OBJEKTspektrum ONLINE Themen special

Artikel

„Unser eigenes Scrum“

Gefahren von agilen Antipatterns

von Masud Fazal-Baqaie, Baris Güldali

Die steigende Komplexität und Dynamik der IT zwingt IT-Verantwortliche zunehmend zum Umdenken in der Softwareentwicklung. Insbesondere die Notwendigkeit, auf unvorhergesehene Umstände schnell zu reagieren, bewegt IT-Verantwortliche dazu, Entwicklungsvorhaben agil umzusetzen. Scrum hat sich in den letzten Jahren als der bekannteste agile Ansatz etabliert. In der Praxis erlebt man allerdings oft, dass gelebte Prozesse in vielen Dingen vom Scrum Guide und den darunter liegenden agilen Ideen abweichen. In diesem Artikel beschreiben wir, wie man Scrum für den eigenen Kontext anpassen kann, ohne in die Falle der agilen Antipatterns zu tappen.

Sowohl in der öffentlichen Wahrnehmung als auch in der Praxis haben in vielen Domänen leichtgewichtige, agile Entwicklungsprozesse, allen voran Scrum, schwergewichtige Vorgehensmodelle verdrängt [Kuh15]. Grund dafür ist das Versprechen agiler Methoden, weniger prozessualen Overhead zu erzeugen und schneller Ergebnisse zu liefern. Durch die kurzen Lern- und Feedback-Zyklen erleichtern sie den Umgang mit instabilen Anforderungen und wenig bekannten Technologien oder neuen fachlichen Themen.

Im Gegensatz zu schwergewichtigen Vorgehensmodellen beschreiben agile Methoden einige Grundregeln, die einfacher zu befolgen sind. Gleichzeitig werden Teams vor die Herausforderung gestellt, selbst effektive und effiziente Prozesse im Sinne der agilen Werte und Prinzipien umzusetzen und zu pflegen.

Allerdings gestaltet sich die „agile Transformation“ für Firmen nicht immer so einfach. Folgendes ist in der Praxis oft zu beobachten:

> Scrum wird als agiles Vorgehensmodell gewählt, aber bei der Einführung werden Scrum-Rollen gleich abgeschafft beziehungsweise Events nicht eingehalten (z. B. kein Scrum Master, keine Retrospektiven, kein Timeboxing für Events). Das führt dazu, dass man die zentralen Ideen hinter Scrum und seiner Zusammensetzung aus Events, Rollen und Regeln nicht

- kennen und verstehen lernt.
- Der Scrum-Eigenbau wird oft durch weitere Anpassungen ergänzt, die im Sinne des Agilitätsgedankens eigentlich kontrapoduktiv sind (z. B. Statusmeetings für das Management anstelle von Daily Stand-ups, fachliche Abnahmetests am Ende des Sprints statt täglicher Ergebnisse aus automatisierten Testläufen).

Solche Abweichungen von Scrum haben den Begriff „ScrumBut" geprägt. Dieser beschreibt in der Praxis oft vernommene Sätze wie:

Wir machen Scrum, aber …

- wir haben unser eigenes angepasstes Scrum.
- wir machen ein „Daily" Scrum pro Woche, das reicht.
- wir brauchen keine Retrospektiven.
- wir haben 8-Wochen-Sprints.
- wir haben keine „Definition of Done".

Während Abweichungen von Scrum nicht kategorisch schlecht sein müssen, sollten diese bedacht und gezielt angegangen werden. In diesem Artikel beschreiben wir:

- wie man Regelabweichungen und schlecht gelebte Praxis zu den eigenen agilen Entwicklungsprozessen erkennt,
- wieso man nicht zu früh von den Regeln abweichen sollte und wieso auch kleine Abweichungen eine große Auswirkung haben können,
- wieso dennoch kontinuierliche und gezielte Anpassungen dem agilen Gedanken entsprechen und notwendig sind.

Scrum in der freien Wildbahn

Wie in der Einleitung angesprochen, sind in der Praxis bestimmte zu vermeidende Muster beim Scrum-Einsatz immer wieder anzutreffen. Eine Online-Suche nach den Begriffen „Antipattern" und „Scrum" führt zu verschiedensten Mustern auf Internetseiten und Blogs (z. B. age-of-product.com/). Oft handelt es sich dabei um Ideen, die auf den ersten Blick naheliegend aussehen, aber dann doch Probleme verursachen, weil sie den agilen Werten und Prinzipien entgegenstehen. Im Folgenden gehen wir exemplarisch auf einige dieser Muster ein.

Nicht verstandene Rollen: Mit Scrum werden andere Rollen eingeführt, als man sie von klassischen Vorgehensmodellen kennt. Der Product Owner verantwortet die User Stories und ihre Akzeptanzkriterien, ein cross-funktionales Team verteilt eigenverantwortlich Aufgaben, der Scrum Master sorgt dafür, dass das Team ungestört arbeiten kann und die Regeln von Scrum eingehalten werden. Wenn die Zielsetzung dieser Rollen und insbesondere ihre Unterschiede zu den klassischen Rollen nicht ausreichend verstanden werden, führt das zwangsläufig zu Problemen. Da wird zum Beispiel der klassische Projektleiter zum Scrum Master, kämpft aber mehr mit Budget- und Managementfragen, als sich auf das Team zu konzentrieren; die Fachabteilung wird vom Kunden als Product Owner verstanden, fühlt sich aber nicht verantwortlich für den Produkterfolg; und die Entwickler- und Tester-Silos werden aufrechterhalten, selbst wenn diese in den gleichen Büros sitzen.

Falsch verstandene Agilität: In Teams ohne Scrum Master und mit wenig Vorwissen bezüglich Agilität wird diese oft mit Spontanität und einer gewissen Planlosigkeit gleichgesetzt. Dazu kommt dann oft, dass man das unkoordinierte Handeln in Ad-hoc-Prozessen auch noch als besonders fortschrittlich, besonders agil empfindet. Kurzfristige Plan-, Verantwortlichkeits- oder Scope-Veränderungen haben nichts mit Agilität gemein, insbesondere nicht mit Scrum. Scrum setzt das disziplinierte Befolgen von zeitlich befristeten Events voraus. Der Scope für den Sprint wird zu Beginn eingefroren und generell wird ein stabiles Team befürwortet.

Akute Nicht-Verfügbarkeit: Der Product Owner ist für das Entwicklungsteam während des Sprints kaum erreichbar und kann keine Rückfragen des Teams beantworten. Dadurch wird das Team gezwungen, einen „Mini-Wasserfall-Prozess" im Sprint durchzuführen. Entweder weicht das Team auf ausführlichere Vorabdokumentation der Anforderungen aus oder riskiert eine Umsetzung, die nicht den eigentlichen Wünschen des Product Owners entspricht.

Späte, viele manuelle Tests: Ein weit verbreitetes Antipattern ist mangelnde Testautomatisierung in Kombination mit manuellen

Tests. In Kombination mit einem inkrementell-iterativen Vorgehen sind manuelle Tests aufgrund der wiederholt notwendigen Regressionstests und der stetig wachsenden Anzahl von Regressionstestfällen sehr aufwendig. Dies führt dazu, dass Tests auf das Ende des Sprints verlagert werden und zudem sehr lange dauern. Dadurch wird die Timebox überschritten und gefundene Fehler können nicht mehr rechtzeitig bis zum Ende des Sprints behoben werden. Oft finden auch manuelle Abnahmephasen erst nach einer Reihe von Sprints statt oder zum Ende eines Releases. Entsprechend erfolgt die Rückmeldung an die Entwickler hier noch sehr viel später. Auch wenn Tests automatisiert und in Continuous Integration (CI) eingebunden sind, verursachen sie Abbrüche der Build-Pipelines, wenn diese schlecht implementiert sind.

Management Stand-up: Die Stand-ups als Format für Koordination und gegenseitige Unterstützung innerhalb des Entwicklungsteams weichen einem Status-Stand-up, in dem der Product Owner oder andere Stakeholder täglich den Status von Aufgaben oder User Stories abfragen. Schlimmstenfalls geht man dabei alle User Stories im Sprint durch. Damit ist das Entwicklungsteam gezwungen, auf einer Ebene zu kommunizieren, die für die entsprechenden Stakeholder verständlich ist, statt zum Beispiel technische Details auszutauschen. Außerdem ist die tägliche Besprechung von noch gar nicht begonnenen User Stories beziehungsweise Aufgaben schlicht ineffizient.

Fehlende Retrospektiven: Eine der Stärken von Scrum ist es, dass es empirischen Prinzipien folgt. Scrum kombiniert kurze Entwicklungszyklen mit regelmäßigen Terminen zur Reflexion. Damit sind die Retrospektiven ein wichtiger Erfolgsfaktor, damit sich eine Feedbackkultur und eine kontinuierliche Verbesserung einstellen. Nur dadurch können mittel- und langfristig mit Scrum schnelle, hochqualitative Softwareentwicklungsprozesse etabliert werden.

Warum sollte man Scrum in Reinform anstreben?

Wir sind keine Apologeten und religiösen Verfechter der einen oder anderen Methodik. In unseren bisherigen Arbeiten und in der täglichen Praxis betonen wir, dass die Entwicklungsprozesse bedarfsgerecht zum Team und dessen Umfeld passen müssen. Dennoch weichen unserer Erfahrung nach Teams oft allzu schnell von den Vorgaben, zum Beispiel des Scrum Guide [Scrum], ab. Dass es sinnvoll ist, bei der Adaption von agilen Prozessen und Scrum möglichst nahe am Original zu bleiben, hat verschiedene Gründe, die wir im Folgenden darlegen.

Zunächst gibt es ein Argument auf Basis des natürlichen Lernprozesses. Die Adaption von Agilität ist in der Regel mit einem Lernprozess verbunden. Im agilen Kontext zitiert man dabei gerne das Prinzip von „ShuHaRi“, das aus der japanischen Kampfkunst stammt und die drei Stufen des Lernens beschreibt:

- Auf der initialen Stufe „shu“ gilt es, zu gehorchen, um die Grundlagen anzueignen.
- Auf der folgenden Stufe „ha“ wird, wo angemessen, bewusst mit Traditionen gebrochen und von dem Vorgegebenen punktuell abgewichen, um auch Wirkzusammenhänge zu verstehen.
- Auf der finalen Stufe „ri“ spielen dann die Vorgaben keine Rolle mehr, wobei die Grundlagen prinzipiell ihre Gültigkeit behalten. Vielmehr ist man selbst Experte.

Kent Beck, Begründer des Extreme Programming (XP), unterscheidet passend dazu die drei Stufen der Reife von XP:

- Do everything as written.
- After having done that, experiment with variations in the rules.
- Eventually, don’t care if you are doing XP or not.

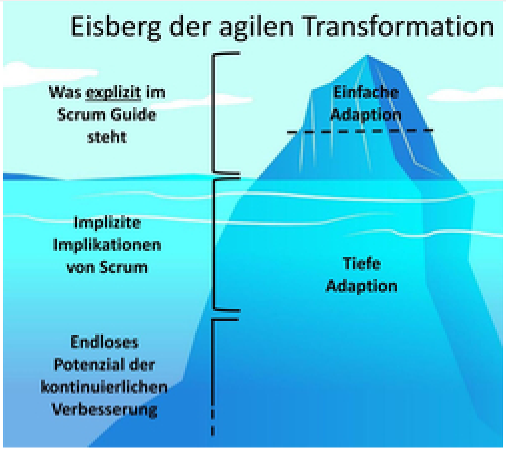

Abb. 1: Eisberg der agilen Transformation

Analog dazu zeigt Abbildung 1 den Eisberg der agilen Transformation (angelehnt an [Bun16]). Wie abgebildet und bereits dargestellt, findet in der Praxis oft nur eine einfache Adaption von Scrum statt, die sogar hinter dem Scrum Guide zurückbleibt. Selbst bei Befolgung des Scrum Guide ist noch sehr viel Potenzial für die wahre Adaption von Agilität im Sinne der Werte und Prinzipien vorhanden, das über die kontinuierliche Verbesserung ausgeschöpft werden kann.

Zusammengefasst ist es ratsam, erst dann von den Vorgaben von Scrum abzuweichen, wenn man ausreichend mit Softwareentwicklungsprozessen, insbesondere Scrum, agilen Werten und Prinzipien, sowie dem Team und dessen Umfeld vertraut ist.

Das nächste Argument ist in diesem Zusammenspiel durch die Bestandteile von Scrum begründet. Der Scrum Guide beschreibt einen sehr reduzierten Satz von grundlegenden Regeln. Die verschiedenen Vorgaben sind dabei aufeinander abgestimmt. Das führt dazu, dass schon eine einzelne Regelabweichung bereits mehrere unerwünschte Seiteneffekte haben kann.

Nehmen wir als Beispiel die Schätzgenauigkeit. In Scrum wird die Schätzgenauigkeit erfahrungsbasiert für einen relativ kleinen Umfang bestimmt. Verschiedene Bestandteile von Scrum unterstützen diesen Ansatz: Ein stabiles Team und eine feste Sprintdauer erleichtern es, aus der Erfahrung heraus zu schätzen. Ein fester, unveränderlicher Sprint-Scope erleichtert es, den abzuschätzenden Umfang zu bestimmen. Die festen Events „Planning“ und „Review“ zwingen dazu, die Schätzung und das tatsächlich Erreichte miteinander zu vergleichen. Mit der Retrospektive sollen Erkenntnisse daraus für zukünftige Sprints und damit Schätzungen abgeleitet werden.

Dieses kleine Beispiel sollte illustrieren, dass man vom Scrum Guide möglichst erst dann abweichen sollte, wenn man durch ausreichend Wissen und Erfahrung die Folgen und Seiteneffekte abschätzen kann.

Als letztes Argument möchten wir den berühmten Satz anführen, der auch auf Scrum zutrifft: Es ist einfach zu verstehen, aber schwierig zu meistern. Oft haben die Teams gerade in den ersten Wochen und Monaten nach der Einführung mit verschiedenen „Schmerzen“ zu kämpfen. Diese können aber durchaus ein positives Zeichen sein, setzten sie doch einen deutlichen Anreiz für das Team, die Schmerz verursachenden Umstände zu lindern.

Beispiele aus der Praxis sind, dass es schwerfällt, Stories klein genug zu schneiden (-> technische Voraussetzungen für kleinteiligere Veränderungen schaffen), die Timeboxes von Events einzuhalten (-> auf das Wesentliche konzentrieren, priorisieren) oder sich gemeinschaftlich auf die Sprintergebnisse zu committen (-> gegenseitige teaminterne Unterstützung, gemeinschaftliche Verantwortung).

Wenn Teams statt der Umstände, die zu den „Schmerzen“ führen, die Regeln abschwächen oder ignorieren, behindern sie damit mittel- und langfristig Verbesserungen hin zu agilen Werten und Prinzipien.

Warum sollte man Scrum kontinuierlich verbessern?

Scrum beschreibt ein Management-Framework, bei dem die beteiligten Rollen aufgefordert sind, die Lücken im Sinne der agilen Werte und Prinzipien auszufüllen. Die Reduktion des Scrum Guide auf einen kleinen Satz relativ einfacher explizit definierter Regeln ist dabei zugleich der Grund dafür, dass Scrum zum einen einfach zu verstehen ist, zum anderen aber viele konkrete

Ausprägungen haben kann.

Dabei sollte jedes Team seine konkrete Scrum-Ausprägung aktiv gestalten und weiterentwickeln, statt sie dem Zufall zu überlassen. Mittel dazu sind in erster Linie Vereinbarungen wie die „Definition of Ready“ und „Definition of Done“ sowie Plattformen wie die Retrospektive. Retrospektiven dienen dazu, die kontext-bezogenen Hindernisse aus dem Weg zu räumen und kontext-spezifische Verbesserungen zu etablieren. Ergänzend können auch mehrschrittige, systematische Verbesserungsinitiativen durchgeführt werden [FBS17]. Dabei sollte das Augenmerk insbesondere auch darauf liegen, wie aufgetretene Probleme in Zukunft vermieden werden können, also auf prozessualen und grundsätzlichen Verbesserungen unabhängig von der aktuellen Situation. Anregungen dazu geben wiederum Pattern-Sammlungen, die gut mit Scrum einhergehen, wie [Scr17].

Eine große Rolle spielt dabei die Berücksichtigung der individuellen Gegebenheiten. Beispielsweise spielen nicht selten die etablierte Unternehmenskultur oder gesellschaftliche Diskussionskultur eine große Rolle bei der Durchführung von Retrospektiven. Fehlende Transparenz-Kultur oder indirekte Kritik-Kultur können bei Retrospektiven zu Missstimmungen führen. Die Akzeptanz dieses Formats hängt somit sehr vom Fingerspitzengefühl des Scrum Masters ab. Deshalb muss die Ausgestaltung von Retrospektiven (und auch von anderen Scrum Events) die Gegebenheiten heterogener Teams berücksichtigen.

Beispielsweise könnte man für Retrospektiven die anonyme Sammlung von Verbesserungsvorschlägen einführen, falls dies kulturell notwendig ist. Ein weiteres Beispiel sind die Umstände bei verteilter Entwicklung. Verteilte Teams organisieren die Regeltermine örtlich und zeitlich anders als Teams, die sich am gleichen Ort oder sogar in gleichen Räumlichkeiten befinden. Die Verteilung der Teams hat auch einen Einfluss darauf, wie Informationen ausgetauscht werden und wie hoch der Dokumentationsaufwand ist. Als letztes Beispiel sei hier das Einbeziehen von dedizierten Experten (Security-Experten, Fachtester, Performanz-Analysten usw.) genannt. Dies macht oft die Vereinbarung von dedizierten (Regel-)Terminen oder zusätzlichen Aktivitäten notwendig. Zu beachten ist hier, dass dies nicht zu einem Rückfall in wasserfallartige Verhaltensweisen führen darf. In jedem Fall sollte man sich dann der „Process Debt“ bewusst sein und versuchen, sie mittelfristig abzubauen.

Bei allen vorgestellten Beispielen und weiteren Anpassungen und Verbesserungen gilt es zu berücksichtigen, dass die Veränderungen im Sinn der agilen Werte und Prinzipien [AgMan] sind. Neben der initialen Bewertung sollte auch die gelebte Praxis mit den Veränderungen hinterfragt werden. Jede Verletzung oder Nicht-Erfüllung der agilen Werte und Prinzipien kann auf ein Antipattern hindeuten und muss analysiert, verstanden und in Bezug auf seine „Process Debt“ abgewogen werden. Im Rahmen der Retrospektiven und kontinuierlichen Verbesserung kann diese dann abgebaut werden.

Take Aways

Unterschied verstehen: Doing Agile vs. Being Agile

Um wirklich agil zu sein, reicht es nicht, einfach nur den explizit beschriebenen Regeln agiler Entwicklungsprozesse zu folgen und zu hoffen, dass dies alleine schon für den Erfolg sorgen wird. Vielmehr ist es Aufgabe des Teams, die Regeln im agilen Sinne auszugestalten und sich immer weiter zu verbessern. Der Praxisalltag ist voller Überraschungen, auf die es zu reagieren gilt. Nur die Verinnerlichung des agilen Mindsets verschafft dem Team die Fähigkeit, auf unerwartete Ereignisse schnell zu reagieren und bedarfsgerechte Lösungen zu finden.

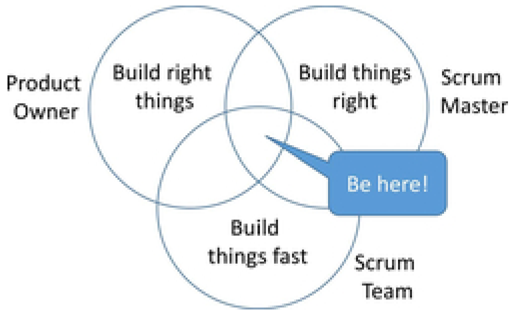

Abb. 2: Drei Rollen, drei Zuständigkeiten

Build the right things, build things right, build things fast

Zu einem erfolgreichen agilen Team tragen drei gut verstandene Rollen bei (siehe Abbildung 2):

Ein *Product Own*er, der dafür sorgt, dass das Product Backlog durchgehend mit relevanten Anforderungen befüllt und priorisiert wird, und der deren Wert und Komplexität klärt.

Ein *Scrum-Team* bestehend aus motivierten und engagierten Mitgliedern, die die Sprintziele in dem definierten Zeitrahmen zur Zufriedenheit des Product Owners erfüllen.

Ein *Scrum Master*, der die Hindernisse aus dem Weg räumt und es dem Team ermöglicht, sich auf ihre Sprintziele zu konzentrieren.

To be excellent or not to be excellent: that is the question

Agilität und insbesondere die Transformation aus der klassischen Entwicklung zur agilen Entwicklung erfordert viel Überzeugungsarbeit. Beteiligte Personen, die an ihrer historisch gewachsenen Rolle und ihren Aufgaben festhalten, müssen von den agilen Werten und Prinzipien überzeugt werden. Für Akzeptanz bei den Beteiligten und deren Umfeld zu werben, funktioniert am besten mit Ergebnissen. Das Team muss durch ein eingespieltes, immer weiter verbessertes Zusammenspiel in allen Disziplinen überzeugen: die vorbildliche Abstimmung von fachlichen Anforderungen mit Stakeholdern, das sinnvolle Schneiden von User Stories passend zur Sprintkapazität, modulare und gut anpassbare Architektur zur Integration von Inkrementen, unmittelbares Feedback auf Entwicklungsaktivitäten durch durchdachte CI-Pipelines, Vermeidung von Regressionen und Beherrschung von Testaufwänden durch Testautomatisierung.

Keine Angst vor der Retrospektive. Sie macht uns zu besseren Menschen

Die Retrospektive dient der kontinuierlichen Verbesserung durch Reflexion, und das nicht nur fachlich und technisch, sondern vor allem auch zwischenmenschlich. Neben offenen Aussprachen über Hindernisse und Verbesserungspotenziale sollten auch Lob und Bestätigung nicht fehlen. Dabei erarbeitet ein Scrum-Team nicht nur, was es nächstes Mal besser machen kann, sondern trägt damit dazu bei, dass das Vertrauen und die Wertschätzung zwischen den Teammitgliedern wachsen.

Literatur & Links

[AgMan] Manifesto for Agile Software Development, 2001, siehe: http://agilemanifesto.org/

[Bun16] R. Bunning, Illuminating the potential of Scrum by comparing LeSS with SAFe, 2017, siehe: https://www.slideshare.net/rowanb/illuminating-the-full-potential-of-scrum-by-comparing-less-with-safe

[FBS17] Softwareprozessverbesserung in der Praxis. Projektmanagement und Vorgehensmodelle (PVM) 2017, 129–143, GI, 2017, siehe auch: http://pvm-tagung.de/

[Kuh15] M. Kuhrmann, M. D. Fernandez, Systematic Software Development: A State of the Practice Report from Germany, in: IEEE 10th Int. Conf. on Global Software Engineering (ICGSE 2015), 2015, pp. 51–60

[Scr17] Scrum Pattern Community, ScrumPLoP 2917, 2017, siehe: http://www.scrumplop.org/

[Scrum] J. Sutherland, K. Schwaber, The Scrum Guide, Juli 2016, siehe: https://www.scrum.org/resources/scrum-guide

---

Dr. Masud Fazal-Baqaie

ist Managing Consultant der S&N CQM GmbH (http://www.sn-cqm.de). Zu seinen Arbeitsschwerpunkten gehört die Qualität und Verbesserung von Entwicklungsprozessen. Er ist stellvertretender Sprecher der Fachgruppe „Vorgehensmodelle für die betriebliche Anwendungsentwicklung“ und Mitglied im Leitungsgremium Produktmanagement in der Gesellschaft für Informatik.

E-Mail: masud.fazal-baqaie(at)sn-cqm.de

Dr. Baris Güldali

ist Managing Consultant der S&N CQM GmbH (http://www.sn-cqm.de). Seine Arbeitsschwerpunkte sind Test- und Qualitätsmanagement. Er ist Mitglied im Leitungsgremium der Fachgruppe TAV (Test, Analyse und Verifikation von Software) in der Gesellschaft für Informatik.

E-Mail: baris.gueldali(at)sn-cqm.de

Bildnachweise:

iStock.com/Kerrick, S&N CQM GmbH