtbar werden. Dafür werden Blockaden und Hindernisse sichtbar gemacht (z. B. mit knallroten Haftnotizen, die auf die jeweiligen Tickets geklebt werden, **Abb. 2**). In Kanban tut man darüber hinaus gut daran, Rückläufer (also Tickets, die sich entgegen der Konvention von rechts nach links auf dem Board bewegt haben) kenntlich zu machen (durch rote Punkte zum Beispiel). Und auch Nacharbeit (Rework), die im Lean Thinking als eine der größten Formen von Verschwendung angesehen wird, sollte zu jedem Zeitpunkt schmerzhaft sichtbar sein. Kanban kennt hierfür das Konzept der Bedarfsanalyse (Demand Analysis) [3]: Die zu erledigende Arbeit wird in unterschiedliche Aufgabentypen untergliedert, z. B. Features, Bugs usw. Durch verschiedene Ticketfarben lässt sich dann auch gut sichtbar machen, wie groß der Anteil von wertschaffender Arbeit (Value Demand) im Verhältnis zum Failure Demand ist, also Aufgaben, die eigentlich vermeidbar gewesen wären. Oft haben Teams durch diese einfache Visualisierung schon ein Aha-Erlebnis nach dem Motto „So wenig Kapazität wenden wir also tatsächlich für neue Features auf?“

3. Scrum und Kanban integrieren die Anwender und Kunden kontinuierlich in kurzen Abständen in die

Entwicklung (in Kanban ist diese Abstimmung zwar streng genommen nicht vorgeschrieben, aber für eine nachhaltige Verkürzung der Durchlaufzeit fast immer geboten). Die entwickelten Produktinkremente werden den Anwendern und Kunden immer wieder demonstriert und zur Verfügung gestellt. Ihr Feedback wird direkt in die weitere Entwicklung integriert. Durch diese kurzen Feedbackzyklen können wir sicherstellen, dass wir tatsächlich das entwickelt haben, was unsere Kunden haben wollen; ob wir also Qualität im Sinne Weinbergs geliefert haben.

Das alles mag sich trivial anhören, aber insbesondere der dritte Punkt hat es in sich. Denn zum einen ist entscheidend, dass nicht irgendwelche Arbeitspakete geschnürt werden, sondern dass man die Anforderungen in kleine, fachliche Einheiten unterteilt (z. B. User Storys oder Minimum Marketable Features). Denn nur diese garantieren uns, dass wir uns tatsächlich echtes Feedback von Kunden und Anwendern einholen können. Um solche sinnvollen fachlichen Einheiten zu definieren und dann mit vertretbarem Aufwand zu implementieren, ist schon etwas Übung nötig.

Zum anderen sind es die meisten Teams nicht gewohnt, so eng zusammenzuarbeiten und so häufig externes Feedback zu bekommen (das ja nicht immer so ausfällt, wie man es sich wünschen würde). Darüber hinaus entstehen neue Herausforderungen bezüglich der internen Qualität. Durch die häufigen unvorhergesehenen Änderungen muss existierender Quellcode immer wieder verändert werden. Und dabei muss der Code ständig sauber strukturiert bleiben, um die Entwicklungskosten dauerhaft gering zu halten. Architektur und Entwurf des Systems müssen sich inkrementell über die Zeit herausbilden. Man spricht von „inkrementellem Entwurf“ [4], [5]. Inkrementeller Entwurf ist dann erfolgreich, wenn eine so genannte flache Aufwandskurve entsteht (**Abb. 4**): Wenn wir immer wieder kleine Änderungen an unserem System vornehmen möchten, müssen wir Architektur und Design so ausrichten, dass eine flache Aufwandskurve entsteht – also die Kosten für neue Features mit der Zeit nur minimal ansteigen. Die Kosten je Feature wachsen über die Zeit nur logarithmisch bis linear. Dadurch ist sichergestellt, dass das System über einen langen Zeitraum kontinuierlich weiterentwickelt werden kann.

Wenn Architektur und Entwurf des Systems nicht auf inkrementelle Änderungen ausgelegt sind, steigen die Entwicklungskosten für neue Features mit der Zeit exponentiell an (**Abb. 5**). Das macht sich in Scrum und Kanban an sinkender Geschwindigkeit (Velocity) bzw. steigenden Durchlaufzeiten (Lead Times) bemerkbar. Leider wird das Problem in diesen Metriken erst dann sichtbar, wenn sich schon relativ viele Qualitätsprobleme angesammelt haben. Diese zu beseitigen, verursacht spürbaren Aufwand, der die Entwicklungsgeschwindigkeit temporär noch weiter reduziert. Schnell findet man sich in einem Teufelskreis wieder: Die Geschwindigkeit bleibt immer weiter hinter den Erwartungen zurück, das Team spürt dadurch immer stärkeren Druck und geht immer mehr

Abb. 4.: Flache Aufwandskurve

Abb. 5: Steile Aufwandskurve

Qualitätskompromisse ein. Diese reduzieren wiederum die Entwicklungsgeschwindigkeit und so weiter.

Interessanterweise wissen die meisten Entwicklungsteams von den Qualitätsproblemen, lange bevor diese in den Metriken sichtbar werden. Schließlich ist das Team diese Qualitätskompromisse ja auch eingegangen – in der Regel mehr oder weniger bewusst. Wenn es also gelingt, dieses implizite Wissen des Teams explizit zu machen, kann man Qualitätsprobleme frühzeitig angehen. In Scrum hat der Scrum Master unter anderem die Aufgabe, genau dafür zu sorgen. Er arbeitet eng mit dem Team zusammen und bewahrt doch eine gewisse Distanz. Er kann Qualitätsprobleme frühzeitig bemerken und diese zur Sprache bringen.

Pragmatisch können Team und Scrum Master zum Beispiel nach dem folgenden Schema vorgehen: Wenn im Daily Scrum ein Entwickler seinen Fortschritt berichtet, fallen häufig Aussagen wie „Mit Feature XYZ bin ich fertig. Eigentlich müsste man noch …“ Dieses „Eigentlich müsste man noch …“ ist oft ein Indiz dafür, dass das Feature eben doch nicht vollständig (Done) implementiert wurde. Es ist mindestens noch ein technischer Task zu erledigen, um den Code in eine saubere Form zu bringen. Der Scrum Master fragt also direkt nach: „Wenn man dort eigentlich noch was tun müsste, sollten wir einen Task dafür am Taskboard haben. Lasst uns den Task schreiben!“ Damit ist jetzt für jeden sichtbar, dass das Feature eben doch nicht fertig ist, und das Team kann diskutieren, wie es damit umgehen will.

In Kanban muss es nicht zwingend eine Rolle wie den Scrum Master geben, aber das geschilderte Prinzip bleibt dasselbe: Eine Person (egal ob Teammitglied, Projektleiter, interner Coach usw.) übernimmt die „Führung“ (Leadership) und stellt im richtigen Moment die richtigen, meist unangenehmen Fragen: Wenn nicht im Standup-Meeting, dann spätestens in einer Retrospektive.

ZUSAMMENFASSUNG

Qualität ist keineswegs die Übereinstimmung von Anforderung und Umsetzung. Was (externe) Qualität ist, definieren die Kunden. Daher ist es sinnvoll, die Kunden während des ganzen Entwicklungsprozesses eng mit einzubinden. Daraus resultieren hohe Herausforderungen an die interne Qualität. Auch hier definieren die Kunden, was Qualität ist. Nur sind die Kunden hier nicht die Anwender des Systems, sondern die Entwickler selbst, die Tester und ggf. andere technische Rollen. Auch hier ist eine enge Einbindung dieser Kunden in den Entwicklungsprozess sinnvoll. Scrum tut dies, indem alle diese Rollen im Team integriert werden. Kanban unterstützt durch die WIP-Limitierung den Fokus auf (dauerhaft) kurze Durchlaufzeiten sowie das Konzept der Demand Analysis (und die entsprechende Visualisierung) ebenfalls eine enge Zusammenarbeit zwischen diesen Rollen. Voraussetzung dafür ist der inkrementelle Entwurf: Das System muss so gestaltet sein, dass immer wieder kleine Änderungen mit geringem Aufwand vorgenommen werden können.

Links & Literatur

[1] Weinberg, Gerald M. „Quality Software Management, Volume 1: Systems Thinking“, Dorset House, 1991, Seite 7

[2] John Medina: „Gehirn und Erfolg. 12 Regeln für Schule, Beruf, Alltag“, Spektrum Akademischer Verlag 2009, Seite 95

[3] Anderson, David J.: „Kanban. Evolutionäres Change Management für IT-Organisationen“ dpunkt.verlag 2011, Kapitel 6

[4] Stefan Roock: „Flexible Architekturen – Mit Handwerkskunst zukunftssichere Systeme bauen“ in Business Technology 2.2010

[5] Roman Pichler, Stefan Roock (Hrsg.): „Agile Entwicklungspraktiken mit Scrum“, dpunkt.Verlag 2011

[6] Robert C. Martin: „Clean Code“ Prentice Hall, 2008

Dipl.-Inform. Stefan Roock

ist Senior IT-Berater bei der it-agile GmbH in Hamburg. Er verfügt über mehrjährige Erfahrung aus agilen Softwareprojekten (Scrum, eXtreme Programming, Feature-driven Development) als Coach, Trainer, Scrum-Master/Facilitator und Entwickler. Darüber hinaus hat er zahlreiche Artikel und Tagungsbeiträge über agile Softwareentwicklung verfasst und ist Autor der Bücher „Software entwickeln mit eXtreme Programming“ und „Refactorings in großen Softwareprojekten“.

Dr. Arne Roock

ist Trainer und Coach bei der it-agile GmbH in Hamburg. Er beschäftigt sich seit Längerem mit Lean Thinking und Software-Kanban, hat das Kanban-Standardwerk von David Anderson ins Deutsche übersetzt und zahlreiche Artikel zu diesen Themen publiziert. Seine aktuellen Überlegungen veröffentlicht er in seinem Blog unter http://www.software-kanban.de.