AGILES PROJEKTMANAGEMENT

# Theoretisch kann ich praktisch alles!

Der Übergang aus der Theorie in die Praxis entscheidet über den Erfolg agiler Projekte. Wie kann das in einer VUCA-Welt funktionieren?

Das englische Akronym VUCA steht für volatility (Unbeständigkeit), uncertainty (Unsicherheit), complexity (Komplexität) und ambiguity (Mehrdeutigkeit). Doch Augenblick, genau dafür sind doch agile Konzepte da, um komplexe Vorhaben erfolgreich umzusetzen. Und jetzt soll die Umsetzung an der Komplexität des Projekts und seines Umfelds scheitern? Das scheint ja so gar nicht zusammenzupassen. Lösen wir den Widerspruch Schritt für Schritt auf.


Quelle: Martin Fowler [5]

**Die vier Quadranten** technischer Schulden nach Fowler (Bild 1)

## Theorie und Praxis verbinden

Die Umsetzung von Theorie in die Praxis scheint schwieriger zu sein, als es auf den ersten Blick aussieht. Gleichzeitig ist dieser Schritt essenziell für den Erfolg eines Projekts. Das lohnt einen genaueren Blick. Das hat sich auch die Unternehmensberatung Ernst & Young gedacht und eine interessante Studie durchgeführt [1]. Dabei ging es darum, die Erfolgsfaktoren bei der Strategiefindung und -umsetzung von Organisationen genauer zu erfassen. Heraus kam, dass es für eine Organisation wesentlich entscheidender ist, eine Strategie überhaupt umsetzen zu können, als es die inhaltliche Qualität der Strategie selbst ist.

Die beste Strategie nützt also nichts, wenn wir nicht grundsätzlich in der Lage sind, eine Strategie über die Hierarchien und die verschiedenen Bereiche einer Organisation umzusetzen. Zwei Jahre später fand eine Untersuchung der Zeitschrift „Fortune" über zentrale Fehler von CEOs heraus, dass in circa 70 Prozent aller Fälle die mangelhafte Umsetzung der Strategie die wesentliche Ursache für die Probleme war [2].

Unsere Fähigkeit, Ideen und Konzepte auch innerhalb einer Organisation umsetzen zu können, ist also von besonderer Bedeutung. Das ist bei der Umsetzung agiler Konzepte nicht anders, da sie ähnlich wie Strategien eine Organisation grundlegend ausmachen beziehungsweise bei ihrer Einführung mehr oder weniger stark verändern. Was genau wirft uns dabei immer wieder zurück? Ähnlich wie die Mütter des Erfolgs sind die Väter des Misserfolgs zahlreich und miteinander verwoben. Drei oftmals kritische Aspekte bei der praktischen Anwendung agiler Werte und Prinzipien betrachte ich im Folgenden etwas genauer:

- technische Schulden,
- Backlog und Prioritäten,
- und die inflationäre Nutzung des Begriffs agil.

## Technische Schulden

Der Begriff „technische Schulden" ist eine Metapher, die auf Ward Cunningham zurückgeht [3]. Er bezeichnet damit negative Aspekte der Programmierung, die trotz umsichtiger Arbeit eine Überarbeitung des Codes notwendig machen. Ansonsten verringert sich später die Geschwindigkeit der weiteren Entwicklung entsprechend. Damit schließt er bewusst schwache Softwareentwicklung, die sich nicht an die gängigen Standards, etablierte Methoden und passende Patterns hält, aus [4].

Martin Fowler differenziert technische Schulden in vier verschiedene Arten [5]. Technische Schulden können absichtlich und bewusst oder unabsichtlich aufgenommen worden sein. Sie können trotz umsichtiger Arbeit entstanden sein oder sich einfach nur aus Leichtsinn oder Sorglosigkeit ergeben haben (Bild 1). Die ursprüngliche Bedeutung der Schulden-Metapher ist der obere rechte Quadrant. Die Entscheidung für technische Schulden wird bewusst und umsichtig getroffen. Der untere, rechte Quadrant ergibt sich jedoch zwangsläufig aus der Komplexität der Aufgaben. Komplexe Aufgaben können nicht vollständig vorab erfasst und geplant werden. Daher sind wir (hoffentlich) hinterher klüger und können unseren Code spätestens dann entsprechend anpassen [6].

Was hat das mit der Umsetzung agiler Konzepte zu tun? Technische Schulden, die wir umsichtig und mit Bedacht eingehen, sind Teil des Risikomanagements. Dabei ist es egal, ob wir die technischen Schulden geplant oder unabsichtlich eingehen. Risikomanagement – damit sind wir bei den Grundlagen des Projektmanagements angekommen.

Agilität ist das Konzept zur Umsetzung komplexer Vorhaben. Damit ist Agilität selbst eine zentrale Maßnahme des Risikomanagements. Wir entwickeln unsere Organisation hin

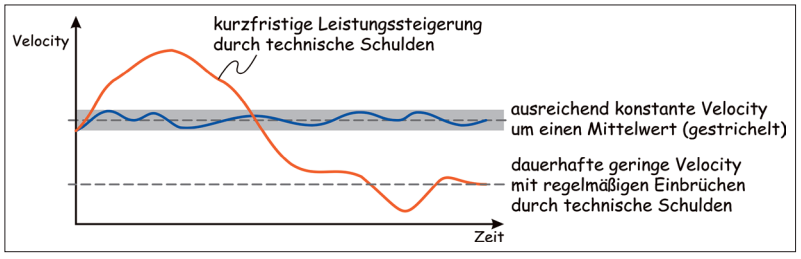
zu einer agilen Organisation, um komplexe Vorhaben bewältigen zu können. Davon steht aber nichts im Scrum Guide! Darin wird das agile Projektmanagement-Framework Scrum beschrieben, nicht mehr und nicht weniger [7].

Es gibt jedoch zahlreiche andere Disziplinen, die zusätzlich benötigt werden. Im Scrum Guide finden Sie nichts zu den Themen Anforderungsanalyse, Requirements Engineering, Qualitätssicherung und so weiter. Dafür gibt es andere, passende Quellen. Und es gibt grundlegende Projektmanagementaspekte, die Scrum unterstützen [8] [9]. Zu diesen zählen zum Beispiel die vorbereitenden Maßnahmen, die zur Entscheidung für die Durchführung eines Projekts und seiner Umsetzung mit Scrum führen, und Aspekte, die die Lieferungsprozesse betreffen oder die spätere Übergabe an die Wartung und Pflege behandeln. Natürlich finden sich alle diese Themen im Scrum-Backlog in entsprechenden User Stories wieder und werden darüber organisiert.

Zum Kern dieses grundlegenden Projektmanagements gehört ein initiales Risikomanagement [9]. Dieses Risikomanagement läuft parallel zur agilen Umsetzung des Projekts mit Scrum fortlaufend weiter. Dort werden auch die technischen Schulden betrachtet und bewertet sowie die entsprechenden Maßnahmen initiiert, um diese technischen Schulden abzutragen.

Dann ist doch alles in Butter. Warum lasse ich mich hier so lang und breit darüber aus? Zum einen sollten wir ehrlich sein: Wo wird wirklich ein wirkungsvolles Risikomanagement betrieben? Zum anderen haben technische Schulden eine Eigenschaft, die sie auch im Risikomanagement so schwer greifbar macht: Ihre negative Wirkung entfaltet sich erst spät und schleichend. Es geht einfach zu lange gut. Abstrakt ausgedrückt ist die Rückkopplungsschleife bei technischen Schulden meist zu lang, als dass uns die Konsequenzen aus den technischen Schulden wirklich bewusst werden. Und wenn sie bemerkt werden, ist es oft schon sehr spät. Um in der Metapher zu bleiben: Die Schulden haben sich so weit angehäuft, dass wir unter der Zinslast zusammenbrechen (Bild 2)!

Wir kennen dieses Phänomen als sprichwörtliche Leichen im Keller, bezeichnen es als faule Kompromisse oder beschreiben es mit dem Ausspruch, dass Provisorien länger leben als gedacht. Egal wie Sie es bezeichnen, die Umsetzung agiler Konzepte ist dabei heftig aus dem Ruder gelaufen. Meist unter äußerem Druck haben wir gedacht, eine Abkürzung nehmen zu können. Es sieht ja auch lange Zeit so aus, als ob alles gut ginge. Doch dann gerät unser Projekt ins Stocken. Wir müssen so viel anpassen und umfangreiche Refactorings durchführen, dass es, wenn überhaupt, nur noch äußerst geringen Projektfortschritt gibt. Und selbst der bricht immer wieder ein. Damit ist unser Projekt nicht mehr steuerbar (Bild 2).




Quelle: Uwe Vigenschow [9]

**Die Projektfortschrittsgeschwindigkeit** (Velocity) ohne beziehungsweise mit getilgten technischen Schulden (blau) und mit technischen Schulden (orange). Ungetilgte technische Schulden geben unserem Projekt einen kurzen Boost, den wir danach (zu) teuer bezahlen (Bild 2)

## Es gibt keine Abkürzungen

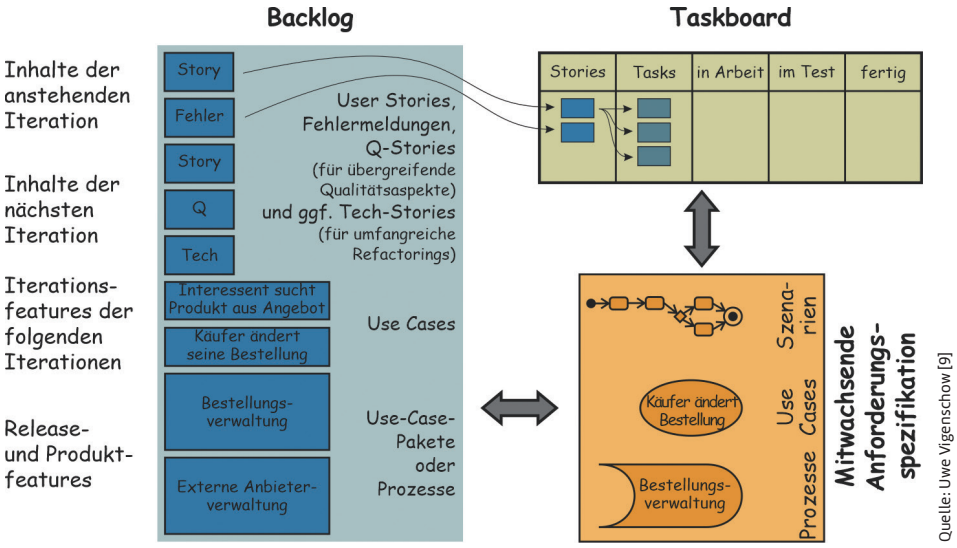
Wie hätten wir es besser machen können? Mit dem Eingehen technischer Schulden muss stets auch der Plan für deren Abzahlung beinhaltet sein. Genau diesen Zusammenhang beschreibt die Metapher. Wenn wir ein Haus kaufen und dafür die notwendige Finanzierung aushandeln, steht gleichzeitig auch der Plan für die Rückzahlung inklusive der Verzinsung. Wir wissen auf den Euro genau, was wann auf uns zukommt und wie das unseren restlichen finanziellen Spielraum einengt. Genauso methodisch müssen wir auch bei den technischen Schulden vorgehen. Wir machen einen Rückzahlungsplan und wissen um die Einschränkungen beim Fortschritt in der Entwicklung neuer beziehungsweise bei der Anpassung bestehender Funktionen.

Die Metapher der technischen Schulden ist entstanden, um Stakeholdern ohne technischen Hintergrund die Zusammenhänge begreifbar zu machen, die bestimmte Kompromisse auf die weitere Entwicklung haben. Um mit dem Druck, der über die Stakeholder auf das Projektteam ausgeübt wird, angemessen umzugehen, gilt es verständlich und nachvollziehbar zu argumentieren und zu diskutieren. Die lange dauernde Rückkopplungsschleife technischer Schulden macht diese Diskussionen gerade mit Nichttechnikern besonders anspruchsvoll. Doch nur so können wir zu tragfähigen Kompromissen und nachhaltigen Lösungen kommen, die wir benötigen, um in unserer Softwareentwicklung agil zu sein.

## Backlog und Prioritäten

Das Backlog ist ein zentrales Element, um agil zu arbeiten. Damit wird sichergestellt, dass die Prioritäten der Arbeit in den Teams für alle Mitarbeitenden klar und eindeutig sind. Das Backlog ist eine eindeutig geordnete Liste von Aufgaben, die zu bewältigen sind. Die Product Owner sind für ihre Backlogs verantwortlich und setzen damit den Fokus des aktuellen beziehungsweise nächsten Sprints für das Entwicklungsteam. Das Backlog ist das zentrale Steuerungselement der Product Owner (Bild 3, links).

Bei der Arbeit mit einem Backlog werden wichtige Elemente bei der Bewältigung komplexer Aufgaben sofort deutlich. Ein komplexes Vorhaben ist dadurch definiert, dass wir es nicht vorab vollständig erfassen können. Daher können wir ebenso wie unsere fachlichen Ansprechpartner auf Kundenseite die Anforderungen nicht vollständig erfassen. Wir gehen daher iterativ und inkrementell, also schrittweise vor. Meist ist der Kern des gewünschten Produkts bereits klarer als andere Aspekte. Daher fangen wir dort mit unserer Zerlegung der Anforderungen an und überführen diese in ein Backlog. ▶




Quelle: Uwe Vigenschow [9]

**Schema eines Backlogs** für größere Projekte mit mehreren Teams (Bild 3)

Die Aufgaben an oberer Stelle müssen bestimmten Anforderungen genügen, um direkt umsetzbar zu sein. Sie sind eindeutig und vom Umfang her klein genug, damit sie von einem Team innerhalb eines Sprints von wenigen Wochen in Produktqualität umgesetzt werden können. Der Fortschritt der Anforderungsanalyse und der damit verbundenen Zerlegung der Aufgaben von größeren Blöcken wie Features oder Epics in User Stories wird ebenfalls im Backlog sichtbar. Was bereits klar ist, ist auch heruntergebrochen, in kleine User Stories zerlegt, wandert nach oben und kann umgesetzt werden. Die Stories werden dann im Planning zu Beginn eines Sprints auf das Taskboard des Teams überführt, auf dem diese dann in technische Tasks zerlegt werden, um sie im Sprint umsetzen zu können (Bild 3 rechts oben).

Die für die Entwickler notwendige Granularität der User Stories wird nur für den nächsten Sprint (und bei größeren Projekten höchstens noch für den übernächsten Sprint) erreicht. Je weiter wir im Backlog herunterwandern, desto größer und damit auch noch unschärfer sind die Aufgaben formuliert. Die Flexibilität, die wir benötigen, um auf Veränderungen bei den Anforderungen und im Umfeld reagieren zu können, kann sich daher nur auf die Teile des Backlogs beziehen, die nach dem nächsten Sprint liegen. Dieser inkrementelle und iterative Prozess ist der Kern agilen Vorgehens.

So weit ist das doch logisch und kann direkt so gemacht werden. Ein paar kleine Anpassungen hier und da, und es läuft. Was wir jedoch häufig sehen, sind überfrachtete Backlogs, die starr geworden sind und auch den Projektteams keine Richtung mehr geben können. Was ist passiert?

Auf der zweiten Seite des Agilen Manifests finden wir 17 agile Prinzipien [10]. Dort steht unter anderem: „Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell." Diesem Prinzip unter dem Druck der Stakeholder zu folgen ist die eigentliche Arbeit der Product Owner. Hier trennt sich die Spreu vom Weizen. Perfektion ist dann erreicht, wenn wir nichts mehr weglassen können, und nicht, wenn wir nichts mehr hinzufügen können.

Die Diskussionen mit Stakeholdern darüber, dass ihre gewünschten funktionalen Erweiterungen nicht umgesetzt werden, weil es zum Beispiel bereits einen anderen Weg dafür gibt, der für andere, wichtigere Stakeholder gut funktioniert, sind schwierig und ermüdend. Hinzu kommt, dass, wenn wir Produkte entwickeln oder im Kundenauftrag Individualsoftware erstellen, ein wirtschaftlicher Druck entsteht, immer neue Features und Varianten einzubauen. Der einfache Weg, dem Druck nachzugeben, macht dann in der Folge unser Vorgehen immer starrer und unflexibler. Die Komplexität unsere Software wird über alle Maßen erhöht, was dann weitere Probleme nach sich zieht.

Wir brauchen eine Balance zwischen wirtschaftlichen Zielen und nachhaltiger Softwareentwicklung, die immer wieder neu ausgehandelt werden muss. Das Verhandlungsgeschick der Product Owner ist gefragt, um ihre Projekte dauerhaft zum Erfolg zu führen. Wir werden also weder die ideale Softwareentwicklung erreichen noch die Gewinne maximieren können. Satisficing, also eine ausreichend gute Softwareentwicklung, die den Zweck hinsichtlich der Funktionalität und der inneren Qualität erfüllt, ist die Lösung [9]. Diese Balance ist dynamisch und muss daher regelmäßig überprüft werden. Immer wieder wird eine neue Balance zu finden sein.

Noch eine kurze Bemerkung zu Bild 3. Im Backlog werden auch Sonderfälle wie Q-Stories und Tech-Stories gezeigt. Für kleine Projekte mit einem bis drei Teams ist das nicht sinnvoll und überflüssig. Die technischen Aspekte und die nichtfunktionalen Qualitätsthemen werden dort von den Teams implizit bei der Umsetzung der fachlichen User Stories mit erledigt. Erst bei größeren Projekten, in denen die Kommunikation und Abstimmung zwischen den Teams deutlich schwieriger wird, kann es sinnvoll sein, diese Themen im Gesamtbacklog im Sinne maximaler Transparenz für alle Teams sichtbar zu machen. Wir kommen später noch darauf zurück.

## Wir können es nicht allen recht machen

Den Umfang eines Softwareprodukts zu steuern ist Schwerstarbeit. Die Product Owner sind permanent am Verhandeln. Trotz aller Bemühungen, einen guten Ausgleich der Interessen zu finden, können wir es nicht allen recht machen. Um ein nachhaltig erfolgreiches Produkt zu schaffen, müssen wir dessen Komplexität minimieren, was weitgehend gleichbedeutend ist mit einem minimalen Funktionsumfang.

Wir erkennen daran, dass ein agiles Entwicklungsvorgehen nicht auf die Softwareentwicklung begrenzt ist, sondern mindestens alle Stakeholder mitbetrifft. Die dafür zu treffenden Entscheidungen der Product Owner sind zu respektieren. Sorgen unzufriedene Stakeholder per Eskalation zum Beispiel für ein „Durchgreifen der Führungskräfte", brauchen wir uns über Agilität keine weiteren Gedanken mehr zu machen. Sie verschwindet durch die Hintertür.

Für größere Projekte fällt jeweils dem Produktmanagement bei Produktherstellern beziehungsweise dem Vertrieb bei Projektfirmen eine besondere Bedeutung zu. Hier werden die ersten Weichen gestellt. So kann zum Beispiel ein Produkthersteller nicht den ganzen Markt befriedigen. Es wird verschiedene Zielgruppen beziehungsweise Marktsegmente geben. Damit gibt es auch immer Segmente, die für die Produktvorgaben wegfallen. Wenn dem nicht so wäre, gäbe es in Windows 11 immer noch einen DOS-Kompatibilitätsmodus. Wer zu jung ist, um zu wissen, was damit gemeint ist, darf froh darüber sein. Ich erinnere mich noch an die aufgebrachten Artikel, als Microsoft dieses Feature 2001 mit der 64-Bit-Version von Windows XP abgekündigt hat. MS-DOS-Anwender waren aber keine Zielgruppe mehr, und die Entscheidung war damals nur konsequent und – auch im Nachhinein betrachtet – richtig.

## Alles ist agil

Ohne das Adjektiv „agil" geht heutzutage offensichtlich in der Softwareentwicklung und darüber hinaus nichts mehr. Gefühlt ist alles agil. Doch wenn alles agil ist, ist es auch gleichermaßen nicht agil. Etwas kann nur im Kontrast zu seiner andersartigen Umgebung wahrgenommen werden. Der dunkle Fleck auf dem weißen Hemd fällt sofort ins Auge, auf einem stark gemusterten Hemd eher weniger. Durch den inflationären Gebrauch des Wortes „agil" wird seine Bedeutung unscharf. Dave Thomas, einer der 17 Verfasser des agilen Manifests, hat das bereits vor zehn Jahren erkannt [11].

Die Konsequenzen sind jedoch dramatisch, wenn wir versuchen, mit unserer Softwareentwicklung wirklich agil zu werden. So wird zum Beispiel ein eher chaotisches Vorgehen als „agil" gelabelt oder das Eingreifen in das Backlog an den Product Ownern vorbei mit „Wir sind doch agil" begründet. Das methodische, am Agilen Manifest ausgerichtete Vorgehen gerät mehr und mehr ins Hintertreffen. Hier sind vor allen Dingen die Scrum Master gefordert, aufzuklären, zu erklären und immer wieder gegenzusteuern.

Ein tiefes Verständnis von Agilität kann nur über eigene Erfahrungen erreicht werden. Das braucht Zeit. Wenn der Grad der Agilität einer Entwicklungsorganisation erhöht werden soll, benötigt das eine entsprechende Planung und Begleitung. Wie jede Organisationsentwicklung kostet das verfügbare Kapazität und kann nicht einfach so nebenbei erfolgen oder gar verordnet werden. Wir gehen dabei vom Kern agilen Vorgehens aus und prüfen bei jedem nächsten Schritt und jeder nächsten Erweiterung, dass diese zu den agilen Werten und Prinzipien des Agilen Manifests konform sind.

Gerade beim Skalieren eines Projekts und angesichts von immer mehr Teams, die parallel arbeiten, ist es von besonderer Bedeutung, methodisch sauber vorzugehen. Wie im Beitrag „Ist Agilität tot" [12] in dieser dotnetpro-Ausgabe aufgezeigt, treten viele Probleme bei der Umsetzung von Agilität mit der Skalierung beziehungsweise dem damit verbundenen Wachstum auf. Die Lösung kann nur aus den agilen Werten und Prinzipen abgeleitet werden.

Kommen wir kurz noch einmal auf Bild 3 zurück. Dort sehen wir zwei Erweiterungen von Scrum: Q-Stories und Tech-Stories. Das kann und sollte kritisch gesehen werden. Wir führen solche Erweiterungen nur ein, wenn sie uns helfen, agile Werte besser umzusetzen. Werden sie nur selten und mit Bedacht eingesetzt und unterstützen sie die Transparenz und direkte Kommunikation zwischen den Teams, kann das sinnvoll sein. Ansonsten bleiben wir beim Kern von Scrum und lassen solche Erweiterungen am besten weg.

## Einen Lernprozess etablieren

Wie schaffen wir es, wirklich agil Software zu entwickeln und nicht nur ein Label auf einen Entwicklungsbereich zu kleben, der nicht nach den agilen Werten und Prinzipen arbeitet? In Scrum ist ein einfacher Lernprozess für ein Team bereits beschrieben: der Inspect-and-Adapt-Regelkreis (Bild 4).

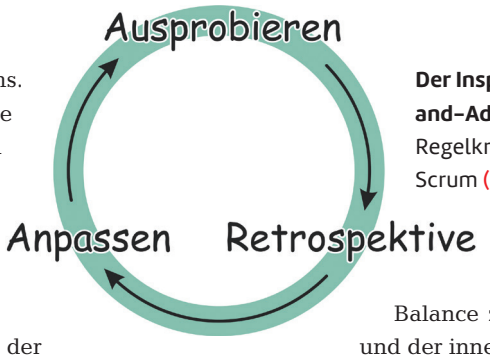
„In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann, und passt sein Verhalten entsprechend an." So lautet das letzte der Prinzipien aus dem Agilen Manifest [10]. Es beschreibt den Regelkreis aus Bild 4 und funktioniert sehr gut für einzelne Teams. Beim Skalieren braucht es übergreifende Regelkreise und darauf aufbauende Lernprozesse. Zu betrachten ist dabei das Lernen in jedem einzelnen Team, in der Entwicklungsorganisation mit mehreren Teams bis hin zur ganzen Organisation. Dabei geht es stets darum, Wege zu finden und zu etablieren, wie Methoden und Konzepte konkret praktisch umgesetzt werden können. Lernen kann nur aus der konkreten Erfahrung heraus erfolgen.

Die Scrum Master setzen den Lernprozess auf und begleiten ihn. Sie achten darauf, dass Lösungsideen konform zu den agilen Werten und Prinzipien sind. Sie holen gegebenenfalls die notwendigen Experten dazu, um fachliche oder technische Fragestellungen zu beantworten oder bei der Antwortfindung zu unterstützen.

## Etablierte Methoden einsetzen

Die gute Nachricht: Wir müssen das Rad nicht neu erfinden. Ganz im Gegenteil besteht das Problem oft darin, aus der Menge der publizierten Möglichkeiten an agilen Praktiken die passende zu finden. Auch hierbei stehen die Scrum Master in der zentralen Verantwortung, ebenso wie natürlich bei der Einführung und Anwendung der neuen Praktik. Ein ungenügendes Verständnis von Agilität kann dann dazu führen, dass eine an sich passende und hilfreiche Praktik doch nicht zum Ergebnis führt. Nicht selten werden agile Praktiken unvollständig eingeführt oder wichtige, aber auf den ersten Blick nervige Aspekte der Praktik nicht mehr durchgeführt. Erneut wird versucht, Abkürzungen zu finden, wo es keine gibt.

Ein Beispiel dafür ist das Zerlegen einer User Story auf dem Taskboard in einzelne Tasks (Bild 3 rechts oben). Dieser Schritt, der meist im zweiten Teil des Planning Meetings zu Beginn jedes Sprints stattfindet, soll einerseits zu einzelnen Tasks führen, die jede für sich an ungefähr einem Arbeitstag abgearbeitet werden können. Natürlich gibt es Abhängigkeiten zwischen den Tasks, die sich auf dem Board auch sichtbar machen lassen. Andererseits schaffen wir während des Zerlegeprozesses ein tiefes Verständnis im Team für die Aufgabe, ihre Umsetzung und das zu erreichende Ergebnis. Beides ist von großer Bedeutung für den gleichmäßigen und erfolg- ▶




**Der Inspect-and-Adapt-Regelkreis in Scrum (Bild 4)**

reichen Fortschritt der Arbeit eines Teams. Doch es kostet auch Zeit und ist intensive Arbeit. Daher wird es gerne auch mal eingespart. Man weiß ja, was zu tun ist.

Die Konsequenzen aus dieser vermeintlichen Abkürzung zeigen sich oft erst später in den folgenden Sprints. Die Aufgabe wurde doch nicht vollständig durchdrungen und jetzt fehlen Teile der Umsetzung. Oder der Entwickler, der sich der Aufgabe angenommen hat, wird krank oder muss kurzfristig eine andere Aufgabe übernehmen. Wer kann jetzt übernehmen und einspringen? Die Zerlegung ist, wenn überhaupt, nur im Kopf entstanden und schwer kommunizierbar. Zwischenergebnisse sind nicht transparent. Dumm gelaufen …

Methoden und Praktiken sind stets vollständig zu übernehmen. Weiterentwicklungen beziehungsweise individuelle Anpassungen können bestenfalls nach einer längeren Arbeit und dem tiefen Verständnis der Methode beziehungsweise Praktik erfolgen. Für alle Anpassungen ist sicherzustellen, dass sie die agilen Werte und Prinzipien unterstützen. Nur so kann es dauerhaft funktionieren.

## Zusammenfassung

Der Transfer der agilen Werte, Prinzipien und Praktiken aus der Theorie in die Praxis ist keine theoretische Frage. Sie ist essenziell für die Art und Weise, wie wir Softwareentwicklung verstehen und Einfluss darauf nehmen beziehungsweise sie weiterentwickeln. Dabei gibt es eine Reihe von Hürden zu nehmen, und eine enge Verbindung von Theorie und Praxis ist notwendig, um dafür effektive Lösungen zu entwickeln und eine ganzheitliche Sicht auf Agilität zu fördern.

Um Agilität wirklich zu verstehen und in der Anwendung zu durchdringen, braucht es Zeit. So lange werden die eingeführten Methoden vollständig umgesetzt und angewendet. Es wird monatelang nur gelernt und nicht verbessert, weil noch nicht klar sein kann, was wirklich eine Verbesserung darstellt. Erst dann wird variiert. Auch dabei verlassen wir uns auf etablierte Methoden, die wiederum vollständig eingeführt werden. Jede Veränderung oder Erweiterung muss konform zu den Werten und Prinzipien des Agilen Manifests sein. Die konkreten Lösungen für eine Vorgehensweise können nur aus den agilen Werten und Prinzipien abgeleitet werden, um Agilität zu erreichen. Es gibt keine Abkürzungen!

Ein wesentlicher Aspekt ist dabei der Druck, der aus wirtschaftlichen Rahmenbedingungen entsteht. Hier werden kurzfristige Notwendigkeiten, um zu überleben, gegenüber langfristig wertvollen Themen abgewogen. Wie bei der Betrachtung zu den technischen Schulden bedarf es dazu eines expliziten Risikomanagements, um den Fokus wieder auf die langfristigen Werte und Ziele zu lenken und sich nicht in den zahlreichen Ausnahmen zu verlieren.

Mit den wirtschaftlichen Themen wie auch mit der Dynamik im Umfeld (VUCA) geht eine agile Organisation mit den Mitteln und Wegen der Agilität um. Wir schaffen Transparenz durch ein entsprechendes Backlog, gehen inkrementell und iterativ damit um und machen Risken sowie unsere Maßnahmen zum Umgang mit den Risiken öffentlich. So können wir die Ideen aus den Teams nutzen, um damit angemessen umzugehen, und erreichen immer wieder eine nachhaltig wirksame Balance zwischen den wirtschaftlichen Zielen und der inneren Qualität unserer Software.

Wenn wir akzeptieren, dass unser Projekt und unser Umfeld komplex und damit nicht vollständig vorab planbar sind, kommen wir zwangsläufig dahin, unsere gesamte Organisation an den agilen Werten und Prinzipien auszurichten. Dann gehen die Abläufe Hand in Hand; die Teams und die Organisation lernen, wie sie mit der Dynamik erfolgreich umgehen können, und Entscheidungen werden schnell von den Experten und Expertinnen in den betroffenen Teams getroffen. So schaffen wir hervorragende Produkte beziehungsweise Dienstleistungen. Der wirtschaftliche Erfolg stellt sich als Folge in der Regel ebenfalls ein. ■

*[1] Ernst & Young, Measures That Matter, www.dotnetpro.de/SL2408TheoretischAgil1*
*[2] Ram Charan, Geoffrey Colvin, Why CEOs Fail, Fortune Magazine, Seite 68–78, 21. Juni 1999*
*[3] Ward Cunningham, Ward explains debt metaphor, www.dotnetpro.de/SL2408TheoretischAgil2*
*[4] Robert C. Martin, A Mess is not a Technical Debt, www.dotnetpro.de/SL2408TheoretischAgil3*
*[5] Martin Fowler, Technical Debt Quadrant, www.dotnetpro.de/SL2408TheoretischAgil4*
*[6] Uwe Vigenschow, Lernende Organisationen, dpunkt.verlag, 2021, ISBN 978-3-86490-798-2*
*[7] Ken Schwaber, Jeff Sutherland, Der Scrum Guide – Der gültige Leitfaden für Scrum: Die Spielregeln, November 2020, www.dotnetpro.de/SL2408TheoretischAgil5*
*[8] Ken Schwaber, Agiles Projektmanagement mit Scrum, Microsoft Press, 2007, ISBN 978-3-86645-631-0*
*[9] Uwe Vigenschow, APM – Agiles Projektmanagement, dpunkt.verlag, 2015, ISBN 978-3-86490-211-6*
*[10] Kent Beck et al., Manifest für agile Softwareentwicklung, www.dotnetpro.de/SL2408TheoretischAgil6*
*[11] Dave Thomas, Agile is Dead (Long Live Agility), www.dotnetpro.de/SL2408TheoretischAgil7*
*[12] Uwe Vigenschow, Ist Agilität tot?, dotnetpro 8/2024, Seite 6 ff., www.dotnetpro.de/A2408Agilitaet*

**Uwe Vigenschow**
ist Abteilungsleiter bei Körber Pharma Software. In seinem neuesten Buch „Lernende Organisationen" beschreibt er die Transformation hin zu der zukunftssicheren Gestaltung komplexer Aufgaben und Strukturen.

**dnpCode** A2408TheoretischAgil