Sonderdruck aus OBJEKTspektrum 02/2017

Agile und DevOps

Füttern Sie nicht das Undone-Monster!

Häufig erhalten Features den „fertig"-Stempel, obwohl noch Aufgaben offen sind. Diese „Undones" sind quasi Futter für das „Undone-Monster". Je größer es wird, desto eher kann es in Form von höheren Kosten, Terminverschiebung oder schlechter Qualität zuschlagen. Inkremente zu erzeugen, die wirklich „Done" sind, ermöglicht, die Wertschöpfungskette vom Business zum Kunden durchgängig und schnell zu gestalten. Agile und DevOps können dabei helfen – und die Innovationsfähigkeit fördern.

Was hat die Geldkarte mit vielen IT-Projekten zu tun? Auf den ersten Blick wenig, auf den zweiten sehr viel. Die Geldkarte ist ein Beispiel für ein durchdachtes Produkt, das trotzdem kaum Marktakzeptanz gefunden hat. Millionen Girocards sind mit einer Geldkartenfunktion ausgestattet, genutzt wird sie kaum.
Die Angst vor solchen teuren Fehlschlägen bringt viele Organisationen in ein Dilemma:

- Auf der einen Seite brauchen sie dringend Innovationen, um ihre Wettbewerbsfähigkeiten zu erhalten.
- Auf der anderen Seite können sie sich kostspielige Flops nicht leisten.

Das Problem besteht einfach darin, mit jeder neuen Idee das Risiko einzugehen, viel Zeit und Geld zu verschwenden.
Zur Lösung des Dilemmas gilt in der IT-Branche häufig Agilität als Mittel der Wahl, verspricht sie doch schnelle Rückmeldungen und damit eine gewisse Sicherheit, das Risiko überschaubar zu halten (vgl. [Schw12]). Das Prinzip dahinter: Je eher die Idee in Gestalt eines Produkts zum Kunden kommt, desto früher schaffen wir Wert. Seit einiger Zeit addiert sich „DevOps" zur Agilität hinzu. Ziel beider Konzepte: Die Wertschöpfungskette von der Idee bis zur Auslieferung so durchgängig zu gestalten, dass sie an jeder Stelle transparent ist. Die Wertschöpfungskette wird bereits komplett in den agilen Prinzipien beschrieben, DevOps verstärkt die Sicht auf die Automatisierung der Lieferung von der Entwicklung in den Betrieb, auf das Nutzen von Feedback bis zur Quelle.
Wir erläutern im Folgenden, wo die Schwachstellen liegen, wie Agilität und DevOps darauf reagieren und wie sich beide Ansätze ergänzen.
Ein anderes Wort für diese Schwachstellen sind die „Undones" – Aufgaben, die als gelöst deklariert wurden, es aber nicht sind. Um das zu erklären, beginnen wir mit den Prinzipien der Agilität (vgl. [Lar08]).

„Done" und seine Rolle innerhalb agiler Methoden

Gemäß den agilen Prinzipien gliedert sich die Entwicklung in Teilintervalle, die Sprints. Am Ende jedes Sprints soll ein lieferfähiges Inkrement „fertig" sein. Was „fertig" ist, steht in der „Definition of Done" und hängt vom jeweiligen Rahmen aus Mitarbeitern, Technologien und der Organisation selbst ab.
An der Sinnhaftigkeit dieses Ansatzes ist gar nicht zu rütteln. Nur setzen ihn einige Organisationen nicht so konsequent um, wie es nötig wäre, und sabotieren damit unbewusst die implizite Risikokontrolle. Letztendlich beschreiben die Punkte des agilen Manifestes einen Status, der immer angestrebt wird: Einen Zustand als „fer-

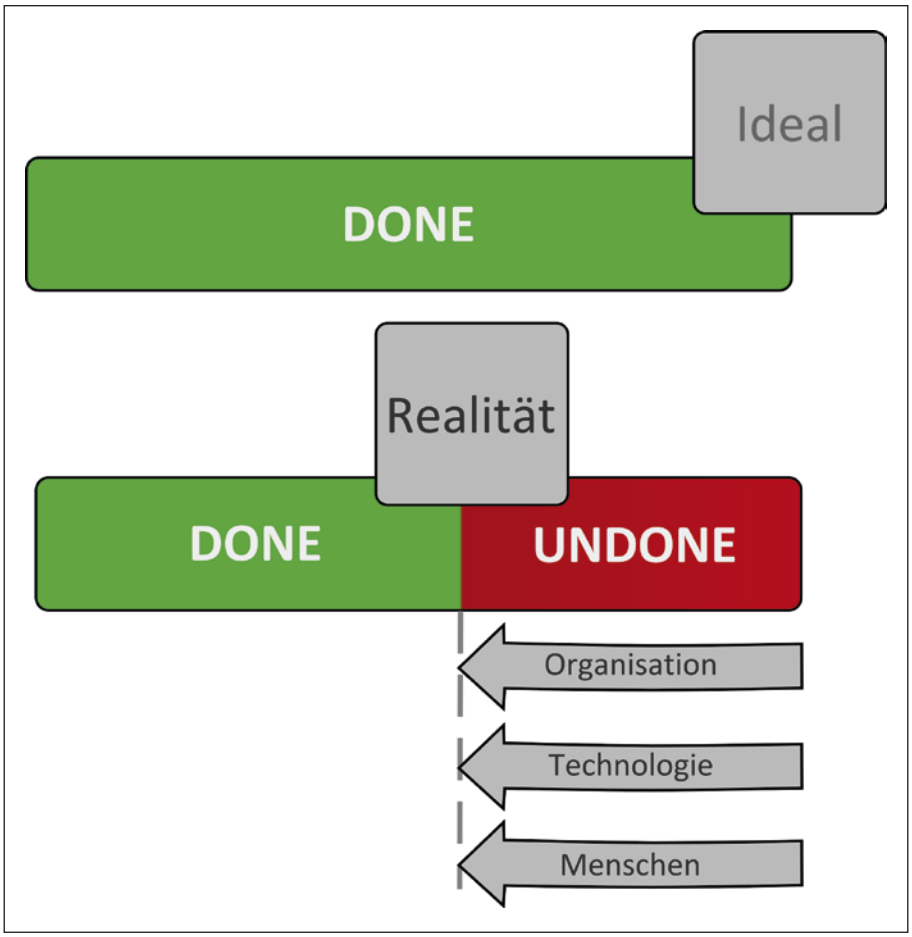

Abb. 1: „Done" im Ideal, „Undone" in der Realität: Häufig werden Features als „done" bezeichnet, die es tatsächlich nicht sind. Welche das sind, ist abhängig von der Organisation, den Technologien und den Beteiligten

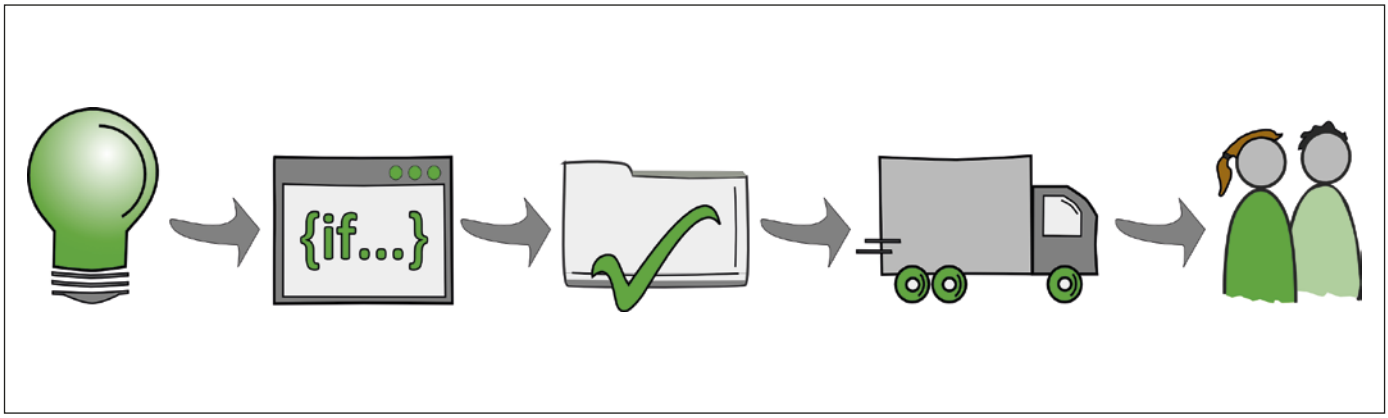

Abb. 2: Stationen der Wertschöpfungskette

tig“ zu deklarieren, der potenziell ausgeliefert werden kann. Das sollte dann in der Definition of Done festgehalten und damit transparent und klar sein.

Doch in der Realität passiert manchmal etwas anderes (**siehe Abbildung 1**). Die Unternehmen schaffen zwar eine veritable „Done“-Liste. Nur hat diese Liste einen dunklen Spiegel, den Katalog der „Undones“.

Scheinbar ist er nach der Umsetzung eines Features leer, aber eben nur scheinbar. Man glaubt lediglich, fertig zu sein. „Undones“ – Einträge, die im Sinne der „Definition of Done“ eben nicht „done“ sind – wurden zum Futter für das „Undone“-Monster. Es ist das Gespenst, das auch in IT-Projekten umgeht: Viel zu spät zu erfahren, was erfolgsentscheidend gewesen wäre. Oder dass der Markt das Produkt generell nicht will.

Schnittstellen der Wertschöpfungskette

Woher kommen diese „Undones“? Sie können zweierlei sein: Versäumnisse, die sich erst beim Wunsch nach Änderungen bemerkbar machen, beispielsweise technische Schulden. Oder fehlende Rückmeldungen über mindestens zwei der Stationen der Wertschöpfungskette hinweg. Die Kette Business – Entwicklung – Test – Operations – Kunde verfügt häufig über vier Schnittstellen (**siehe Abbildung 2**).

Damit hat nicht nur jede einzelne Station ihre eigenen, abzuleistenden Anforderungen, zusätzliche Komplexität resultiert aus dem Zusammenspiel der Stationen. Prompt kann es sein, dass sich bei jeder Station innerhalb der Kette „Undones“ ansammeln – nicht präzisierte Anforderungen, Code Smells, fehlende Tests usw. Teilweise standen diese Punkte vielleicht nie auf der Liste dessen, was für „Done“ zu erfüllen sei, wären aber wichtig gewesen. Häufig geht auch unter, dass sie nicht erledigt sind, weil das erst in den Folgestationen der Kette überprüft werden könnte und es an Feedback mangelt.

Die Wertschöpfungskette kann offensichtlich Lücken haben, was den Austausch zwischen den Stationen anbelangt. Allgemeiner Konsens dürfte darüber herrschen, dass Agilität die Lücke schließt zwischen Business und Entwicklung beziehungsweise Development. Agile Requirements Engineering definiert die Anforderungen so, dass der Fachbereich das entstehende Produkt für erstrebenswert hält. Scrum-Praktiken wie Refinement oder Review schaffen Transparenz im direkten Gespräch. Die agilen Prinzipien fordern, dass Fachexperten und Entwickler während des Projekts täglich zusammenarbeiten. Darüber hinaus tragen Just-in-Time Planning, die Untergliederung in einzelne Sprints und das iterative Vorgehen dazu bei, die Komplexität handhabbar zu machen.

Zusammengefasst ermöglicht Agilität das „Business Alignment“, also die Ziele von Fachbereich und Entwicklung auf eine gemeinsame Linie zu bringen. Das reduziert bereits das Risiko, ein aus Sicht des Fachbereichs falsches Produkt zu erstellen. Es sagt allerdings noch nichts über die Qualität des Produkts aus und erlaubt ebenfalls noch keine Aussage hinsichtlich der folgenden Stationen.

Innerhalb der Entwicklung wird auch durch die Prinzipien des Agile Software Engineering ein hoher Grad der Professionalisierung angestrebt. Methoden aus dem eXtreme Programming wie Pair Programming sollen die Zusammenarbeit verbessern und die innere Qualität der Software sichern. Das reduziert die Zahl der „Undones“ innerhalb der Entwicklung.

Agilität hat auch die Lücke zwischen Entwickeln und Testen weitestgehend geschlossen. Agile Software Engineering sieht eine möglichst hohe Abdeckung mit automatisierten Tests vor. Testen ist eine der Aufgaben des Teams, Unittests beispielsweise liefern praktisch sofort Feedback. Regressionstests, Performanztests, Exploratory Testing – sie alle schaffen Transparenz hinsichtlich der inneren und der äußeren Softwarequalität.

Die Entwicklung zu professionalisieren sowie Testen und Entwickeln zu verzahnen, reduziert weitere Risiken. Beispielsweise sinkt die Gefahr, irgendwann eine Codebasis zu haben, die sich kaum noch warten oder erweitern lässt. Continuous Integration senkt Risiken weiter, weil die Entwickler ständig den aktuellen Stand einchecken und damit nur kleine Änderungen machen. Das automatisierte Buildsystem gibt dann sehr schnell eine Rückmeldung zur gerade gemachten Änderung. Das Ziel besteht in einer höheren Codequalität und darin, die Software robust zu halten.

Schnitt- oder Sollbruchstelle? Die Auslieferung an Operations

Werden alle agilen Prinzipien konsequent umgesetzt, dann wirken sie wie eine Abmagerungskur für das „Undone“-Monster. Nur befinden wir uns jetzt erst am Ende der dritten Station der Wertschöpfungskette. Und es ist der Übergang zur nächsten Station, der häufig aus einer Schnitt- eine Sollbruchstelle werden lässt. Die Rede ist von der Auslieferung eines Releases an Operations, also den IT-Betrieb. Denn hier treffen zwei unterschiedliche Ambitionen aufeinander:

- Entwicklerteams möchten ihre Releases, Updates oder die vom Fachbereich sehnlichst erwarteten Funktionalitäten möglichst schnell den Kunden zur Verfügung stellen. Denn nur so bekommen sie das erwartete Feedback und erhalten die Option, Produkt und/oder

Prozess zu verbessern. Letztendlich bleibt das Team auch blockiert, bis die neue Lieferung wirklich abgenommen ist. Verständlich, dass ein Team diese Phase kurz halten möchte.

- Anders sieht es der IT-Betrieb: Jede Neuerung birgt potenzielle Risiken für die Sicherheit und Stabilität des Systems. Die Wartbarkeit muss überprüft, der zuverlässige Betrieb nachgewiesen werden. Das braucht Zeit und wird umso aufwendiger, je weniger Austausch es generell zwischen der Entwicklung und dem Betrieb gibt.

DevOps, die Verbindung von Entwicklung und Betrieb, soll die Lösung sein (vgl. [Kim16]). Allerdings gibt es unterschiedliche Auffassungen darüber, was „DevOps" ist: Eine Philosophie, Unternehmenskultur, vollständige Automatisierung, … Ernest Mueller [Muel10] definierte DevOps wie folgt: *„DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support."* („DevOps ist die Praxis, derzufolge IT-Betrieb und Entwickler am gesamten Service-Lebenszyklus gemeinsam teilhaben, beginnend beim Design über den Entwicklungsprozess bis zur Betriebsunterstützung", Übersetzung der Autoren.)
Weniger abstrakt ausgedrückt soll DevOps durch weitgehende Automatisierung und direktes Feedback sicherstellen, dass sich nirgends „Undones" ansammeln. Wichtig ist dafür die Art der Zusammenarbeit, die kontinuierliche Lieferung (Continuous Delivery), der intensive Austausch von Informationen und möglichst objektive Messung definierter Parameter. Alle Punkte der Produktentwicklung, auch bezüglich der Auslieferung, müssen fertig abgeschlossen sein.

Was ist neu am neuen Ansatz?

Ist dieser Ansatz neu? Eigentlich nicht. Bereits das agile Manifest fordert, regelmäßig (bei Scrum in jedem Sprint) ein lieferfähiges Inkrement zu haben. Also ein Inkrement, das die DoD, die „Definition of Done", erfüllt. Das impliziert, dass alle „Undones" inzwischen zu „Dones" wurden. Das gilt unabhängig davon, ob die „Undones" innerhalb einer Station angesammelt wurden oder aufgrund von mangelndem Feedback dazwischen, am Ende sollte nichts mehr zu tun übrig sein („nothing left to do").
Nur ist das, wie beschrieben, in realen Projekten manchmal nur bis zum Ende der Entwicklungsphase tatsächlich der Fall. Insofern liefert DevOps einen sehr wertvollen erneuten Fokus, indem es den Blick richtet auf die Lücke zwischen Entwicklung und Betrieb. Enge Zusammenarbeit, entsprechende Coachings und nützliche Tools tragen dazu bei, den intensiven Austausch zu fördern und damit Funktionsfähigkeit und Schnelligkeit zu vereinen. Dahinter steht die Überzeugung, dass sich negative Verhaltensweisen umso leichter einstellen, je größer der Abstand wird zwischen einer Handlung und ihren Konsequenzen. Zwei Aufgabenfelder zu verzahnen bedeutet, Tat und Resultat dicht zusammen zu bringen. Sollten Organisationen also am besten sofort mit DevOps starten?

Erfolgreich starten oder die Frage der Professionalisierung

So einfach ist es nicht. DevOps hilft, das Produkt der Entwickler schneller zum Kunden zu bringen. Wenn es der Fall sein sollte, dass, lapidar gesagt, die Entwicklung Schrott liefert, dann liefert diese Entwicklung mit DevOps Schrott schneller. Die Frage, die sich eine Organisation vor der Entscheidung für DevOps stellen sollte, ist die nach dem Stand der Professionalisierung der eigenen Entwicklung. Um es ganz deutlich zu sagen: DevOps ist kein Ersatz für eine professionelle Entwicklung. Die Praktiken des Agile Software Engineering sind elementar. Denn sonst besteht die Gefahr, dass Probleme beim Betrieb zurück zur Entwicklung wandern, die Entwicklung jedoch (noch) nicht in der Lage ist, damit umzugehen.
Eine maßgebliche Rolle spielt die automatisierte Testabdeckung. Sie liefert das Sicherheitsnetz, das den Entwicklern erlaubt, den Code zu verändern, ohne sich unerwünschte Seiteneffekte einzuhandeln. Andernfalls werden die Entwickler eher entmutigt, Code einzuchecken.
Stabile und hundertprozentig automatisierte Tests – besonders auf der Ebene der Systemintegration – sind erst mit Operations möglich. Es ist also durchaus wertvoll, DevOps-Themen aufzugreifen. Wie können Organisationen nun konkret vorgehen, die Agilität und DevOps vereint einsetzen wollen?
Sie können in kleinen Schritten starten und zuerst die Ziele definieren. Das bedeutet, als erstes die „Undones" zu sammeln und transparent zu machen. Dinge, die auf einer „Definition of Done" beziehungsweise auf der „Undone"-Liste stehen können, sind beispielsweise:

- Feature umgesetzt,
- Akzeptanzkriterien erfüllt,
- Stakeholder-Feedback eingeholt,
- Code erfolgreich integriert,
- Code-Review erfolgt,
- Unittest-Abdeckung > XX Prozent,
- Coding-Standards erfüllt (keine Warnings),
- Tests erfolgreich durchgelaufen,
- Oberflächentests erstellt,
- Systemtests erstellt,
- Integrationstests erstellt,
- Explorative Tests erfolgt,
- automatisierte Akzeptanztests erstellt,
- Performanztests erfolgt,
- in Produktion,
- Monitoring eingerichtet,
- Chaos Monkey getestet,
- Nutzungswerte erhoben,
- Kundenfeedback erhalten,
- Verkaufszahlen erhoben.

Liefert Ansatzpunkte: Die „Undone"-Analyse

Was wird im Verlauf der bereits etablierten Prozesse zu „Dones", was nicht? An welchen Stellen treten „Undones" auf? Beispielsweise könnte die Analyse zeigen, dass die Performanztests nicht am Ende des Sprints erledigt sind, sondern erst nachgelagert in einer zusätzlichen Phase. Oder dass es an einem Code-Review mangelt. Aus den „Undones" ergeben

sich sofort mögliche Ansatzpunkte zur kontinuierlichen Verbesserung, um möglichst schnell Feedback zu erhalten und das Risiko zu minimieren.

Die Liste erscheint auf den ersten Blick recht umfangreich zu sein, sie bildet aber nur transparent ab, was Entwicklungs- und Betriebsteams heute alles leisten müssen. Um trotz der langen Liste schnell und agil zu bleiben, setzten viele Teams auf bedingungslose Automatisierung der Auslieferung des Inkrements.

Wichtig ist, von Anfang an alle Betroffenen einzubeziehen und die Unterstützung des Managements einzuholen. Das ist umso wichtiger, als sowohl die Maßnahmen zur Professionalisierung als auch die einer möglichen Umstrukturierung erhebliche Investitionen erfordern können. Letztere sollten sich indessen in einem überschaubaren Zeitrahmen amortisieren. Genau dieses Vorgehen, konsequente Transparenz über „Undones“, kann als Hilfsmittel für Teams und Scrum Master dienen, um eine kontinuierliche Verbesserung zu ermöglichen.

Als Beispiel sei ein Team genannt, das aus dem Punkt „Performanztests erfolgt“ auf der „Undone“-Liste einen Punkt auf der „Done“-Liste gemacht hat. Das Team hat also transparent werden lassen, welche Risiken und Kosten entstehen oder zumindest entstehen können, wenn diese Tests erst nachgelagert ausgeführt werden. Um Sicherheit statt Risiko zu schaffen, hat das Team begonnen, innerhalb jedes Sprints automatisierte Performanztests abzuschließen. Das geht nicht von heute auf morgen und ist mit viel Aufwand verbunden. Dennoch lohnt sich das Vorgehen, weil es die nachfolgenden – und dank der Transparenz offen sichtbar gewordenen – Risiken minimiert.

Aus „Undones“ konsequent „Dones“ zu machen, ermöglicht kontinuierliches, frühes Feedback, vom Beginn der Entwicklung an. Damit schließt sich die Wertschöpfungskette zu einem Kreis, der Gedanke der Business Agility wird konsequent umgesetzt bis zu den Kunden. Das bedeutet, schnell Wert zu schaffen, schnell aus den Rückmeldungen zu lernen und schnell auf den Markt zu reagieren. Kurz gesagt, es geht um Innovationsfähigkeit. ||

Literatur & Links

[Kim16] G. Kim, J. Willis, P. Debois, J. Humble, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, It Revolution Press, 2016

[Lar08] C. Larman, B. Vodde, Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum, Addison-Wesley Professional, 2008

[Muel10] E. Müller, What is DevOps? Blog-Beitrag, 2010, siehe **https://theagileadmin.com/what-is-devops/**

[Scha16] F. Schad, B. Seidler, Don't feed the Undone-Monster, Video des Vortrags auf der Tools4AgileTeams, 2016, siehe: **https://www.youtube.com/watch?v=F_laNmppJ4U**

[Schw12] K. Schwaber, J. Sutherland, Software in 30 Tagen: Wie Manager mit Scrum Wettbewerbsvorteile für ihr Unternehmen schaffen, dpunkt.verlag, 2013

Die Autoren

Benjamin Seidler
(benjamin.seidler@andrena.de) ist Agile Coach bei der andrena objects ag und verfügt über ca. zehn Jahre Erfahrung als Software-Engineer, Product Owner, Scrum Master und Coach. Er unterstützt Unternehmen beim Etablieren und Optimieren agiler Prozesse und hält oft Trainings und Vorträge zu Agilität und Scrum.

Felix Schad
(felix.schad@andrena.de)
ist Scrum Master und Agiler Coach bei der andrena objects ag und unterstützt Unternehmen bei der Umsetzung agiler Projekte. Er verfügt über mehrjährige Erfahrung als Softwareentwickler mit starkem Background in Test-Driven Development (TDD) und Clean Code.