HANSER

Leseprobe

zu

SAP, The Agile Way

von Klaus Wybranietz

Print-ISBN: 978-3-446-46567-1
E-Book-ISBN: 978-3-446-46586-2
E-Pub-ISBN: 978-3-446-46706-4

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-46567-1
sowie im Buchhandel

Inhalt

Vorwort

In den vergangenen Jahren habe ich von vielen Führungskräften auf der ganzen Welt gehört, wie toll ihre agilen Entwicklungsteams zum Beispiel im Java-Umfeld arbeiten. Ich habe gehört, wie glücklich und effektiv Softwareentwickler sind, die mit Scrum arbeiten, wie viel Power und positive Energie in diesen Teams entsteht. Und: Mir wurde auch erzählt, wie viele Bewerbungen von Top-Spezialisten diese Unternehmen bekommen, nachdem sie sich zum agilen Arbeiten bekannt oder sogar eine agile Transformation angestoßen haben.

Wenn ich mit Personen aus dem SAP-Umfeld spreche, höre ich genau das Gegenteil. Ich verstehe SAP-Projekte ganz einfach immer so: Der Kunde kauft eine Standard-Software, die nur angepasst werden muss und danach genutzt werden kann. Es wird also keine neue Software entwickelt, sondern gekaufte Software adaptiert. Das ist der zentrale Unterschied zu allen anderen individuellen Entwicklungsprojekten dieser Welt.

Trotzdem habe ich zu viele SAP-Projekte gesehen, die agil gescheitert sind, und ich habe zu oft gehört, dass SAP und Scrum einfach nicht kompatibel seien. Meistens lag und liegt die Ursache des Scheiterns darin, dass die Anforderungen nicht konsequent und kontinuierlich genug priorisiert werden. Ganz im Sinne des Wasserfalls werden umfangreiche Anforderungen definiert, verbunden mit der Erwartung, dass sie auch zu 100 Prozent umgesetzt werden. Das hat nichts mit agilem Arbeiten zu tun und ist zum Scheitern verurteilt. Dies hat einige Projektverantwortliche dennoch nicht daran gehindert, ihren Wasserfall-Projekten eine agile Masche umzubinden, indem sie einfach die Events „agil" benannt haben. Die gängigsten Ausreden, die hartnäckigsten „Ja, aber …", finden Sie gleich in Kapitel 1. Der Grundtenor lautet: SAP ist angeblich viel zu komplex und viel zu integriert, als dass sich da agil was machen ließe.

Das mag in der Welt dieser SAP-Profis so sein. Doch auch an ihnen geht die Agilisierung der Arbeitswelt nicht spurlos vorbei, und wenn sie noch nicht mitten in einer agilen Transformation stecken, stehen die Chancen gut, dass sie in den kommenden Jahren davon betroffen sein werden. Spätestens dann werden sie – sehr schnell – lernen müssen, mit agilen Organisationseinheiten und vielleicht sogar mit ihren SAP-Teams agil zu arbeiten.

Ein solcher Anlassfall könnte auch S/4HANA sein. Mit „SAP Activate" forciert SAP die schnelle, agile Einführung anhand vorgedachter Konfigurationen (Guided Configurations). Aus meiner Sicht unterstützt SAP Activate alle Ausgangspunkte für den agilen Projektstart, von der Neuimplementierung von S/4HANA über die Systemkonvertierung bis hin zur Landschaftstransformation. Beim empfohlenen Framework setzt SAP auf Scrum und Scrum@Scale – beides werde ich in diesem Buch besprechen.

Ich brauche aber nicht erst S/4HANA, um zu wissen, dass SAP und Agile gut zusammenpassen. Der Erfolg bei internationalen Großkonzernen, wo wir nicht nur agil, sondern mit

weltweit verteilten Teams agil arbeiten, gibt mir in dieser Hinsicht recht. Für mich war es deshalb an der Zeit, die „guten agilen Praktiken“, die ich in den letzten 14 Jahren (von 25 im SAP-Umfeld insgesamt) mit vielen Teams erarbeitet habe, komprimiert weiterzugeben. Mir geht es immer darum, die Dinge möglichst einfach zu machen. Daher werden Sie in diesem Buch keine philosophischen Vertiefungen in das Thema Agilität finden, sondern praktisch Erprobtes und ständig Verbessertes. Für sämtliche Aspekte von Scrum, Kanban und Agile im Allgemeinen gibt es inzwischen eine breite Palette an anderen Publikationen, die ich Ihnen ans Herz legen möchte, damit Sie das agile Arbeiten von Grund auf verstehen. Ich halte mich in meiner agilen SAP-Praxis sehr eng an den Scrum Guide von Jeff Sutherland und Ken Schwaber, alle Fragen der Skalierung haben den Guide zu Scrum@Scale als Basis.

Ein paar Dinge müssen für das SAP-Umfeld aber angepasst werden, um erfolgreich agil arbeiten zu können – diese Praktiken beschreibe ich in diesem Buch.

Klaus Wybranietz

München, September 2020

Der Autor

Klaus Wybranietz war zwölf Jahre als SAP-Berater auf der ganzen Welt unterwegs, bevor er 2001 die antagon AG gründete. Nach der Fusion der antagon AG mit der EXXETA GmbH im Jahr 2006 wurde das SAP-Team rund um Klaus Wybranietz aufgrund der herausragenden Erfolge in agilen Projekten mit höchstem technischen Anspruch zu einem SAP Service Partner. Zu den Spezialgebieten des SAP ERP-Experten zählten Verkauf, Absatzplanungsprozesse, Verkaufssteuern und Sales Taxes (USA) sowie Rechnungslegung mit Sales Reporting.

Seit 2012 widmet sich Klaus Wybranietz ausschließlich der Verbindung von Agilität und SAP-Projekten. Als Agile Coach und Trainer begleitet er Organisationen und Führungskräfte in agilen Transformationen mithilfe von Scrum, Scrum@Scale, LeSS, Kanban, Design Thinking und den Management-3.0-Praktiken nach Jurgen Appelo. Als einer der wenigen auf SAP-Projekte spezialisierten Agile Coaches und Projektleiter arbeitet Klaus Wybranietz regelmäßig mit Teams, die über Europa, die USA und Südafrika verteilt sind.

4 Mit agilen SAP-Teams starten

Auch wenn ich umgehend wieder das Argument „Aber wir entwickeln doch gar keine Software!“ zu hören bekomme, mache ich als Agile Coach und Berater mit jedem neuen Team das Gleiche: Ich werfe mit den Teammitgliedern gleich zu Beginn einen genauen Blick auf das Manifest für Agile Softwareentwicklung:

> *Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:*
>
> | | | |
> |---|---|---|
> | *Individuen und Interaktionen* | *mehr als* | *Prozesse und Werkzeuge* |
> | *Funktionierende Software* | *mehr als* | *umfassende Dokumentation* |
> | *Zusammenarbeit mit dem Kunden* | *mehr als* | *Vertragsverhandlungen* |
> | *Reagieren auf Veränderung* | *mehr als* | *das Befolgen eines Plans* |
>
> *Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.*
>
> *agilemanifesto.org*

Das Agile Manifest wird zwar sehr oft zitiert, allerdings fallen dabei oft die zwölf Prinzipien unter den Tisch, die hinter diesem Manifest stehen. Man kann in diesen zwölf Prinzipien das Wort „Software“ durch was auch immer ersetzen – die Aussagen haben universelle Gültigkeit:

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufriedenzustellen.
2. Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen als Wettbewerbsvorteil für den Kunden!
3. Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projekts täglich zusammenarbeiten.
5. Setze Projekte rund um motivierte Individuen auf. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode, Information an ein und innerhalb eines Entwicklungsteams zu übermitteln, ist das Gespräch von Angesicht zu Angesicht.
7. Funktionierende Software ist das wichtigste Maß für den Fortschritt.
8. Agile Prozesse fördern eine nachhaltige Entwicklung. Die Sponsoren, Entwickler und User sollten in der Lage sein, eine konstante Geschwindigkeit unendlich aufrecht zu erhalten.

9. Das ständige Augenmerk auf technische Exzellenz und gutes Design fördert die Agilität.
10. Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Wie ich schon in Kapitel 1 erwähnt habe, wurde der Fokus von Scrum im Scrum Guide schon längst erweitert. Es wird ausdrücklich betont, dass Scrum in allen möglichen Kontexten eingesetzt werden kann. Daher nehme ich das „Aber …“ zwar zur Kenntnis, aber mit Scrum kann man definitiv arbeiten, wenn eine Standardsoftware angepasst werden soll. Es hält der Einführung von Prozessen im Sinne einer erweiterungsfähigen Software stand, deren Release-Fähigkeit wir durch modifikationsfreie Anpassungen und Erweiterungen erreichen wollen.

Ja, das Agile Manifest gilt mit allen seinen zwölf Prinzipien für jede Art von agilen SAP-Projekten! Deshalb ist es meine Grundlage für die Arbeit mit agilen SAP-Teams.

4.1 Wir brauchen eine Vision

Mein inzwischen emeritierter BWL-Professor sagte in seinen Vorlesungen noch vor Helmut Schmidt: „Wer Visionen hat, sollte zum Arzt gehen.“ Auch in einem meiner Projekte stellte mein Gegenüber Folgendes fest: „Produktvisionen oder Vision Statements für das neue SAP-Modul, für die S/4-HANA-Implementierung, für das neue SAP-Projekt – das ist ganz oft eine Fehlanzeige. Man hat keine Zeit und ist zu beschäftigt, um über eine Vision nachzudenken. ‚Eine Vision erstellen‘ klingt immer irgendwie nach Weichei und Zeitverschwendung, das brauchen wir nicht.“

Aber wahrscheinlich würden Sie in Ihrem SAP-Projekt gerne den folgenden Zustand sehen:

- Die Teammitglieder wissen, warum sie die Dinge tun.
- Die Teammitglieder wissen, welche Ziele sie haben.
- Die Teammitglieder verstehen, warum welche Prioritäten gesetzt werden.
- Die Teammitglieder wissen, was ihr Produkt auszeichnet.

Im Scrum Guide 2020 wird zwar nur mehr vom „Produkt-Ziel“ gesprochen, aber ich möchte hier gerne beim Begriff der Vision bleiben. Visionen sind weder Utopien noch Halluzinationen. Im agilen Kontext sind sie ein Idealbild, die Wunschvorstellung des Product Owners für sein Produkt. Der Product Owner ist nämlich derjenige, der für den Erfolg eines Produkts oder Projekts verantwortlich ist, denn laut Scrum Guide hat er die Aufgabe, den Wert des Produkts zu maximieren. Deshalb wird er die Einträge im Product Backlog auch so sortieren, dass er sein Ziel, seine Mission – seine Vision – erreichen kann. Also ist es richtig und wichtig, dass sich ein Product Owner eine Vision für sein SAP-Projekt entwickelt.

Was ist eine Produktvision?

- Die Produktvision beschreibt den Zweck eines Produkts.
- Die Produktvision reflektiert die Absicht, mit der das Produkt erschaffen wird.
- Die Produktvision zeigt, was die Kunden, die direkte Zielgruppe und die Endanwender mit diesem Produkt erreichen sollen.

Wir sprechen bei einer Vision also über den zukünftigen Zustand des Produkts, eines Prozesses oder eines SAP-Moduls und darüber, welche Probleme damit gelöst werden. Das ist jetzt nichts Verwerfliches und hat auch nichts von Weichei, oder?

Ganz im Gegenteil: Eine glasklare und inspirierende Produktvision hilft dabei, unsere Entwicklungsteams, unsere Stakeholder, unsere Kunden und Anwender zu motivieren – und das ist auch der Unterschied zu einem bloßen Ziel. Sie liefert dem Scrum-Team ein gemeinsames Verständnis der Richtung, in die es sich bewegen soll. Dem Entwicklungsteam hilft sie bei technischen Überlegungen und die Stakeholder erkennen, warum welche Priorität gesetzt wurde und was welchen Wertbeitrag liefert. Für die Anwender ist eine Vision spürbar, weil sie intuitiv verstehen, warum und wie sie das Produkt einsetzen sollen. Darüber hinaus unterstützt die Vision den Product Owner bei allen Entscheidungen, die er jeden Tag treffen muss, denn er hat damit etwas in der Hand, an dem er sich orientieren kann. Es ist daher einfach unglaublich wichtig, eine Produktvision zu haben.

Wichtig ist, dass sich ein Product Owner dieser wichtigen Aufgabe nicht entzieht. Er ist die Quelle und der Treiber der Vision, er wird sie leidenschaftlich vertreten und verbreiten. Wenn er seine Vision vorstellt, muss sein Herz schneller schlagen und das seiner Zuhörer natürlich auch. Ich meine das nicht pathetisch, sondern ernst und genau so: Das Herz muss schneller schlagen – messen Sie doch einfach mal den Puls, wenn Sie Ihre Vision vorstellen! Selbst bei einer SAP-Erweiterung und der Einführung einer Standardsoftware wie S/4 HANA oder SAP ERP geht ohne die Vision gar nichts.

Wer als Product Owner nur eine vorgegebene Roadmap nachplappert, verschenkt eine wichtige Möglichkeit zur Motivation.

Die Vision formulieren

Im Mittelpunkt der Vision sollte immer der Nutzen für den Anwender oder Kunden stehen. Alle technischen Details können Sie bei der Formulierung weglassen. Gehen Sie auf jeden Fall mit der Einstellung an die Formulierung, dass eine Vision niemals in Stein gemeißelt ist. Die Umstände können sich ändern, das hat uns die COVID-19-Pandemie eindrücklich gezeigt. Haben Sie als Product Owner deshalb keine Angst, die Vision anzupassen – die erfolgreichsten Unternehmer dieser Welt machen das ja auch.

Die beste Erfahrung habe ich damit gemacht, Produktvisionen so weit zu schärfen, dass ich sie in einem Elevator Pitch unterbringen kann. Dabei handelt es sich um eine informative und einprägsame Präsentation der Idee für ein Produkt, die sich in maximal drei Minuten unterbringen lässt – besser noch in maximal 120 oder 60 Sekunden, damit man sie dem Vorstand im Aufzug innerhalb von drei Stockwerken vermitteln kann. Gelernt habe ich die Kunst des Elevator Pitches direkt beim Erfinder, nämlich bei Joachim Skambraks.

In einem Kreis von 150 Zuhörerinnen und Zuhörern haben wir geübt, die Leistungen und die Leistungsfähigkeit unserer Unternehmen auf den Punkt zu bringen und zu vermitteln. Nach Skambraks hat ein guter Elevator Pitch drei zentrale Elemente:

1. **Zielgruppe:** An wen genau wollen wir verkaufen bzw. für wen wird das SAP-Modul angepasst und implementiert? Wer sind die Endkunden, Zielkunden, Märkte?
2. **Produkt:** Was zeichnet unser Produkt aus?
3. **Bedürfnisse der Zielgruppe:** Welche Vorteile hat die Zielgruppe durch unser Produkt? Ich frage immer drei Mal: „Was bringt's? Was bringt's? Was bringt's?"

Da wir im SAP-Umfeld fast immer für interne Kunden und Endkunden arbeiten oder länderspezifische Prozesse einführen, füge ich dem Ganzen gerne noch einen vierten Punkt hinzu (siehe Bild 4.1):

4. **Ziele:** Welche geschäftlichen Ziele werden durch die SAP-Einführung verfolgt?

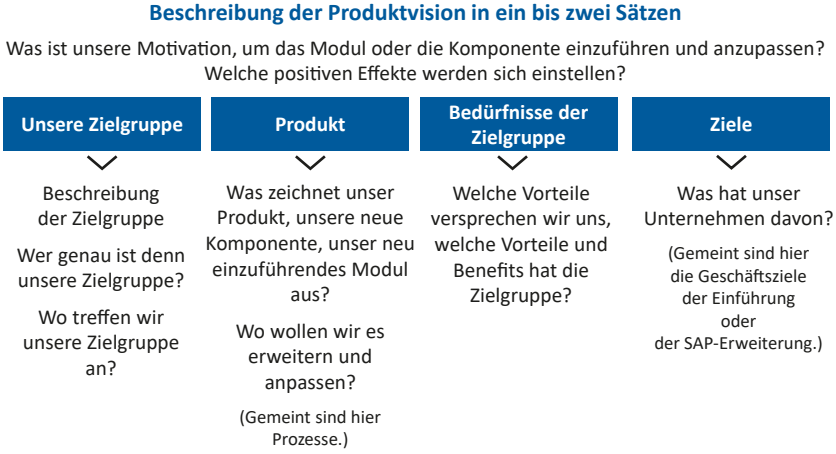


Bild 4.1 Fragen für die Formulierung des Elevator Pitch

Als Formulierungshilfe können Sie sich an der guten AIDA-Formel aus der Werbung orientieren:

- **Attention:** Aufmerksamkeit erregen (z. B. mit einem ersten Satz, der irritiert oder gleich das Bedürfnis auf den Punkt bringt).
- **Interest:** Interesse für das Produkt wecken, damit das Gegenüber mehr darüber wissen will.
- **Desire:** Verlangen nach diesem Produkt erzeugen, indem deutlich gemacht wird, wie das Produkt dem Bedürfnis des Gegenübers entgegenkommt.
- **Action:** Dem Gegenüber wird gezeigt, wie man an dieses Produkt kommt – es kann sofort eine Handlung („Ich will das live sehen!") setzen.

Die AIDA-Formel ist natürlich idealtypisch und zeigt eine lineare Reiz-Reaktion-Situation, aber sie ist ein guter Anhaltspunkt, um wichtige „Trigger" im Elevator Pitch unterzubringen. Erweiterungen dieser Formel fügen am Schluss zum Beispiel noch ein „S" für „Satisfaction" (Zufriedenheit) ein: Auch der Kaufprozess bzw. der Prozess der SAP-Einführung muss so gut verlaufen, dass der Kunde immer wieder auf diese Leistung zurückgreifen würde.

Wenn Sie die Fragen beantwortet haben, können Sie daraus mehrere Varianten der Produktvision formulieren und jeweils das Wichtigste in den Vordergrund stellen:

- Eine Kurzversion für die Fahrt im Aufzug mit dem Vorstand
- Eine Version für die Stakeholder
- Eine Version für die Entwicklungsteams
- Eine Version für Endkunden und/oder Endanwender
- Eine Version in PowerPoint
- Eine Langversion als Arbeitsgrundlage für Sie selbst oder für Präsentation, die einen weiteren zeitlichen Rahmen bieten

4.2 Wir brauchen ein Team

Der Kern von Scrum ist der Teamansatz: Das Entwicklungsteam ist eine kleine Gruppe von Spezialistinnen und Spezialisten, die flexibel und anpassungsfähig sind und selbstorganisiert arbeiten. Diese Fähigkeit zur Zusammenarbeit über die Grenzen der eigenen Profession hinaus wirkt auch zwischen mehreren Entwicklungsteams, wenn zum Beispiel ein größeres Projekt umgesetzt werden soll.

Das passt nun so gar nicht mit meinen Erkenntnissen zusammen, die ich über SAP-Berater und -Entwickler – kurz „SAP-Profis" – gewonnen habe. Meistens sind das hochspezialisierte Einzelkämpfer, eben die in Kapitel 1 erwähnten Superstars, die schon viele Herausforderungen gemeistert haben. Diese Superstars stecken wir nun in ein kleines, selbstorganisiertes Team mit drei bis neun Mitgliedern und erwarten Teamfähigkeit. Vor allem erwarten wir, dass sie ihre Erfolge mit dem Team teilen, Wissen weitergeben und den anderen Mitgliedern helfen. Richtig: Das passiert nicht einfach so – dafür müssen wir etwas tun!

Allerdings werden Sie nicht viel erreichen, wenn Sie mit diesen Superstars in regelmäßigen Abständen fürs Teambuilding auf Berge steigen, Team-Karaoke-Abende oder Kochevents veranstalten. Da sich das Homeoffice durch COVID-19 als langfristig attraktive Alternative entpuppt hat und SAP-Teams ohnehin die Tendenz zur internationalen Verteilung haben, müssen wir es auch noch schaffen, die Superstars online zu integrieren. Ich nutze dafür die „Personal Map", eine Idee von Jurgen Appelo.

Personal Maps

Die meiste Zeit arbeiten wir in unseren Jobs mehr nebeneinander statt miteinander, die Kolleginnen und Kollegen kennen sich gar nicht wirklich. Um jemand anderen zu verstehen, sollte man aber etwas über seine Lebensgeschichte wissen, über seine Ziele, Werte und Beweggründe. Bevor ich mit einem neuen Team starte, mache ich daher gerne ein 45-minütiges Mini-Teambuilding mit Hilfe der Personal Map.

Die Personal Map selbst ist im Grunde eine Abwandlung der Mind Map. Rund um einen zentralen Begriff (in diesem Fall eine Person) werden die Gedanken zu Hauptthemen aufgeschrieben, die bei der Personal Map auch vorgegeben sein können (siehe Bild 4.2).

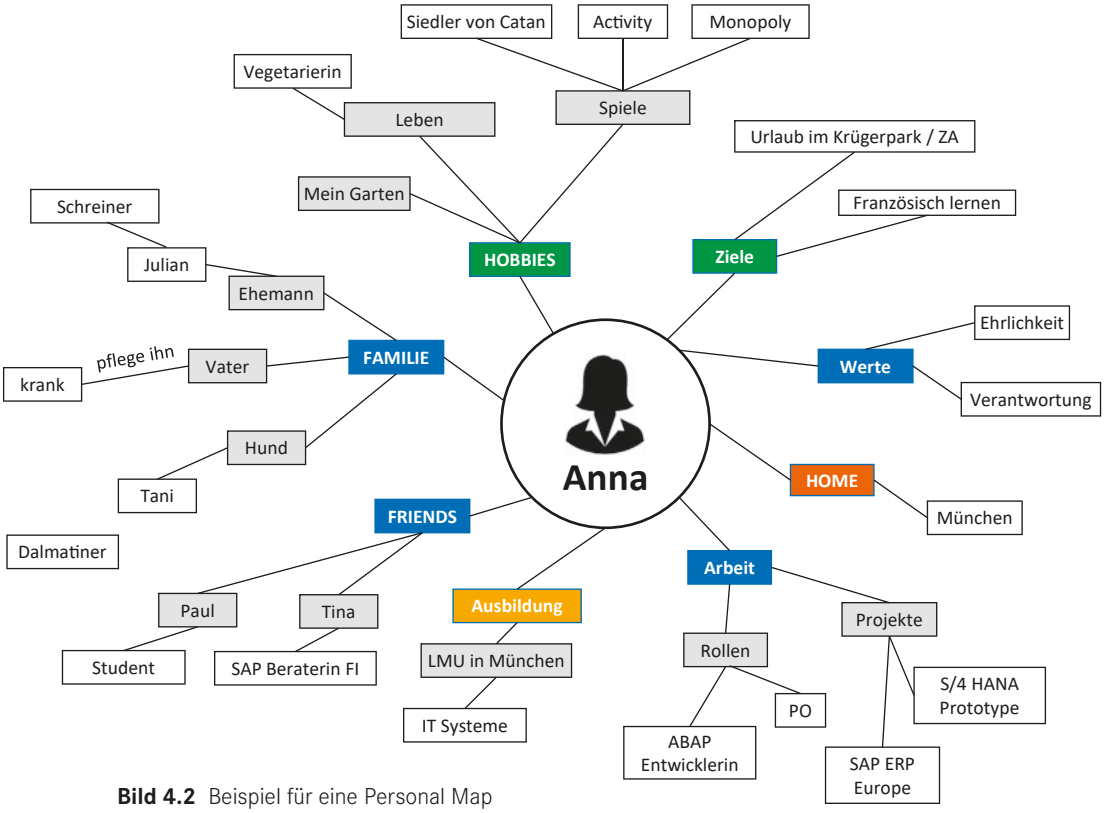


Bild 4.2 Beispiel für eine Personal Map

Der Ablauf

1. Vorstellung der Personal-Map-Idee (5 Minuten)
2. Personal Map anfertigen (10 Minuten)
3. Einen Präsentationspartner finden (2 Minuten)
4. Dem Präsentationspartner die eigene Personal Map erklären (2 Minuten)
5. Die Personal Map des Präsentationspartners wird dem Team präsentiert (2 Minuten pro Person)

Die Regeln

Starte mit deinem Namen in der Mitte des Blatts und erzähle uns etwas über

- deine Ausbildung
- deinen bisherigen Werdegang
- deine Arbeit bis heute
- deine Familie (optional)
- dein Auto (optional)
- dein Motorrad (optional)
- deine Hobbys (optional)
- deine Ziele
- deine Werte

Wie moderieren Sie dieses Mini-Teambuilding?

Natürlich ernte ich zuerst einmal Widerstand, wenn ich die Personal Map vorstelle. Die Argumente sind fast immer die gleichen:

- Wir kennen uns seit Jahren!
- Wir arbeiten doch jeden Tag zusammen!
- Das ist Zeitverschwendung!

Lassen Sie sich von solchen Einwürfen nicht ablenken. Da müssen Sie durch!

Gerade wenn sich Kolleginnen und Kollegen seit vielen Jahren kennen, gibt es immer wieder echte Aha-Erlebnisse:

- Was, du gehst auch gerne wandern?
- Ich wusste gar nicht, dass du töpferst.
- Cool, ich habe auch einen BMW-Oldtimer!

Wenn manche Kolleginnen und Kollegen ihr Privatleben nicht ausbreiten wollen, dann respektieren wir das, daher sind diese Fragen optional. Manchmal kann es schon schwierig sein, offenzulegen, welches Auto oder Motorrad jemand fährt. Wenn man für Mercedes-Benz arbeitet und einen BMW-Oldtimer fährt – naja. Achten Sie also darauf, mit solchen Themen sensibel umzugehen.

Natürlich könnte jedes Teammitglied seine eigene Personal Map präsentieren, aber ich wähle bewusst die Variante, die Map eines Partners vorzustellen. Das hält die Egotrips einiger Kollegen im Zaum und hilft, die Timebox konsequent umzusetzen.

Die Personal Map wird übrigens nicht einmal angefertigt und dann wieder vergessen. Es sind lebendige Artefakte, die im Teamraum oder im Büro des Scrum Masters, Line Managers oder Product Owners hängen und immer wieder ergänzt werden sollten. Vor allem, wenn es einen Grund zum Feiern gibt, zum Beispiel Geburten, Hochzeiten, ein neues Auto oder wenn S/4 HANA auf der Integrationsumgebung läuft.

Wir haben was zu feiern!

Machen Sie aus den Ergänzungen der Personal Maps ruhig ein Ritual.

- Erster Akt: Wir läuten die Teamglocke.
- Zweiter Akt: Die Personal Map wird ergänzt.
- Dritter Akt: Es gibt Kuchen, Weißwurst mit Brezen oder etwas Gesundes – je nach den Vorlieben des Teams.

Die Online-Version

Klar, am einfachsten funktioniert das Teambuilding mit der Personal Map, wenn sich alle Teammitglieder an einem Ort befinden und jedes Mitglied ein Blatt Papier zum Bemalen bekommt, das einfach an die Wand gehängt werden kann. Wenn ein Team aber über Südafrika, Frankreich, Rumänien und Deutschland verteilt ist, müssen wir auf die Personal Map trotzdem nicht verzichten.

Die Online-Version wird mit zwei PowerPoint-Folien anmoderiert, auf denen der Ablauf und die Regeln gezeigt werden. In der gemeinsamen Session werden auch die Präsentations-

partner gesucht. Danach kann jede und jeder die Personal Map erstellen, so wie sie oder er möchte: entweder mit PowerPoint oder einem anderen Online-Tool, gerne auch mit Papier und Buntstiften – einfach ein Foto machen und hochladen, fertig.

Für die 2x2 Minuten, um dem Partner oder der Partnerin die eigene Personal Map zu erklären, können Sie in Zoom entsprechend vorbereitete Breakout-Sessions anlegen, die nach vier Minuten automatisch enden und in die gemeinsame Session zurückführen. In Skype oder Teams müssen die Sessions vorher den Partnern zugewiesen werden.

Ob on- oder offline: Dieser kurze Teambuilding-Trip klappt immer wunderbar und regt viele positive Gespräche im Team an.

4.3 Wir brauchen Regeln

Ich bemerke in vielen Unternehmen die Unart, Meetings prinzipiell mit drei bis fünf Minuten Verspätung zu starten. Das gehört quasi zur Unternehmenskultur und wird akzeptiert. Die zweite Unart ist, unvorbereitet in ein Meeting zu gehen. Vielleicht waren Sie es auch viele Jahre so gewöhnt: Per Outlook wird zu einem Meeting eingeladen, man schaut kurz, wer der Einladende ist und wie der Titel des Meetings lautet, und sagt zu oder ab. Im Meeting erwartet man die Vorstellung einer Agenda und eine Einführung zum Thema. Viele von uns haben gar nie daran gedacht, dass wir selbst vorbereitet zum Meeting kommen sollten. Es wurde ja nie verlangt.

Ein agiles Team kann das nicht tolerieren. Für ein Scrum-Team im SAP-Umfeld, für die SAP-Berater und Entwickler, gelten ab sofort zwei Regeln:

1. Wir starten jedes agile Event auf die Minute pünktlich!
2. Wir kommen immer top vorbereitet zu den agilen Events!

Wenn es am Anfang zu Verspätungen kommt, wird der Scrum Master kurz, schmerzlos und wertschätzend erklären, warum absolute Pünktlichkeit ein Pfeiler der Agilität ist. Pünktlichkeit ist der Ausdruck eines der fünf Werte von Scrum: Respekt. Wer pünktlich ist, geht rücksichtsvoll mit der Zeit der anderen um und stellt die gemeinsame Arbeit über persönliche Befindlichkeiten. Sobald die Teammitglieder die agilen Werte verstanden haben und leben, kann der Scrum Master darauf vertrauen, dass sein selbstorganisiertes Team Verspätungen einfach nicht mehr toleriert und sie offen anspricht. Nicht nur gegenüber Teammitgliedern, sondern auch gegenüber Führungskräften und Stakeholdern, die zum Beispiel zu Reviews eingeladen werden.

Genauso gehört das antrainierte „Sich-berieseln-lassen“ in Meetings der Vergangenheit an. Alle agilen Events, inklusive Product Backlog Refinement, haben eine Agenda und einen definierten Input, der vorher adressiert wird. Als Mitglieder eines Scrum-Teams kommen wir vorbereitet zu den Events und erwarten das auch von den Gästen. Zum Beispiel sehen wir uns die Storys für das Refinement im Vorfeld an, wir verändern den Status am Scrum-Board oder bereiten uns mit einer Generalprobe auf das Review vor.

Ein agiles Team lebt diese Regeln vor und alle ziehen mit. Diese Regeln offen anzusprechen, wirkt Wunder!

Komplexität reduzieren mit einem fixen Stundenplan

Eines der wesentlichen Ziele aller agilen Arbeitsweisen ist es, Komplexität in jeglicher Hinsicht zu reduzieren. Scrum verfolgt dieses Ziel bei seinen eigenen agilen Events durch fixe Timeboxen und definierte Start- und Endzeitpunkte. Wenn ich online mit meinen Teams arbeite, reduziere ich die Komplexität zusätzlich, indem wir immer die gleiche Zoom Meeting ID oder den gleichen Skype-Link nutzen. Irgendwann landet man natürlich bei der Idee, einen gemeinsamen Stundenplan für alle agilen Events zu führen, „One Calendar" genannt. In Bild 4.3 sehen Sie beispielhaft den Kalender für eines meiner Teams.

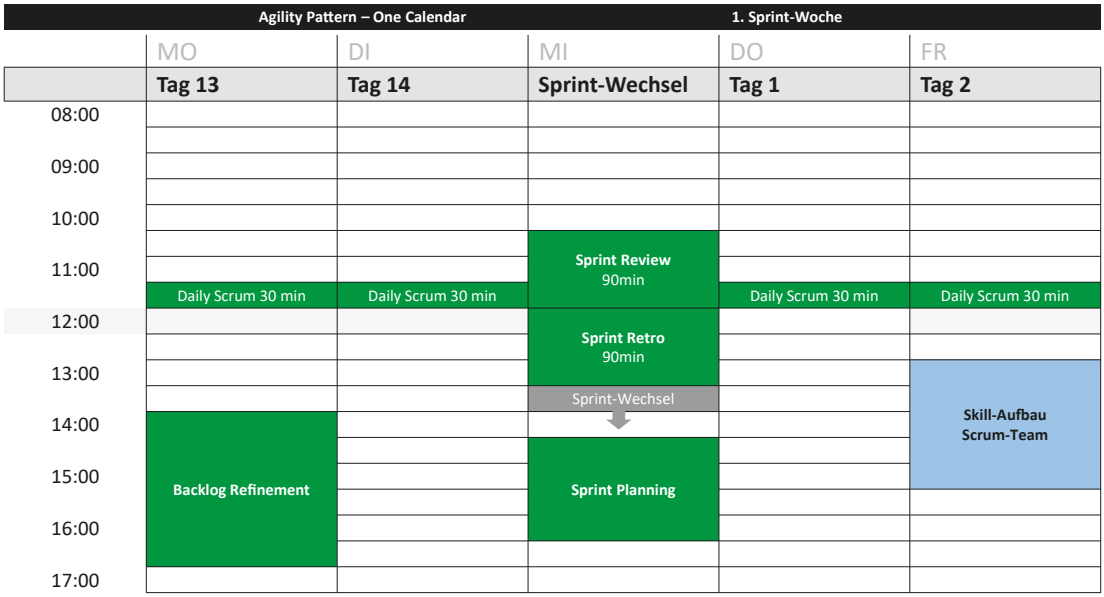


Bild 4.3 One Calendar für Übersichtlichkeit und Routine in den agilen Events

Montagnachmittag findet immer das Product Backlog Refinement statt. Wir haben es so aufgebaut, dass wir im Refinement bereits die Storys schätzen und davor sogar noch die Frage klären: „Wie ist die Story umzusetzen?" Es findet also eine Aufgaben- und Taskplanung statt, wobei hier dazugesagt sei, dass es sich um ein fortgeschrittenes Team handelt. Das erklärt, warum das Sprint Planning zwei Tage später relativ kurz ist.

Das Daily Scrum findet immer zur selben Zeit statt. Viele Agile Coaches und Scrum Master setzen es gleich morgens an, das habe ich früher auch gemacht. Nach der fünften oder zehnten gemeinsamen Überarbeitung des Stundenplans sind wir aber zu dem Schluss gekommen, das Daily kurz vor oder kurz nach der Mittagspause anzusetzen. Damit haben wir auf unsere Langschläfer Rücksicht genommen, aber auch auf den unberechenbaren öffentlichen Personennahverkehr in München oder London. Es ist einfach wichtig, das komplette Team im Daily zu haben, damit jede und jeder über den Arbeitsfortschritt informiert ist.

Wenn wir einen Blick auf den Mittwoch werfen, so ist das der vollkommen durchgetaktete Tag des Sprint-Wechsels. Am Vormittag wird der Sprint abgeschlossen. Mit den immer gleichen Online-Tools und der immer gleichen Zoom-ID starten wir dann in das Sprint Review. In vielen Fällen mache ich vor dem Review mit dem Product Owner und dem Entwicklungsteam eine Generalprobe. Dabei spielen wir einmal durch, was wir den teilnehmenden Stakeholdern in welcher Reihenfolge mit welchen Testdaten live zeigen.

In der Retrospektive gleich nach dem Review blicken wir gemeinsam auf den Sprint zurück. Nach dem Sprint Review wissen wir, wie gut wir mit dem Sprint Forecast gelegen sind. Wir wissen, welche Storys wirklich „done“ waren (siehe Abschnitt 4.3.3) und suchen zusammen nach Verbesserungen, die wir in den kommenden Sprints meist noch als Experimente umsetzen wollen. Mit diesen Ergebnissen geht es erst einmal in die Mittagspause.

Der nächste Sprint startet mit dem Sprint Planning am Nachmittag. Wie gesagt, können wir es im Falle dieses Teams durch das ausgereifte Refinement relativ kurz halten, weil es meistens nur noch um den Forecast für den kommenden Sprint geht. Dabei sehen wir uns die priorisierten, bereits im Detail geplanten und geschätzten User Storys an und legen fest, was wir im kommenden Sprint umsetzen werden.

Eine Besonderheit sehen Sie am Freitagnachmittag: Entsprechend unserer Team-Charta setzen wir 2 bis 2,5 Stunden als sogenannte „Knowledge Hour“ an. Das ist sozusagen unsere teaminterne Weiterbildung. Es handelt sich dabei um eine vollkommen freiwillige Veranstaltung, die vom Team selbstständig organisiert wird und feste Regeln hat. Jeweils ein Teammitglied erklärt den anderen zielgerichtet seine Erkenntnisse anhand einer Mini-Dokumentation mit Skizzen, Videos, Folien etc. Wann immer möglich, schließt dieses Event mit einer kurzen Übung ab.

Dieser gemeinsame Kalender ist gerade für neue Scrum-Teams besonders wichtig. Er bringt eine Struktur in die Woche, an der sich die Teammitglieder ausrichten können. Genau diese strukturierenden Elemente sind es ja, die den Erfolg agiler Methoden ausmachen – wenn man sie entsprechend einsetzt. Daher gelten natürlich auch für diesen Kalender feste Regeln.

Die Events starten immer zur gleichen Zeit.

Die Events finden immer am gleichen Tag statt.

Die Events haben einen festen Stundenplan und Ablauf.

Die Events haben immer Priorität 1 und dürfen von niemandem überbucht werden!

4.3.1 Regeln managen mit dem Delegation Board

In der täglichen Arbeit ist es für Manager oft nicht einfach, das richtige Maß an Delegation zu finden. Besonders schwierig ist das in jenen Phasen, in denen sich die gelebten Werte dramatisch ändern oder anders gesagt: wenn sich klassische Hierarchien zu agilen Netzwerken wandeln sollen. Scrum ist tatsächlich eine Revolution, gerade im SAP-Umfeld. Einerseits erwarten die Scrum-Teams größere Freiräume beim Treffen von Entscheidungen, andererseits hoffen Führungskräfte auf mehr Selbstorganisation in den Teams oder es fällt ihnen extrem schwer, Verantwortung und Entscheidungsbefugnis abzugeben. Fehlende Freiräume genauso

wie fehlende Delegation bergen die Gefahr des Mikromanagements. Statt Autorität ist aber Kollaboration gefragt, sonst verpufft der agile Spirit gleich wieder.

Die Idee der Selbstorganisation ist nichts Neues. Schon Peter F. Drucker hat in „Management Cases“ gesagt:

> *Alle Autorität, die nicht explizit dem obersten Management vorbehalten ist, wird automatisch dem Team gegeben.*
>
> *Alle Autorität, die nicht explizit dem obersten Management vorbehalten ist, wird automatisch dem Team gegeben.*

Nein, wir haben uns nicht verdruckt. Man sollte dieses Zitat einfach zweimal lesen. Den Satz von Drucker ergänzen wir am besten noch durch einen von Jurgen Appelo oft gebrauchten Satz:

> *Manage the system, not the people.*

Wir sollten also nicht Menschen managen, sondern lediglich deren Zusammenarbeit – ob nun zwischen Management und Team oder zwischen dem Team und anderen Parteien. Was man dafür aber braucht, sind glasklare Regeln. Dazu zählt auch die Frage: Welche Entscheidungen trifft das Scrum-Team und welche nicht?

Jurgen zeigt in seinem Buch „Managing for Happiness“ ein einfaches Werkzeug, mit dem sich die geltenden und abgestimmten Regeln zu den Entscheidungshorizonten festhalten lassen: das Delegation Board. Dieses Board kann man auf eine Tafel, ein Whiteboard oder in eine Excel-Tabelle zeichnen. Ich bevorzuge PowerPoint oder Confluence, die umfangreiche, funktionale, aber auch kostenpflichtige Wiki-Software von Atlassian. Wie dokumentiert man nun die Regeln mit dem Delegation Board?

In Bild 4.4 sehen Sie waagrecht von links nach rechts die sieben Ebenen der Delegation – also wie weit reicht die Autorität bzw. Entscheidungsbefugnis, die delegiert wird.

	Delegationsebenen						
Entscheidungsbereiche	**1 = TELL** Entscheide als Manager	**2 = SELL** Überzeuge das Team	**3 = CONSULT** Berate Dich mit dem Team	**4 = AGREE** Entscheide im Konsens	**5 = ADVISE** Trage zur Entscheidung bei	**6 = INQUIRE** Lass Dir die Entscheidung mitteilen	**7 = DELEGATE** Team entscheidet
Keine Modifikationen der SAP Standard-Software	X						
Toolentscheidung Nutzung von Jira		X					
24-Stunden-Refinement-Regel			X				
Tandem-Regeln				X			
Daily Rules					X		
Team-Charta/Working Agreement						X	
MVE (1 oder 2 Sprints)							X
CCO Collective Code Ownership				X			

Bild 4.4 Delegation Board

Von oben nach unten sind die Bereiche aufgelistet, in denen Entscheidungen getroffen werden müssen. Lassen Sie mich diese aufgezählten Entscheidungen anhand eines Beispiels aus meiner Praxis erklären.

Beispiel: Modifikation einer SAP-Standard-Software

Bei ERP und S/4HANA wird das komplette Coding immer mit ausgeliefert und kann natürlich modifiziert werden. Diese Modifikation ist in jedem SAP-Projekt ein heißes Thema. Es gibt Anforderungen, die mit dem SAP-Standard nicht erfüllt werden können, also nimmt man das Coding und passt es an. Ein 08/15-SAP-Entwickler versteht aber anfangs nicht, was er damit anrichtet, wenn er die SAP-Software modifiziert. Diese Veränderung verteuert die Wartung der Software immens, denn die Modifikation muss bei jedem Update nachgezogen und neu getestet werden – und Updates gibt es bei SAP im Abstand von drei Monaten. Daher muss ganz klar festgehalten werden, welche Entscheidungen in puncto Software vom Entwicklungsteam getroffen werden können und welche nicht. Das geht nur mit 1 = Tell.

1 = Tell [verkünden]

Über manche Entscheidung kann oder soll es aus den unterschiedlichsten Gründen keine langwierigen Diskussionen geben. „Tell" bedeutet also: Eine Person trifft die Entscheidung und teilt sie den anderen mit, die von dieser Entscheidung betroffen sind, und erklärt idealerweise die Motivation hinter dieser Entscheidung. In unserem Beispiel bedeutet das: Ein Manager verkündet, dass es keine Modifikationen im SAP-System geben wird, ohne Wenn und Aber. „Wir haben eine Standard-Software gekauft. Punkt. Aus." Eine Begründung wird nicht mitgeliefert. Warum auch, denn wer als SAP-Berater dafür eine Begründung braucht, ist in der SAP-Welt wirklich fehl am Platz. Expertinnen und Experten wissen, welche Nachteile SAP-Modifikationen nach sich ziehen und wie groß der Kostenblock ist, der durch eine modifizierte Standardsoftware in der Zukunft entsteht.

2 = Sell [verkaufen]

Auf der zweiten Stufe der Delegation wird die Entscheidung noch immer von einer Person getroffen, aber zumindest wird ein „Warum" mitgeliefert. Der oder die Entscheidende versucht deutlich zu machen, weshalb die Entscheidung richtig ist. In unserem SAP-Projekt wäre das etwa die Entscheidung für ein Collaboration-Tool: Freeware, Paperware – es gibt viele Möglichkeiten, vom Whiteboard an der Wand bis zu JIRA. Warum wird in unserem Beispiel nur JIRA für die Nutzung freigegeben?

JIRA ist der weltweite Standard. Bei den jährlichen Scrum Days fragt Jeff Sutherland jedes Mal, wer von den etwa 600 Teilnehmerinnen und Teilnehmern im Auditorium JIRA nutzt. Jedes Mal gehen fast alle Hände hoch. Ich möchte keine Werbung für JIRA machen, aber damit konnte ich als Agile Coach und Trainer bisher alle Aufgaben und Herausforderungen in Projekten lösen. Selbst wenn es um die richtig großen Fragen im Projektmanagement, Programm-Management, in der Kapazitäts-, Release- oder Test-Planung geht, kommt mit dem „Portfolio-Plug-in" ein mächtiges Add-on zum Zug, das von der ganzen Community genutzt wird. Ergänzt wird JIRA oft um ein Testmanagement-Tool wie Xray: Dieses Tool erlaubt es einem Team, alle Arten von Tests aufzubauen und in Plänen wiederholt und integriert zu testen.

3 = Consult [befragen]

Bevor eine Entscheidung von einer Person getroffen wird, werden die anderen um ihre Sichtweisen gebeten, die so gut wie möglich in die Entscheidung einfließen. Im Falle eines

Scrum-Teams bittet der Product Owner die anderen Teammitglieder um ihren Input. Im Beispiel würde der Product Owner, basierend auf seinem Vorschlag, 24 Stunden vor dem Refinement die Prioritäten und die anstehenden Themen bekannt geben. Das Team kann sich dann gezielt auf das Refinement vorbereiten.

4 = Agree [sich einigen]
Auf Stufe 4 treten alle Beteiligten in eine Diskussion über das zu entscheidende Thema ein. Aus dieser Diskussion entsteht eine gemeinsame Entscheidung im Konsens – jede Stimme ist gleichberechtigt. In meinem Beispiel habe ich dem Team die Collective-Code-Ownership-Regelung vorgestellt. Jede Änderung am Code wird allen Teammitgliedern vorgestellt, denn der Code „gehört" dem Team (siehe dazu Abschnitt 4.9.1). Wir haben sie zuerst in zwei Sprints ausprobiert und dann in unser Working Agreement als verbindliche Regel aufgenommen.

5 = Advise [beraten]
Am fünften Delegation-Level wird den anderen die eigene Meinung lediglich angeboten. Das ist im Grunde die Position des Scrum Masters: Er oder sie kann nur hoffen, dass seine bzw. ihre Sichtweise in die Überlegungen des Teams einbezogen wird, aber das Team trifft schlussendlich die Entscheidung. Meine Aufgabe im Beispiel war es, den Scrum Guide in- und auswendig zu kennen und das Daily Scrum mit seinen Regeln und Abläufen als gute Praxis vorzustellen. Natürlich habe ich gehofft, dass diese Vorschläge vom Team bei der Entscheidung berücksichtigt werden, aber ich konnte (und wollte) die Entscheidung nicht lenken.

6 = Inquire [sich erkundigen]
Die Entscheidung wird völlig dem Team überlassen, zum Beispiel vom Scrum Master oder dem Product Owner. Wie es Jurgen Appelo sagt, ist Level 6 das Spiegelbild von Level 2: Das Team muss den Delegierenden davon überzeugen, dass die Entscheidung sinnvoll und klug ist. In besagtem Projekt war das der Fall mit den Regeln für alle weiteren Meetings und die Team-Organisation. Hilfreich ist es natürlich, wenn das Team diese „Team-Charta" oder das „Working Agreement" für die Mitglieder auf dem Delegation Board visualisiert.

7 = Delegate [delegieren]
Level 7 ist die Meisterklasse der Delegation. Alle Entscheidungen werden dem Team überlassen. Der Berater, Scrum Master, Product Owner oder Manager will gar keine Details mehr dazu haben. Das Team kann zum Beispiel selbstorganisiert bestimmen, wie lange „Minimum Viable Experiments" (MVE) laufen – einen Sprint, zwei, drei oder mehr. Dabei handelt es sich meistens um Ideen aus einer Retrospektive, die für einen begrenzten Zeitraum ausprobiert werden.

Das Delegation Board funktioniert auf jeder Ebene: zur Visualisierung der Regeln innerhalb eines selbstorganisierten Teams, für die Zusammenarbeit mit dem Product Owner und dem Scrum Master oder für die Kooperation mit anderen Teams. Übrigens muss ein Team nicht darauf warten, bis ein Manager mit so einem Board um die Ecke kommt. Jedes Team kann seine Regeln so visualisieren und in jeder Retrospektive hinterfragen, erweitern oder ergänzen.

Das TAD Delegation Board

Im Laufe der Jahre habe ich das Delegation Board für meine Zwecke noch etwas vereinfacht (Bild 4.5). Ich habe nämlich festgestellt, dass manchen Kunden das Delegation Board in seiner ursprünglichen Form zu kompliziert ist – und bevor etwas Sinnvolles nicht verwendet wird, suche ich lieber nach einer simpleren Lösung.

Entscheidungsbereiche	1 = TELL Entscheide als Manager	5 = ADVISE Trage zur Entscheidung bei	7 = DELEGATE Kein Einfluss, Team entscheidet
Sprint-Dauer = 2 Wochen	X		
Development Team arbeitet nach Scrum	X		
Relatives Schätzen	X		
Relatives Schätzen – Fibonacci			X
Magic Estimation		X	
Relatives Schätzen auch für Defects		X	
100 % Agile – Storys und Spikes		X	
Offene Fragen: Spikes nur im Tandem			X
Team-Charta/Working Agreement			X

Bild 4.5 Das TAD Delegation Board

In manchen Fällen reicht es aus, sich auf drei Level zu beschränken. Dadurch wird das Board übersichtlicher und alle Regeln passen auf eine Seite. Die drei Level lauten:

- Delegation Level **T = Tell** oder manchmal auch **V wie verbindlich**
- Delegation Level **A = Agree** – die Entscheidung erfolgt im Konsens
- Delegation Level **D = Delegate** – die Entscheidung liegt beim Team

4.3.2 Die Definition of Ready (DoR)

Bevor das Team ein Product Backlog Item (PBI) in den Sprint nehmen und umsetzen kann, stellt sich die Frage: Ist in der Beschreibung dieses Items schon alles enthalten, was das Team für die Umsetzung wissen muss? Ist das Item also bereit oder „ready"? Jeff Sutherland hat es – frei übersetzt – auf einfachste Weise so beschrieben: „Es ist bereit, wenn das Team sagt: Wir haben's!" Er bezeichnet das Backlog Item dann auch als „ready ready".

„Ready" oder „bereit" bedeutet, dass ein Product Backlog Item in einem für die Aktion geeigneten Zustand ist. Das wirkt sich direkt auf die Arbeit des Teams aus, denn nur wenn die Anforderung klar ist, lässt sich eine Aufgabe schnell erledigen. Mit einer klaren Definition of Ready haben wir also ein wichtiges Instrument für höhere Produktivität in der Hand. Sie ist eine Liste von Kriterien, die ein Product Backlog Item erfüllen muss, damit das Team mit seiner Arbeit daran im nächsten Sprint beginnen kann. Weder im Scrum Guide 2017 noch 2020 steht zu lesen, welche Kriterien das sind. Im deutschen Scrum Guide 2017 finden wir lediglich folgende Aussage:

> *„Die Product-Backlog-Einträge, mit denen sich das Entwicklungsteam im kommenden Sprint beschäftigen soll, werden so weit verfeinert, dass jeder von ihnen innerhalb des Sprints fertiggestellt werden kann. Product-Backlog-Einträge, für die das der Fall ist, werden als bereit (‚ready') für die Auswahl durch das Entwicklungsteam in einem Sprint Planning angesehen."*

Bill Wake hat aber mit den INVEST-Kriterien eine Liste an notwendigen Eigenschaften eines Product Backlog Item (PBI) formuliert, die sehr hilfreich für die Formulierung von Kriterien sind:

- **Independent and immediately actionable:** Das PBI sollte unabhängig von anderen PBIs sein und sofort umgesetzt werden können.
- **Negotiable:** Die PBIs sind keine Vorschriften, sondern lassen Raum für Beratungen, Rückfragen, Verhandlungen und Verbesserungen.
- **Valuable:** Das PBI liefert einen direkten Wert für den Kunden.
- **Estimable:** Die Größe eines PBI kann geschätzt werden.
- **Small:** Ein PBI ist so groß bzw. so klein, dass es in einem Sprint umgesetzt werden kann.
- **Testable:** Das PBI bzw. dessen Beschreibung liefert die entsprechende Information, damit es nach der Fertigstellung getestet werden kann (im Beispiel anhand der Akzeptanzkriterien).

Zu den INVEST-Kriterien werde ich beim Product Backlog Refinement noch etwas mehr sagen. Fürs Erste zeigt Ihnen Bild 4.6 aber mal eine Definition of Ready, die anhand dieser Kriterien formuliert wurde.

#	DoR-Kriterium	Erklärung
1	Die Story ist klar verständlich	Die Story ist vom Entwicklungsteam „ready, ready“ verstanden
2	Die Story ist klein genug für einen Sprint	Hier haben wir oft Team-Regeln dazu aufgebaut
3	Jede Story ist mit Story Points geschätzt	Relativ geschätzt in Story Points
4	Die Story hat Akzeptanzkriterien	Die genannten Akzeptanzkriterien sind auch verstanden!
5	Die Story ist wertvoll	„Was bringt's?“ Welchen Nutzen bringt die Story?
6	Die Story hat einen Kurznamen	Wir vermeiden alle technischen Beschreibungen im Kurznamen
7	INVEST-Kriterien sind dokumentiert	Wir nutzen zur Verfeinerung die INVEST-Kriterien
8	Die Story ist vom Product Owner priorisiert	Prio 1, 2, 3, 4, 5, 6 und nicht ⇨ high, medium, low

Bild 4.6 Definition of Ready anhand der INVEST-Kriterien

4.3.3 Die Definition of Done (DoD)

So wie ein Entwicklungsteam wissen muss, wann es mit einem PBI bzw. einer User Story loslegen kann, muss es auch wissen, wann die User Story als umgesetzt betrachtet werden kann. Die Definition of Done sagt uns also, welche Voraussetzungen erfüllt sein müssen, damit etwas als „fertig“ bezeichnet werden darf oder wir von einer potenziell auslieferbaren Funktionalität sprechen können. Wir können es auch so ausdrücken: Die Definition of Done ist ein gemeinsames Qualitätsverständnis aller Teammitglieder. Nur wenn die Story, neben den Akzeptanzkriterien, alle angeführten Qualitätsansprüche erfüllt, darf sie als „fertig“

bezeichnet und in das Produktinkrement integriert werden. Ist das nicht der Fall, wandert sie wieder in das Product Backlog zurück. In Bild 4.7 sehen Sie ein einfaches DoD-Beispiel aus einem SAP-Projekt. Im echten Leben sind die Regeln umfangreicher, viel technischer und oft mit den entsprechenden Vorgaben oder Confluence verlinkt.

#	DoD-Kriterium	Erklärung
1	SAP Customizing erstellt/angepasst	Sofern notwendig für die User Story
2	SAP Entwicklungen erstellt/angepasst	WRICEF-Standards erfüllt (=modifikationsfrei umgesetzt), Einschränkungen von WRICEF siehe Akzeptanzkriterien
3	Code Review „Grün“	Die Code-Prüfung auf die vorgegebenen Standards ist durchgeführt, das Ergebnis ist grün, die Entwicklungen, ggf. Customizing sind freigegeben, Code Review ist abgelegt.
4	Performance-Check durchgeführt	Bei ABAP-Entwicklungen ist zusätzlich ein Performance-Check durchgeführt.
5	Testfälle überprüft, ggf. um neue Testfälle ergänzt	Wir nutzen dazu das JIRA-Plugin Xray Test Management.
6	Entwicklertest durchgeführt	Entwicklertest mit Xray in der Story dokumentiert
7	Funktionstest durchgeführt	Funktionstests mit Xray in der Story dokumentiert
8	Integrative Testszenarien (WAT) sind in Xray erstellt	Der Werksabnahmetest WAT/UAT wird in einer eigenen User Story abgebildet. Die Test Cases, Testfälle und Testpläne sind vorzubereiten.
9	Produktpräsentation vorbereitet	Wir zeigen im Sprint Review die Ergebnisse der Story „live“ im Integrationssystem. Die Transporte sind dazu durchzuführen, ggf. sind Testdaten aufzubauen.
10	Dokumentation	Dokumentation entsprechend den Doku-Richtlinien in englischer Sprache erstellt
12	Regeln der Collective Code Ownership erfüllt	Das notwendige Customizing, die notwendigen Erweiterungen nach WRICEF, wurde dem Entwicklungsteam vorgestellt. Unsere Clean-Code-Regeln sind geprüft.

Bild 4.7 Beispiel für eine ausformulierte Definition of Done

Das Formulieren der Definition of Done auf Basis der Vorgaben des Unternehmens ist ein Gemeinschaftsprojekt. Das heißt, der Scrum Master, der Product Owner und das Entwicklungsteam arbeiten zusammen daran und bauen die DoD auf Basis der vorhandenen Architektur sowie Dokumentations- und Programmierregeln auf. Die DoD ist eine knallharte Liste, die nur zwei Zustände kennt: „done“ und „nicht done“. Wenn auch nur ein Punkt auf dieser Liste nicht vorhanden ist, dann ist die Story nicht fertig. Es gibt kein Wenn und Aber, keine Diskussion, ob wir vielleicht 80 Prozent der Story Points als Gutschrift bekommen. No way. Die Story kommt nicht in das Sprint Review, sie wird nicht live gezeigt, die Story ist nicht done.

Glauben Sie aber nicht, dass Sie damit für immer von allen Diskussionen über fertig oder nicht fertig sofort verschont bleiben. „Aber der Report ist doch getestet, die Akzeptanzkriterien sind erfüllt, alle sind glücklich, wir könnten live gehen. Die technische Dokumentation fehlt halt noch.“

Versuchen Sie es daher klar, unmissverständlich und ein für alle Mal festzuhalten: Die Story ist unter diesen Umständen nicht done, es gibt in diesem Sprint keine Story Points für diese Story. Die Dokumentation muss nachgeliefert werden – dann, und nur dann ist die Story done. Lassen Sie solche Diskussionen erst gar nicht zu.

Dem Agilisten Scott Downey hat es mit den ewigen Verhandlungen irgendwann gereicht und er hat seine „Shock Therapy“ entwickelt. Mit der Diskutiererei werden Sie es vor allem in frisch formierten Scrum-Teams zu tun haben, die noch entsprechend wenig Ahnung von Scrum und seinen Regeln haben. Deshalb stellt Scott Downey mit mehr oder weniger Nachdruck gleich mal seine Regeln auf: Das Team folgt seinen Ansagen ohne Meckerei und Verhandlungen, bis es hyperproduktiv ist, drei Sprints erfolgreich gemeistert hat und einen guten Grund aus geschäftlicher Sicht gefunden hat, um die Regeln zu ändern. Für die Definition of Done gibt Scott die folgenden Regeln vor:

- Feature fertig
- Code fertig
- Keine bekannten Fehler
- Abgenommen durch den Product Owner
- Bereit für die Übernahme in die Produktivumgebung

Dieser „Schock“ ist so wieder nicht auf gekaufte Standard-Software wie SAP ausgelegt. Aber machen Sie gedanklich einfach die folgenden Änderungen: „Customizing fertig“, „Prozess fertig eingestellt“, „kurz dokumentieren und testen“ – dann kann der Schock schon mal wirken.

Natürlich gibt es in der SAP-Gedankenwelt von den strikten DoD-Regeln eine einzige Ausnahme, die ich in Abschnitt 4.4 genauer beschreiben werde: Design User Storys. Diese nutze ich, um dem Kunden, dem Product Owner oder den Endanwendern einen Prototyp zu zeigen. Sie sind ein Mittel, um Feedback zu sammeln und damit die Risiken bei größeren Add-on-Entwicklungen, Prozessen und komplizierten Reports zu reduzieren. Für diesen Fall nutze ich gerne eine reduzierte Definition of Done, denn Dokumentationen und umfangreiche Tests brauchen wir bei Design-Prototypen eher selten.

4.3.4 Die Sache mit der Dokumentation

Wenn wir gerade bei der Definition of Done sind, kommen wir um das Thema „Dokumentieren – ja, nein, vielleicht?“ nicht umhin. Bei oberflächlicher Betrachtung des agilen Manifests könnte man schnell zu dem Schluss kommen, dass wir von nun an auf alle Dokumentationen verzichten. Schließlich steht da:

> ***Funktionierende Software*** *mehr als umfassende Dokumentation*

Also liefern wir nur noch funktionierende Software und Prozesse, und die Sache ist geritzt? Aber ich kenne auch das andere Extrem. Auf einer Party nach einem erfolgreichen Release-Go-Live habe ich gehört: „Wir nutzen jetzt agile Prozesse, weil wir aufgehört haben, Dokumentationen zu schreiben!“

So einfach ist das natürlich nicht. Wenn wir im Manifest weiterlesen, steht da durchaus, dass Dokumentation immens wichtig ist. Wir betrachten lediglich funktionierende Software als wichtiger.

Die Doku-Regeln der Vergangenheit

In unseren SAP-Wasserfallprojekten und länderspezifischen Rollouts haben wir dokumentiert, dokumentiert und nochmal dokumentiert. Für jeden Schritt des Customizings, selbst

für das SAP-Grund-Customizing, gab es eine Dokumentation. Wenn wir dann noch über eine ABAP-Formroutine die SAP-Preisfindung erweitern mussten, kamen weitere Dokumentationsvorgaben hinzu. So richtig spannend wurde es, wenn wir über ein eigenes ABAP-Programm einen Prozess erweitern oder anpassen mussten: Dann mussten wir eine sogenannte Anbau-Doku (Add-on-Doku) anlegen, versionieren, das Add-on beantragen und auf vielen Seiten das Design mit Visio-Charts beschreiben. Im zweiten Teil wurde dann der ABAP-Code, die technische Dokumentation, in das Word-Dokument kopiert und beschrieben.

Der glasklare, selbsterklärende ABAP-Quellcode musste also noch einmal Schritt für Schritt in Worten erklärt werden – manchmal nur in Deutsch, manchmal in Deutsch und Englisch. Zusätzlich mussten wir die Anwender-Dokumentation mitliefern, und das auch zweisprachig. Allerdings wussten wir nicht, für wen wir diese Dokumentation schrieben – die Zielgruppe blieb im Dunkeln. Für wen erstellen wir die Dokumentation? Für die IT, die zukünftigen Entwickler, für den Fachbereich, für den Endanwender, für Kollegen an der Hotline?

Agile Dokumentationsideen

Natürlich brauchen auch unsere durch User Storys entstandenen Prozesse eine entsprechende, oft auch schriftliche Dokumentation. Allerdings nicht so wie früher, als wir hinter jeden ABAP-Befehl eine Erklärung geschrieben haben.

Die Prozesse, die wir dokumentieren, entstehen in der agilen Welt durch Erzählen, Aufzeichnen, Haftnotizen oder Karteikarten. Wir verlinken unsere Notizen, wir machen Skizzen und Fotos und dokumentieren diese zum Beispiel mit einem Video. Jeff Patton bezeichnet das als den „Urlaubsfaktor der Dokumentation“: Wenn wir die Bilder, Skizzen etc. sehen, erinnern wir uns genau an das, was wir gemacht haben. Entscheidend ist dabei: Diese Dokumentation entsteht nicht im stillen Kämmerlein, sondern im Austausch zwischen allen Teammitgliedern.

Versuchen wir also nicht mehr, die perfekte Dokumentation zu erstellen! Hören wir auf, Word-Dokumente anzulegen, Masken reinzukopieren und diese Feld für Feld mit Pfeilen und Hinweisen zu erklären und zu übersetzen. Wie wir noch beim User Story Mapping sehen werden, sprechen wir in der agilen Software- und Prozessentwicklung von Geschichten (Storys). Wir drücken mit diesen Geschichten aus, wie wir einen Prozess nutzen werden. Wir tauschen uns darüber aus und ergänzen die Geschichten um Bilder, Skizzen und weitere Bilder – und dabei überlegen wir uns genau, für wen wir die Dokumentation erstellen.

Im SAP-Umfeld liegt die Wahrheit sicher zwischen den Extremen, zwischen „gar nicht dokumentieren“ und „penibel und redundant dokumentieren“. Für alle, die nach DIN EN ISO 9001 oder einer entsprechenden Alternative (z. B. C2E – Committed to Excellence) zertifiziert sind oder eine Zertifizierung planen, sind die gerade vorgestellten Doku-Regeln sicher ein Albtraum. Tatsächlich lassen viele Vorgaben der DIN-ISO-9001-Welt, zum Beispiel die notwendigen Prozessbeschreibungen, bereits jetzt viele Darstellungsvarianten zu. Vielleicht können Sie ja doch statt der schwer lesbaren Visio-Charts einige Bilder und Skizzen, die im Team erstellt wurden, einfließen lassen. Inzwischen gibt es viele Tools am Markt, die uns das Dokumentieren auf diese Weise erleichtern.

So viel wie nötig, so wenig wie möglich – so schaffen Sie eine schlanke QM-Dokumentation. Versuchen Sie generell, die Dokumentation zu verschlanken und zu entschlacken.

Aus meiner eigenen Praxis kann ich Ihnen vier weitere Tipps geben:

1. **Dokumentieren Sie so spät wie möglich.** Wenn Sie mit Ihrem Team agil arbeiten, verändern sich die Ideen, Vorschläge und Designs während der Umsetzung von User Storys ständig. Beginnen Sie mit der Benutzer- und Support-Dokumentation, wenn Sie mit der Entwicklung, dem Customizing oder dem Design fertig sind. Auf diese Weise dokumentieren Sie, was tatsächlich umgesetzt wurde. Eine hohe Qualität ist dadurch gewährleistet und es wird der aktuelle Stand dokumentiert.
2. **Vereinfachen Sie die Dokumentation.** Eine umfangreiche Dokumentation ist keine Garantie für den Projekterfolg. Ganz im Gegenteil: Umfangreiche Dokumentationen sind durch ihre Unübersichtlichkeit fehleranfälliger als kurze und prägnante Aufzeichnungen. Was bringt eine 50-seitige Dokumentation, auf deren ersten zehn Seiten nichts anderes zu finden ist als Einführung, Status und Versionsangaben? Lässt sich die Doku einfacher darstellen und auf fünf Seiten zusammenfassen? Wenn ja, dann tun wir das doch!
3. **Fragen Sie immer nach der Zielgruppe.** Für wen schreiben wir die Dokumentation? Identifizieren Sie zuerst die möglichen Benutzer, bevor Sie mit der Dokumentation loslegen. Es kann zuerst durchaus ein grober Überblick sein, den Sie dann immer mehr nachschärfen.
4. **Gute Kommunikation erleichtert das Verstehen der Dokumentation.** Meistens sind es die Wartungsentwickler oder SAP-Berater, die die Funktionsweise des SAP-Systems für den Prozess verstehen müssen, um ihn im Laufe der Zeit weiterentwickeln zu können. Für diese Personen brauchen wir eine leicht und schnell verständliche Dokumentation. Noch besser ist es, wenn wir den Wissenstransfer schon vorher aktivieren und während des gesamten Projekts mit diesen Personen eine gute Kommunikation pflegen.

Ich gebe zu, dass ich oft keine Möglichkeit hatte, in puncto Dokumentation irgendwie beratend einzugreifen. Je nach Kundensituation gibt es manchmal nur einen Link zu den Dokumentationsvorgaben, die nach der Definition of Done zu erfüllen sind – sonst gibt es bei der Abnahme kein „done“. Das ist auch im Jahr 2020 noch in vielen Projekten die bittere Wahrheit. Doch ich vertrete – so wie viele andere Agile Coaches – weiter meinen Denkansatz und bin sicher, dass sich irgendwann etwas bewegen wird.

4.4 Arbeiten mit dem Minimum Viable Product

Agiles Arbeiten mit Scrum ist in erster Linie eine Möglichkeit, die Risiken in der Entwicklung oder Modifikation von Softwareprodukten zu minimieren. Das ist der Sinn hinter der Arbeit in Iterationen und dem frühem Nutzerfeedback zu sogenannten Produktinkrementen. Dieses Prinzip lässt sich hervorragend mit einem weiteren Ansatz ergänzen, nämlich mit jenem des „Minimum Viable Product“ (oder kurz MVP).

Ursprünglich wurde diese Vorgehensweise von Frank Robinson entwickelt, aber erst durch „The Lean Startup“ von Eric Ries (mit Ideen von Steve Blank) hat sie ihren Durchbruch erlebt. Übersetzt bedeutet Minimum Viable Product so viel wie „minimal überlebensfähiges Produkt“. Es ist eine erste testfähige Version, die nur über die grundlegenden Funktionen

verfügt, um den Nutzern den gewünschten Zweck zu demonstrieren. Jeff Patton setzt das MVP mit dem kleinsten Produkt-Release gleich, das erfolgreich die gewünschten Ergebnisse liefert. Ein MVP ist also nicht das mieseste Produkt, das wir anbieten können – obwohl diese Sichtweise in der SAP-Welt einigermaßen verbreitet zu sein scheint.

Wenn ich hier von „Produkt" spreche, dann meine ich damit in der Praxis große, komplexe Storys, die nur mit den wichtigsten Funktionen ausgestattet werden, um ein erstes Feedback vom Kunden zu bekommen. Wir entwickeln oder customizen einen überlebensfähigen Prototyp, auch gerne eine Prozess-Idee, die wir mit den Anwenderinnen und Anwendern erproben können. Mit dem Feedback zum MVP können wir weitere Erweiterungen und Funktionen planen und in den folgenden Ausbaustufen entwickeln. Daher ist Scrum so ein guter Rahmen für diese Idee, weil die validierten Anforderungen fortlaufend in die nächsten Sprints einfließen und die finale Lösung optimal auf ihre Nutzerinnen und Nutzer zugeschnitten sein wird. Auf diese Weise lässt sich das Produkt also bedarfsgerecht gestalten und wir sparen noch dazu Zeit und Geld.

Hamburger-Prinzip statt Skateboard

Bei der eingehenden Beschäftigung mit dem Minimum Viable Product werden Sie im Internet mit 100-prozentiger Sicherheit auf die Erklärung von Henrik Kniberg stoßen (Bild 4.8).

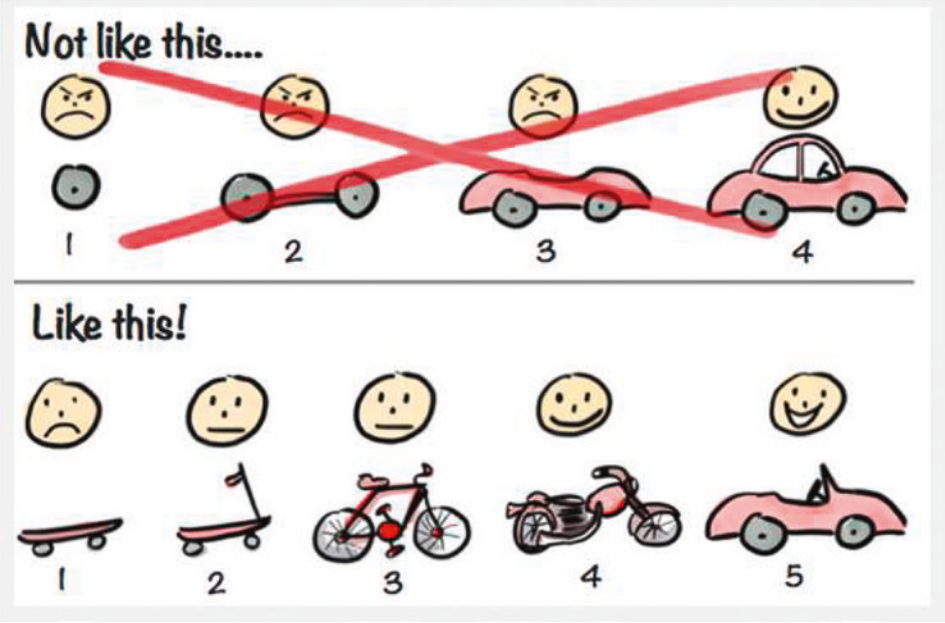


Bild 4.8 Erklärung des Minimum Viable Products von Henrik Kniberg

Die leicht gekürzte Erklärung lautet folgendermaßen: Der Kunde möchte ein Fortbewegungsmittel, um von A nach B zu kommen. Gemäß dem MVP-Gedanken kommen wir diesem Wunsch zuerst mit einem Skateboard entgegen. Das erfüllt zwar grundsätzlich den Zweck der Fortbewegung, ruft aber noch keine Begeisterungsstürme hervor. Im Feedback zeigt sich, dass der Kunde gerne eine bessere Lenkmöglichkeit hätte, er würde die Geschwindigkeit

gerne erhöhen können und überhaupt soll das ganze rot sein. Über Fahrrad und Motorrad landen wir schließlich bei einem Auto.

Das ist jetzt von mir vielleicht etwas lieblos dargestellt, aber ich gebe zu: Ich mag das Bild überhaupt nicht und muss es doch immer wieder erklären. Wer wie ich mit den Führungskräften von Automobilherstellern arbeitet, kriegt sehr schnell zu hören, dass ein Autobauer mit den Ideen nie bei einem Skateboard beginnen würde – selbst wenn das Skateboard die Farbwünsche der Kunden liefern und genauer beschreiben könnte. Für einen Automobilhersteller ist das einfach ein doofes Beispiel. Also habe ich mir mit meinem geschätzten Kollegen Peer etwas anderes ausgedacht, das ein wenig besser in die VUCA-Welt der Automobilhersteller und -zulieferer passt. Wir erklären das Minimum Viable Product anhand des Hamburger-Prinzips.

Statt scheibchenweise kleine Storys auszuliefern, die der Kunde weder nutzen noch zu denen er ein echtes Feedback beisteuern kann, liefern wir ihm lieber kleine, aber dafür komplette Mini-Hamburger. Der Hamburger wird mit jedem Feedback leckerer, weil er mehr und bessere Zutaten bekommt und schließlich in einem tollen Big-Menü geliefert wird (Bild 4.9).

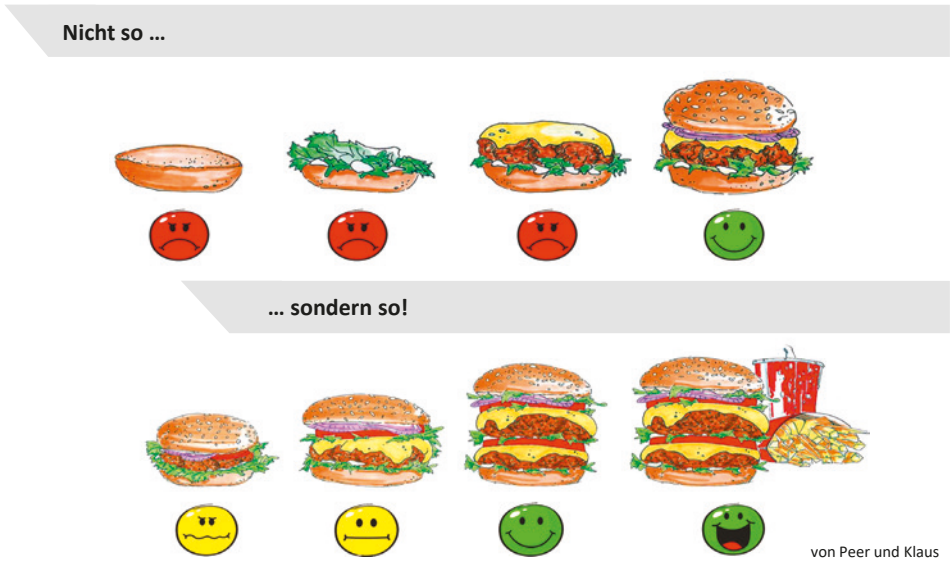


Bild 4.9 MVP anhand des Hamburger-Prinzips

Ein „minimal überlebensfähiges Produkt“ ist für mich also ein Produkt mit den minimalen Anforderungen und Eigenschaften. Es ist ein Produkt in seiner kleinsten Form, ein Prototyp für die Reduzierung des Risikos, eine Grundlage für weitere Ausbaustufen.

Hamburger priorisieren und Hürden meistern

Welche Funktionen sind überlebenswichtig, im Sinne von „viable“? Um diese Frage zu beantworten, lohnt es sich zum einen, bereits umgesetzte Produkte, Prozesse und Storys zu analysieren. Zum anderen müssen Sie einfach mit der Zielgruppe der SAP-Anwender ins Gespräch kommen, um zu verstehen, welche Funktionen am besten ankommen.

Sobald diese Informationen gesammelt sind, werden die gewünschten Funktionen nach der MuSCoW-Methode in vier Gruppen eingeteilt:

1. **Must-have.** Darunter fällt eine begrenzte Anzahl von Funktionen (oder eine einzige Funktion), die wirklich entscheidend sind, um ein MVP-Projekt zu starten und mit den Anwendern zu testen.
2. **Should-have.** Um diese Funktionen können wir das Produkt bei den nächsten Releases erweitern, um für die Anwender einen noch größeren Mehrwert zu schaffen.
3. **Could-have.** Das sind Funktionen, die ganz nett und praktisch sind, aber aus den unterschiedlichsten Gründen jetzt (noch) nicht implementiert werden (müssen).
4. **Won't-have.** Dazu gehören Funktionen, die für die Bedürfnisse der Anwender entweder nicht von Interesse sind oder bereits durch andere Produkte abgedeckt werden.

Wie der Name schon sagt, reicht ein MVP gerade mal so aus. Wenn wir dem Anwender den Mini-Hamburger statt des Big Menüs vorsetzen, lautet das Feedback ziemlich sicher: Ganz nett, **aber** ... Ganz okay, **aber** ... Gute Idee, **aber** ... Das sollte ein Scrum-Team nicht aus der Bahn werfen. Bei der Vorstellung der ersten MVPs dürfen wir uns keine überschwänglichen Reaktionen erwarten, meistens gibt es wenig bis gar keine Begeisterung.

So widersprüchlich es auch klingt: Wenn die ersten Feedbacks zu unserem MVP so oder ähnlich lauten, dann ist das die Bestätigung für uns, dass wir auf dem richtigen Weg sind. Wir müssen damit leben, dass die ersten MVPs immer unterhalb der Anforderungen und der Begeisterungsschwelle liegen werden – es sollte zu unserem Ansporn werden. Wir müssen nur immer im Auge behalten, dass die Story, das Produkt oder der Prozess auch in seinen ersten kleinen Formen immer „viable" ist. Das betone ich gerade im SAP-Umfeld besonders, denn wenn wir uns das Agile Manifesto vor Augen halten, wird dort die funktionierende Software wichtiger eingestuft als eine ausgefeilte Dokumentation. Auf bunte Design-Dokumente und animierte PowerPoints werden wir nicht das Anwender-Feedback bekommen, das wir brauchen. Sie entsprechen nicht dem MVP-Gedanken.

Design User Storys

Bei allen Sprint Backlog Items, die innerhalb eines Sprints umgesetzt werden, muss das Ergebnis – das Inkrement – danach für den Kunden wirklich nutzbar sein.

> *„Das Inkrement ist das Ergebnis aus allen in einem Sprint fertiggestellten Product-Backlog-Einträgen und dem Resultat der Inkremente aller früheren Sprints. Am Ende eines Sprints muss das neue Inkrement fertig („Done") sein; das heißt, es muss in einem verwendbaren Zustand sein und die Definition of Done des Teams erfüllen. Ein Inkrement ist ein Gegenstand inspizierbarer, fertiger Arbeit („Done"), der die Empirie am Ende des Sprints unterstützt. Es muss auch dann im einsatzfähigen Zustand sein, wenn der Product Owner es aktuell noch gar nicht ausliefern will."*
>
> Scrum Guide 2017, S. 17

Im SAP-Umfeld bedeutet das, dass das fertige Inkrement im Produktivsystem landet und sofort vom Kunden genutzt werden kann. Der Scrum Guide lässt sozusagen durch den letzten Satz in diesem Zitat zu, dass Releases gebildet werden. Ich gönne mir hier meine eigene Interpretation mit einer Abweichung vom Scrum Guide:

SAP ERP, SAP/4HANA oder auch SAP allgemein sowie alle anderen Standard-ERP-Systeme dieser Welt sind so komplex, so ineinander verzahnt und so integriert, dass für mich auch ein vorzeigbarer **Prototyp** als Ergebnis einer Story in Frage kommt.

Bei neuen Prozessen, Anwendungen, Reports bzw. generell bei einer Erweiterung des SAP-Standards erlaube und empfehle ich als Scrum Master oder auch Product Owner sogenannte Design User Storys. Die Voraussetzung: Diese Design User Storys müssen entsprechende Akzeptanzkriterien haben und in einem Prototyp münden, der dem Kunden, Anwender und Product Owner im Sprint Review gezeigt werden kann und der ausprobiert werden kann. Ganz egal, wie wir diese Storys nennen (Mockup, Vorführmodell, Prototyp ...): Es geht immer um das Kundenfeedback. Dieses Prototyping hat das Ziel, anhand der ersten Ergebnisse möglichst früh eine Rückmeldung zum Lösungsansatz zu bekommen. Nur so können wir wissen, ob wir in eine geeignete Richtung arbeiten. Im Bereich der komplexen Standard-Software reduziert das die Risiken signifikant. Es ist also eine erweiterte Praxis zum Minimum Viable Product – und sie hat sich bereits in vielen Projekten bewährt.

Eine Design User Story hat klar definierte Akzeptanzkriterien, die auf dem WRICEF-Standard des SAP Solution Managers aufbauen, der modifikationsfreien Erweiterung aus dem SAP Solution Manager eines SAP-Systems. Wofür steht WRICEF?

W = Workflow

R = Reports

I = Interfaces inbound oder outbound (IDocs, BAPIs etc.)

C = Conversions (die Old-Style-Migrationsmethode mit der SAP LSM Workbench)

E = Enhancements (alle Arten von User Exits, um Programme und Reports zu erweitern)

F = Forms (gemeint sind alle Formulare wie Rechnungen, Angebote, Lieferscheine etc.)

Das alles sind mögliche Erweiterungen eines SAP-Systems, die keine Modifikation darstellen. In der Design Story geben wir die Erweiterungsmöglichkeit vor, zum Beispiel „E (Enhancements) = Prototyp mit einem User Exit“. Genauso geben wir das UI-Design vor (z. B. Fiori App, SAP GUI, UI5 Prototyp oder ein Report aus dem Reporting-Tool) und das Tool, in dem der Prototyp erstellt wird (z. B. SAP Build). Wir erwarten, dass für den Prototyp immer gute Testdaten aufgebaut werden – der Prototyp ist nicht produktiv nutzbar! Für die Erstellung von Prototypen gibt es übrigens mehrere Tools von SAP (zum Beispiel *www.build.me*), mit dem sich innerhalb kurzer Zeit ein Design erstellen und zeigen lässt.

Eine Design User Story hat natürlich eine stark reduzierte Definition of Done (Bild 4.10): Wir erwarten keine umfangreichen Tests durch das Entwicklungsteam, auch keine Tests durch die Endanwender und keine schriftliche Dokumentation. Was wir wollen, ist ein Prototyp zum Anfassen und Vorzeigen sowie eine perfekt vorbereitete Live-Demo im Review.

#	DoD-Kriterium	Erklärung
1	**Prototype/UI/Mockup erstellt**	**Entsprechend den Akzeptanzkriterien der User Story sowie der verfügbaren und erlaubten Design Tools**
~~2~~	~~SAP-Entwicklungen erstellt / angepaßt~~	~~WRICEF Standards erfüllt/modifikationsfrei umgesetzt, Einschränkungen von WRICEF siehe Akzeptanzkriterien~~
~~3~~	~~Code Review „Grün“~~	~~Die Code Prüfung auf die vorgegebenen Standard ist durchgeführt, das Ergebnis ist Grün, die Entwicklungen, ggf. Customizing sind freigegeben, Code Review ist abgelegt.~~
4	~~Performance Check durchgeführt~~	~~Bei ABAP Entwicklungen ist zusätzlich ein Performance Check durchgeführt.~~
~~5~~	~~Testfälle überprüft, ggf. um neue Testfälle ergänzt~~	~~Wir nutzen dazu das JIRA Plugin Xray Test Management~~
~~6~~	~~Entwickler Test durchgeführt~~	~~Entwickler Test mit Xray in der Story dokumentiert~~
~~7~~	~~Funktionstest durchgeführt~~	~~Funktionstests mit Xray in der Story dokumentiert~~
~~8~~	~~Integrative Testszenarien (WAT) sind in Xray erstellt~~	~~Der Werksabnahmetest WAT/UAT wird in einer eigenen User Story abgebildet. Die Test Cases, Testfälle und Test Pläne sind vorzubereiten.~~
9	**Produktpräsentation vorbereitet**	**Wir zeigen im Sprint Review die Ergebnisse der Story „live“ im Integrationssystem. Dazu sind die Transporte durchzuführen, ggf. sind Testdaten aufzubauen.**
~~10~~	~~Dokumentation~~	~~Dokumentation entsprechend den Doku Richtlinien in englischer Sprache erstellt~~
~~12~~	~~Collective Code Ownership Regeln erfüllt~~	~~Das notwendige Customizing, die notwendigen Erweiterung nach WRICEF wurden dem Entwicklungsteam vorgestellt. Unsere Clean Code Regeln sind geprüft.~~

Bild 4.10 Verkürzte DoD für die Design User Story

Design User Storys – was bringt’s?

- Sie haben schnell ein vorzeigbares Ergebnis.
- Sie bekommen frühes Feedback und können Änderungswünsche früh erkennen.
- Sie wissen gleich am Anfang, ob Sie richtig liegen.
- Sie reduzieren das Risiko einer Fehlentwicklung deutlich.
- Sie sparen Zeit und Geld.

4.5 Komplexe Anforderungen zerlegen mit User Story Mapping

Alle, die im SAP-Umfeld arbeiten, wissen: Die Anforderungen sind oft sehr komplex. Manchmal sind mehrere Stunden oder gar Tage notwendig, um diese Komplexität zu durchdringen und das Ganze zu verstehen. Zusätzlich passiert oft das, was Sie in Bild 4.11 sehen: Ein Team bekommt die Anforderungen schriftlich zugeworfen und soll sich darauf einen Reim machen. Nur leider macht sich jedes Teammitglied seinen eigenen Reim auf das, was in diesen Dokumenten steht. Wenn wir das nun alles undifferenziert in ein Product Backlog packen, kommen wir erst im Laufe der Sprints darauf, was wir eigentlich alles falsch oder ganz anders als unsere Kolleginnen und Kollegen verstanden haben – und das kostet wieder viel Zeit.

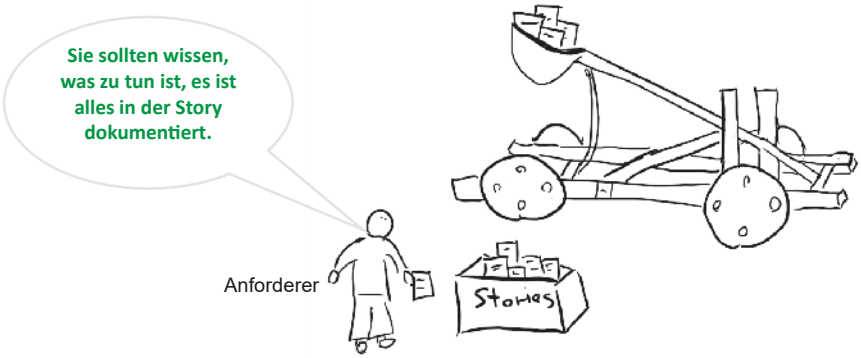


Bild 4.11 Wie Anforderungen auf klassische Weise zum Team gelangen

Außerdem sind Zusammenhänge, Widersprüche oder Irrwege schwerer zu erkennen, wenn die User Storys einfach nur ins Backlog gepackt werden.

Deshalb schätze ich das von Jeff Patton entwickelte User Story Mapping sehr. Ich nutze diese Technik in jedem Projekt und für das Product Backlog Refinement – besonders dann, wenn es unübersichtlich, komplex und schwierig ist. Story Maps helfen mir auch, wenn wir in großen Gruppen, zum Beispiel mit mehreren Teams, ein Epic visualisieren, Prozessschritte abstimmen und mögliche Releases schneiden.

Das User Story Mapping ist eine Methode, um zwischen einem Scrum-Team (den Problemlösern) und dem Anwender ein gemeinsames Verständnis über die Anforderung zu erzeugen. Das heißt, die entstehende Story Map ist ein Gesamtbild der wesentlichen Schritte, die ein Anwender bei der Nutzung des Produkts setzt. Diese wesentlichen Schritte (Backbone) werden noch weiter detailliert. Dadurch ergeben sich einige Vorteile: Zum Beispiel ist es mit der Story Map wesentlich einfacher, ein Minimum Viable Product zu identifizieren, Releases zu schneiden und zu planen oder eine Anforderung aus mehr als einer Perspektive zu betrachten und dadurch auf neue Ideen zu kommen. Wir sehen zwar die Einzelteile, aber wir verlieren nie das Gesamte aus dem Blick.

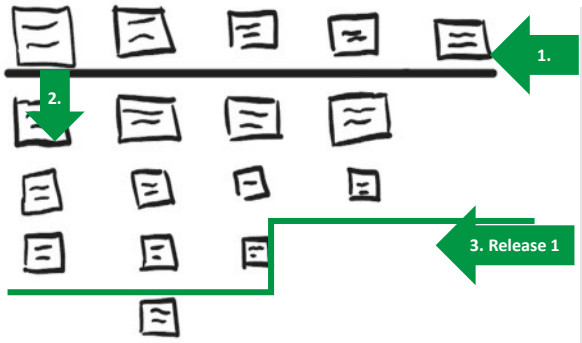
Welche Vorteile bieten Story Maps für SAP-Teams?

- Komplexe Sachverhalte werden übersichtlich in einem Big Picture dargestellt, das sich am Arbeitsfluss der Anwender orientiert.
- Der Fokus liegt auf der User Experience und den Wirkungen, die wir für die Kunden erzielen wollen.
- Mit der Story Map können wir die ersten Schritte einer Release-Planung machen.
- Die Story Map hilft dabei, die Umfänge der Releases zu reduzieren – wir können leichter das passende Minimum Viable Product finden.
- Mit Hilfe der Story Map lassen sich User Storys nach dem MVP-Gedanken erstellen und weitere User Storys darauf aufbauen.
- Es ist einfach ein gutes Instrument, um über die Anforderungen ins Gespräch zu kommen und sie besser zu verstehen.

Mit der User Story Map von Jeff Patton wird klar, dass eine Story nicht einfach eine Anforderung ist, die über den Zaun geworfen und abgearbeitet wird und es nicht mit dem Schreiben von User Storys und Refinen getan ist. Eine Story ist ein Hilfsmittel: Storys heißen Storys, weil sie eben eine Geschichte erzählen, nämlich jene von der Reise des Kunden oder Anwenders durch die Verwendung des Produkts.

Um eine Story Map bauen zu können, müssen wir daher die Perspektive des Anwenders oder Key Users einnehmen. Das hilft uns dabei, nicht mit einer fixen Idee an die Lösung heranzugehen, sondern uns nur einmal die Frage zu stellen: Was tut der Anwender alles mit diesem Produkt, um sein Ziel zu erreichen? Mit der User Story Map kommen wir weg von kleinlichen Diskussionen über Features und reden stattdessen über das, was die Anwender mit diesem Produkt tatsächlich machen.

Wie wird nun eine User Story Map erstellt (Bild 4.12)?



1. Erstelle mit den wichtigsten Nutzungsschritten das Backbone.
2. Füge mit User Storys zu Detailaktivitäten die nötige Tiefe hinzu.
3. Schneide Releases bzw. MVPs aus der Story Map.
4. Arbeite einzelne Storys in Refinements aus.

Bild 4.12 In vier Schritten zur Story Map

Schritt 1 – das Backbone erstellen

Das Backbone ist der übergeordnete Prozess der Nutzung eines Produkts oder einer Funktionalität. Er umfasst die wesentlichen Schritte bzw. übergeordnet die wichtigsten Aktivitäten, die der Nutzer bei der Verwendung einer Software bzw. einer bestimmten Funktionalität setzt. Diese chronologische Reihenfolge (Jeff nennt es „Narrative Flow“) müssen wir zuerst erforschen und dann „erzählen“. Wir bilden auf einem groben Level ab, was sich von außen beobachten lässt, daher werden wirklich alle möglichen Schritte dokumentiert, auch wenn sie zunächst einmal nicht auf derselben Flughöhe sind.

Wichtig ist im Anschluss aber auch, diese Reihenfolge zu hinterfragen, daher ist das User Story Mapping ein ideales Anwendungsgebiet für Haftnotizen. Das Team sortiert so lange um, bis alle mit der Reihenfolge einverstanden sind.

Schritt 2 – die notwendige Tiefe hinzufügen

Jeder Schritt des Backbones umfasst wiederum mehrere Detailaktivitäten, die notwendig sind, um diesen Schritt abzuschließen. Für jeden Schritt stellen wir uns daher die Frage: „Was muss alles passieren, damit wir zum nächsten Schritt weitergehen können?“ Hier können wir alternative Verläufe, Variationen und Ausnahmen durchspielen.

Schritt 3 – Releases schneiden

In Schritt 3 kommt unter anderem das Minimum Viable Product ins Spiel: Wie können wir einen Durchstich durch alle Schritte des Backbones schaffen, um den Anwendern einen ersten Eindruck zum Beispiel der Modifikation zu bieten? Das erste Release sollte eine Lernstrategie beinhalten, um unsere eigenen Annahmen zu überprüfen.

Sobald die Map steht, können die einzelnen Storys im Product Backlog Refinement ausgearbeitet werden.

Story Mapping in der Praxis

Als ich in einem neuen SAP-Projekt als Coach gestartet bin, fand ich sehr große User Storys in einem Format vor, das auf den ersten Blick völlig undurchsichtig war. Aus einem Detail- oder Fachkonzept mit 350 Seiten waren einfach Teile herausgeschnitten und mit den User Storys verknüpft worden. Jede einzelne User Story bestand aus 30 bis 40 Seiten an Beschreibungen und wenig aussagekräftigen Akzeptanzkriterien. Mit einer dieser Riesen-Storys gingen wir in das User Story Mapping.

Runde 1 (10 Minuten): Erklärung des Vorgehens und Anlegen eines Template

Das Team, der Product Owner und ich sahen uns die erste große User Story „SAP Auftragsbearbeitung“ an und zerlegten die unübersichtliche Anforderung nach den Ideen des Story Mappings in einzelne übergeordnete Prozessschritte nach dem Prinzip „Talk and Doc“: In einer DEON Collaboration Session schrieben wir alles Wesentliche auf, während wir darüber sprachen. Die Kärtchen, in DEON „Sticky Notes“ genannt, waren bereits im richtigen Format, in der richtigen Größe, in der richtigen Farbe und in einer lesbaren Schrift vorbereitet

Bild 4.13 Vorbereitung für die Erstellung des Backbone und der Detailaktivitäten

(Bild 4.13). Die Teilnehmer konnten einfach eine gelbe Karte ziehen, den Text überschreiben und auf dem Board platzieren.

Runde 2 (15 Minuten): Die übergeordneten Prozessschritte – das Backbone erstellen
Im nächsten Schritt platzierten wir die übergeordneten Prozessschritte der Auftragsbearbeitung in der passenden Reihenfolge (für bessere Lesbarkeit wird die DEON-Oberfläche nur stilisiert dargestellt; Bild 4.14). Nach einer kurzen gegenseitigen Beratung kamen wir zu dem Schluss, zuerst den Kundenauftrag mit den Auftragskopfdaten (dahinter steckt die Auftragsart) zu bearbeiten und danach die Geschäftspartner auszuwählen. Klarer war uns dann, dass anschließend die Auftragspositionen mit den Verkaufspreisen und -steuern ermittelt werden mussten, die Bezahlarten definiert werden sollten und die Auftragsbestätigung in der einfachsten Form zunächst auf Papier gedruckt werden sollte.

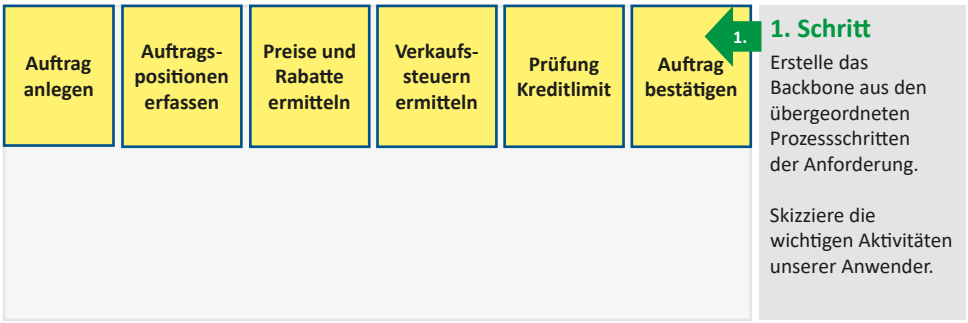


Bild 4.14 Gemeinsam das Backbone erstellen

Runde 3 (15 Minuten): Die Details erarbeiten
Als wir in den Prozessschritten in die Tiefe gingen, wurde deutlich, dass es mehrere Auftragsmöglichkeiten gab: Sofortaufträge, Terminaufträge, Konsignationsläger (die Kundenläger werden beliefert und bestückt, der Kunde bezahlt aber erst bei Warenentnahme) und die recht einfachen Gut- und Lastschriften. Die gleiche Detaillierung wiederholten wir für jeden Prozessschritt (Bild 4.15). Wenn das Story Mapping einmal läuft, sprudeln die Ideen meistens von selbst. Wichtig ist dabei, dass Sie die Ideen gleich aufschreiben und am Board platzieren. Die Teammitglieder können sie dann selbstorganisiert verschieben und sich gegenseitig beraten.

Runde 4 (15 Minuten + Nachspielzeit): Releases bzw. User Storys schneiden
In unserem Beispiel konnten wir noch nicht von Releases sprechen, sondern mussten zunächst User Storys aus dem ursprünglich riesigen Konvolut zurechtschneiden. Natürlich war uns klar, dass die Notizen am Board nur ein Rohformat waren – sie würden dann im Refinement vom Team genauer ausgearbeitet werden. In diesem Fall war die Runde 4 (der dritte Schritt nach Jeff Patton) sehr intensiv, deshalb erweiterten wir die ursprüngliche Timebox von 15 Minuten einmal um zehn Minuten. Das Ergebnis präsentierte sich danach wie in Bild 4.16.

Mit Story 1 schnitten wir ein erstes MVP heraus, um zu zeigen, wie die komplette Auftragserfassung funktionierte. Darauf aufbauend entstanden weitere Ideen zu User Storys – vom Terminauftrag mit Sonderformen der Preisfindung (Story 2) über die Beschickung der Konsignationsläger (Story 3) bis zu Story 4, die den Gutschriftprozess mit der Gutschriftsanforderung einleitet.

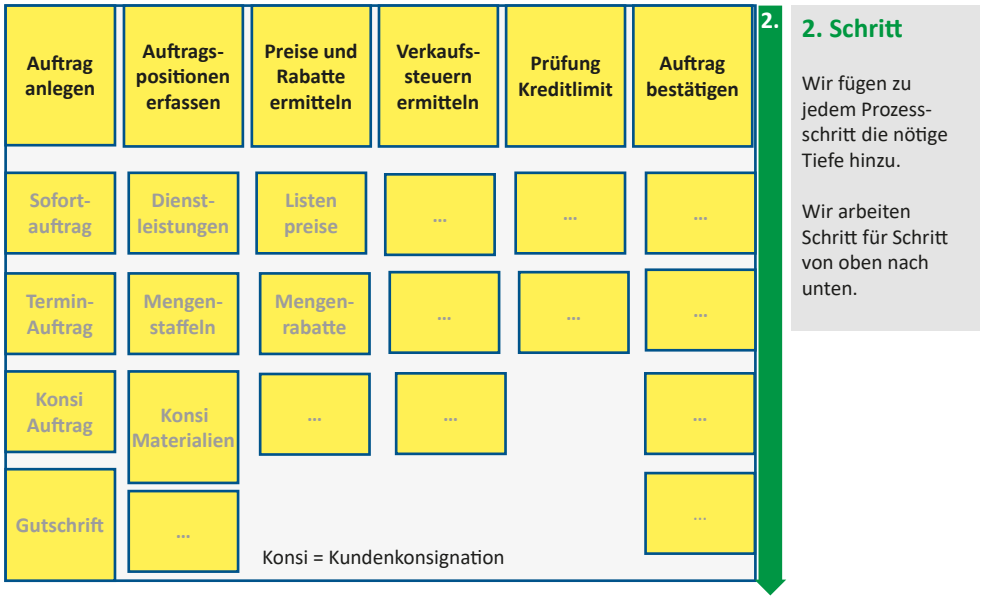


Bild 4.15 Detaillierung der Prozessschritte

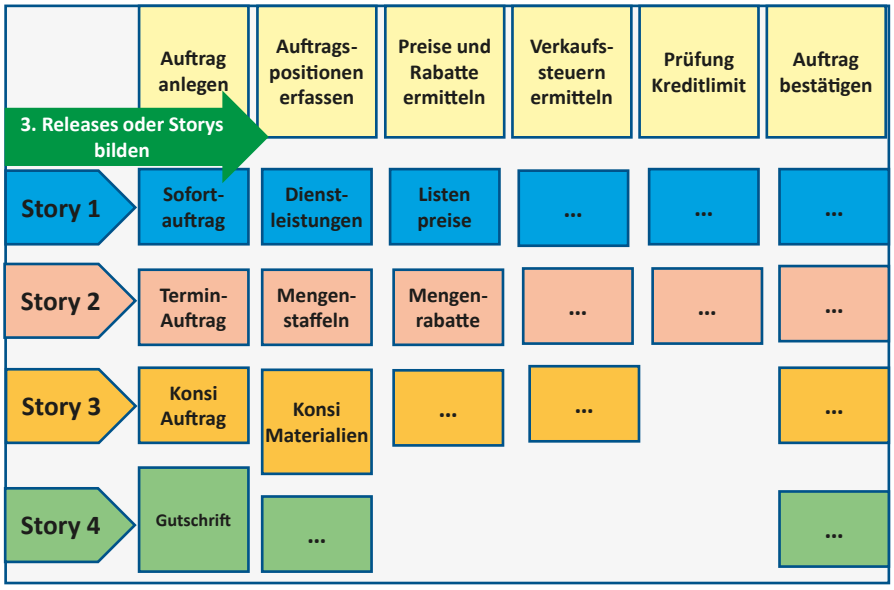


Bild 4.16 Am Ende des Prozesses sind kleinere, sinnvolle User Storys entstanden

Tipps für das Story Mapping

- Nutzen Sie bei einer Online-Session ein gutes und stabiles Collaboration-Tool, das Funktionen wie „Sticky Notes“ bietet. Mit DEON lässt sich das Story-Mapping-Modell sogar als Template vorbereiten.
- Erstellen Sie die Story Maps in gemeinsamen Workshops mit dem Product Owner und den Teammitgliedern.
- Laden Sie – wenn möglich – auch die Stakeholder sowie Anwenderinnen und Anwender zu diesem Workshop ein.
- Mit Haftnotizen können Sie die Story Map einfach visualisieren und bei der Beschäftigung damit die Schritte und einzelnen User Storys immer wieder neu arrangieren.
- Die Gedanken sollten gleich zu Papier gebracht werden, während über die Story gesprochen wird.

4.6 Das Product Backlog Refinement als Motor der Produktivität

4.6.1 Warum das Product Backlog Refinement so wichtig ist

Wahrscheinlich leuchtet es jedem auf Anhieb ein, dass ein Team, das neu an den Start geht, Hilfe beim Teambuilding und mit den Regeln braucht, wie wir sie kennengelernt haben. Auch wenn mit Story Maps Licht in die Komplexität der Anforderungen gebracht wird, hilft das schon einmal enorm.

Nun haben wir es mit Scrum-Teams zu tun, und das noch dazu im SAP-Umfeld. Solche Teams können in einem Sprint nur Geschwindigkeit aufnehmen, wenn der Sprint perfekt vorbereitet wurde. Die Basis dafür sind Product Backlog Items, die „ready ready“ sind (also die Definition of Ready – DoR – erfüllen). Alle Anforderungen sind klar und das Team kann im Sprint sofort loslegen. Jeff Sutherland hat festgestellt, dass sich die Produktivität eines Teams dadurch verdoppeln kann.

Was unsere Teams also brauchen, ist ein ordentliches Product Backlog Refinement. Sehen wir noch einmal im Scrum Guide 2017 nach (S. 14), was das Product Backlog Refinement ist (in der Ausgabe 2020 taucht diese wertvolle Erklärung leider nicht mehr auf):

> *„Als Verfeinerung (Refinement) des Product Backlogs wird der Vorgang angesehen, in dem Details zu Einträgen hinzugefügt, Schätzungen erstellt oder die Reihenfolge der Einträge im Product Backlog bestimmt werden. Die Verfeinerung ist ein kontinuierlicher Prozess, in dem der Product Owner und das Entwicklungsteam gemeinsam die Product-Backlog-Einträge detaillieren. Bei der Verfeinerung des Product Backlogs werden die Einträge begutachtet und revidiert. Das Scrum-Team bestimmt, wann und wie diese Verfeinerungsarbeit erfolgt. Sie sollte normalerweise nicht mehr als 10 % der Kapazität des Entwicklungsteams beanspru-*

Stichwortverzeichnis