O'REILLY®

Deutsche Ausgabe

User Story Mapping

DIE TECHNIK FÜR BESSERES NUTZERVERSTÄNDNIS IN DER AGILEN PRODUKTENTWICKLUNG

Jeff Patton
mit Peter Economy
Übersetzung von Petra Hildebrandt

Inhalt

KAPITEL 4

Planen, (um) rechtzeitig fertig zu werden

Hier seht ihr Aaron and Mike. Sie arbeiten für eine Firma namens Workiva. Workiva stellt eine Produktsuite her, die auf einer Plattform namens Wdesk aufsetzt. Sie löst große Probleme für große Unternehmen, und Workiva ist eine der größten Software-as-Service-Firmen, von denen ihr wahrscheinlich noch nie gehört habt.

Aaron und Mike sehen glücklich aus, oder? Das ist ganz normal bei Leuten, die zusammen schwierige Probleme gelöst haben. Oder liegt es daran, dass der Typ rechts ein Bier in der Hand hält? Nein, das ist

es nicht. Was sie glücklich macht, ist das Gefühl, das man hat, wenn man ein wirklich kniffliges Problem gelöst hat. Das Bier ist nur eine Belohnung. Falls *ihr* nicht ein Bier oder eine ähnliche Belohnung bekommt, wenn ihr komplizierte Probleme gelöst habt, solltet ihr mit jemandem ein ernstes Wort reden.

Aaron und Mike haben mehrere Runden Product Discovery hinter sich und sind zuversichtlich, dass sie etwas haben, das in die Produktion gehen sollte.

Ihr Entdeckungsprozess begann mit dem Umreißen der Feature-Idee, an der sie arbeiteten, um zu begreifen, für wen und weshalb sie dieses entwickeln sollten. Dann haben sie sich direkt mit den Kunden ausgetauscht, um ihre Einschätzungen darüber zu validieren, wie diese arbeiten und was die tatsächlichen Probleme waren. Danach haben sie einfache Prototypen gebastelt. Sie haben diese einfachen elektronischen Prototypen mit Axure umgesetzt und mit ihren Kunden remote getestet – zunächst, um festzustellen, ob denen die Lösung gefiele, und dann, um sicherzustellen, dass sie auch benutzbar war. Sie hatten das Gefühl, dass der Prototyp keine funktionsfähige Software sein musste, um herauszufinden, was sie über das Feature wissen mussten.

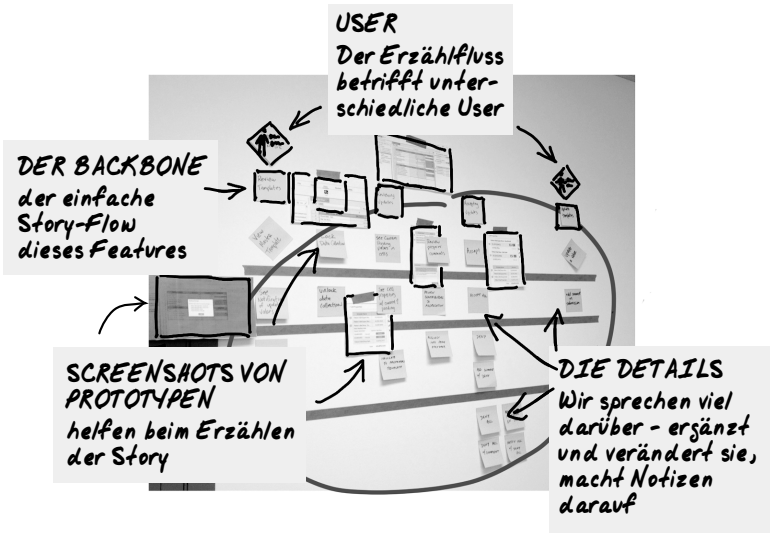
Nach mehreren Iterationen mit einfachen Prototypen waren sie davon überzeugt, etwas zu haben, für das die Entwicklung sich lohnte. Das klingt nach einer Menge Aufwand, kostete sie aber nur drei Tage. Der letzte Schritt bestand dann im Erstellen eines Backlogs und eines Ablieferplans für das Feature. Auf dem Bild seht ihr den Plan. Es ist ein guter Plan, und deswegen sind sie glücklich.

Es ist wichtig, daran zu denken, dass diese Map kein komplettes Produkt abildet, sondern es nur um ein Feature geht, das zu einem bereits bestehenden Produkt hinzugefügt wird. Daher ist die Map kleiner als die von Gary in Kapitel Kapitel 1 oder die des Globo.com-Teams. Ich erwähne das, weil manche Leute fälschlicherweise annehmen, sie müssten das ganze Produkt mappen, wenn sie nur eine kleine Veränderung vornehmen wollen, und sie benutzen das dann als Entschuldigung, gar keine Map anzulegen.

Mappt nur das, was der Unterstüzung eurer Konversation dient.

Redet mit dem Team

Um dieses Feature zu erzeugen, werden die beiden ein gemeinsames Verständnis mit ihrem Team herstellen müssen. Das Team muss dazu in der Lage sein, auf Probleme und Verbesserungsmöglichkeiten hinzuweisen und abzuschätzen, wie lange es brauchen wird. Dafür gibt es diese abschließende Map. Sie benutzten die Map, um die Story des Features zu erzählen – Schritt für Schritt, aus der Sicht des Users. Seht ihr die Bildschirmausdrucke in der Map? Sie können auf Screens und hervorgehobene Details deuten, während sie ihre Geschichte mit der Map erzählen, und die, die ihnen zuhören, können sich dadurch die Lösungen besser vorstellen. Die Leute bei Disney, die Filme mit Storyboards erzählen, können ihnen nicht mal ansatzweise das Wasser reichen.



Erkundigten sich Teammitglieder, warum sich ein Screen so benimmt wie er es tut, konnten sie von den Varianten erzählen, die sie ausprobiert hatten, und wie die User darauf reagiert hatten. Wollte das Team wissen, was genau passiert, wenn Daten eingegeben oder Informationen übertragen werden, hatten sie darüber nachgedacht und eine Antwort parat. Und falls sie es nicht wussten, diskutierten sie darüber mit dem Team und machten sich Notizen auf Klebezetteln, die in das Modell einflossen. Sie ergänzten auch eine Reihe von Notizen zu Details, an die sie nicht gedacht hatten, sondern die das

Team einbrachte. Aaron erzählte mir, dass das Team einige technische Abhängigkeiten entdeckt hatten, die ihm und Mike nie aufgefallen wären.

Die Kunst des gekonnten Schätzens

Jeder, der eine gewisse Zeit im Softwaregeschäft ist, weiß, dass eine der größten Herausforderungen des Jobs darin besteht, abzuschätzen, wie viel Zeit die Entwicklung tatsächlich benötigen wird. Ich verrate euch eines der bestgehüteten Geheimnisse des gekonnten Schätzens:

> **Die besten Schätzungen kommen von Entwicklern, die wirklich verstehen, was sie einschätzen sollen.**

Es gibt eine Menge Methoden, die mit dem Versprechen antreten, sie könnten akkuratere Schätzungen liefern. Ich werde davon hier keine vorstellen. Aber ich *werde* euch verraten, dass keine von denen funktioniert, ohne dass die Leute, die die Software produzieren, ein gemeinsames Verständnis untereinander und mit denen haben, die sich die Software ausgedacht haben.

Der Aufbau von gemeinsamem Verständnis beim Schätzen sollte wirklich kein Geheimwissen bleiben. Ihr solltet dieses Wissen gleich jetzt mit anderen teilen.

Plant, Stück für Stück zu produzieren

Das Team von Workiva kann nicht einfach weniger produzieren an diesem Punkt. Die Teammitglieder können nicht wie bei Globo.com in Kapitel 2 einfach Zeug wegschneiden, denn sie haben bereits validiert, dass sie tatsächlich alles benötigen. Bei der Arbeit mit Prototypen haben sie bereits zahlreiche Eingrenzungen vorgenommen und sichergestellt, dass die Lösung immer noch für die Kunden wertvoll ist. Aber wenn ihr auf ihre Map schaut, seht ihr drei Slices.

»Wieso machen sie sich die Mühe?«, fragt ihr euch vielleicht. Ein Drittel dessen, was der Kunde will, das ist so wie ein Drittel eines Sportwagens auszuliefern. Niemand kann damit fahren. Aber Mike ist der Produkt-Owner. Er ist nicht fertig mit seinem Job, nachdem er eine gute Lösung identifiziert hat. Seine Rolle verändert sich, und nun ist er sowas wie der Regisseur bei einem Film. Er muss bei jeder einzelnen Szene, die aufgezeichnet wird, dabei sein. Und er muss entscheiden, welche Szenen zuerst aufgenommen und welche zum Schluss gefilmt werden. Er weiß, dass an Ende die einzelnen Teile ein Ganzes werden und wie ein zusammenhängender Film wirken müssen.

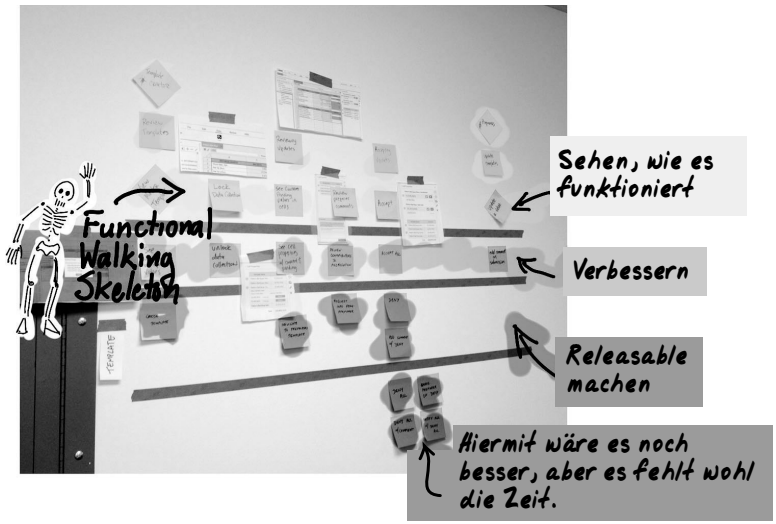
Deswegen hat Mike mit seinem Team einen Plan für die Entwicklung gemacht. Und das haben sie getan: Sie haben ihre Map in drei Teile zerlegt, indem sie Slices herausschnitten.

Der erste Slice schneidet einmal durch die gesamte Funktionalität. Wenn sie alle diese Teile fertig haben, werden sie sehen können, wie die Funktionalität von einem Ende zum anderen durchläuft. Das Produkt würde so noch nicht in allen Situationen funktionieren, in denen es das soll, und würden sie es so an den Kunden ausliefern, gäbe es wildes Geheule. Aber Mike und sein Team können erkennen, wie die Software von Anfang bis Ende läuft. Sie können echte Daten hineinwerfen, um zu sehen, wie gut die Anwendung sich schlägt, und automatisierte Testing-Tools benutzen, die ihnen sagen können, wie gut sie skaliert. Sie können eine Menge über die technischen Risiken herausfinden, mit denen sie später Schwierigkeiten haben könnten. Nun können sie im Projektfortschritt sicherer sein, dass sie alles rechtzeitig werden liefern können, oder wenigstens die unvorhersehbaren Probleme ausmachen, die später Zeit kosten würden. Diesen Slice bezeichne ich als *functional walking skeleton*, ein funktionierendes, laufendes Skelett – ein Begriff, den ich mir bei Alistair Cockburn

ausgeliehen habe; andere nennen es auch einen »Stahlfaden« oder »Leuchtspurmunition« (tracer bullet).

Mit dem zweiten Slice bauen die Teammitglieder die Funktionalität aus – um es näher an den Release zu bringen. Dabei lernen sie vermutlich Dinge, die sie nicht voraussagen konnten. Vielleicht haben sie ein paar Charakteristika übersehen, die das Feature haben sollte – feinere Details, die im Prototyp nicht erkundet wurden. Vielleicht finden sie auch heraus, dass das System nicht das leistet, was sie erwartet haben, und müssen zusätzliche Arbeit hineinstecken, um die Geschwindigkeit zu erzielen, die sie haben wollen. Das sind die »vorhersehbaren Unwägbarkeiten« – ein Konzept, das eng mit Donald Rumsfelds »unbekannten Unbekannten« verwandt ist. Tut nicht so, als gäbe es sie nicht – ihr wisst, dass es sie gibt.

Am Ende legen sie den dritten Slice aus, um das Feature genau auszuarbeiten und so weit wie möglich auf Hochglanz zu bringen. Und sie werden ein paar dieser unvorhersehbaren Sachen einbauen.



Macht nicht aus jedem Slice ein Release

Nicht jeder Slice ist auch ein Release für den Kunden oder die User: Es handelt sich um einen Meilenstein, der es dem Team ermöglicht, innezuhalten und eine Bestandsaufnahme zu vorzunehmen. Aus der

Sicht des Kunden oder Nutzers ist das unvollständig, also erspart euch die Peinlichkeit.

Das Team von Mike und Aaron veranschlagte rund zwei Monate Arbeit für dieses Feature. Genau wie Eric benutzten sie Zwei-Wochen-Sprints, also würden sie vier Sprints benötigen. Sie hätten auch vier Slices machen können, einen pro Sprint, aber das ist nicht ihre Art zu denken. Und ihr solltet auch nicht so denken. Stellt euch die Slices als drei verschiedene Blöcke vor, von denen jeder eigene Lernziele beinhaltet. Legt fest, in welchen Sprints oder Iterationen ihr diese abarbeiten wollt, wenn es so weit ist.

Das andere Geheminis gekonnter Schätzungen

Eine Tatsache, die hochgeheim zu sein scheint, es aber wirklich nicht sein sollte: Schätzungen sind ... etwas, das man schätzt. Werft eine Internetsuche an und sucht ein paar Oxymora (widersprüchliche Wortgruppen). Ich wette, ihr findet darunter auch diesen Ausdruck: *genaue Schätzung*. Wenn wir ganz genau wüssten, wie lang etwas braucht, dann würden wir nicht von *Schätzungen* sprechen, oder?

Aber wenn ihr kleine Stückchen Software geschrieben habt, wisst ihr zumindest eins mit Sicherheit: wie lange es dauerte, bis sie fertig waren. Das nennen wir eine *Messung*, und das ist dann schon ein wenig genauer.

Und hier kommt das nächste Geheimnis: Je öfter ihr solche Messungen anstellt, umso besser werdet ihr bei euren Prognosen. Wenn ihr jeden Tag zur Arbeit fahrt, könnt ihr sicherlich recht genau vorhersagen, wie lange ihr für die Strecke braucht. Wenn ich euch jetzt fragen würde, wie lange es dauert, zu einer andere Adresse in dieser Gegend zu gelangen, könnt ihr wahrscheinlich mit höchstens zehn Minuten Abweichung eine Vorhersage darüber treffen. Und genau so funktionieren Schätzungen.

Indem wir große Sachen in viele kleine zerlegen, erhalten wir mehr Gelegenheiten, Zeiten zu nehmen. Natürlich ist es ein bisschen komplizierter als das, aber allgemein gesprochen, könnt ihr bessere Vorhersagen treffen, je mehr Beispiele dafür ihr habt, wie lange es gedauert hat, etwas Ähnliches zu programmieren.

Als Produkt-Owner ist Mike letztlich dafür verantwortlich, dass sein Feature rechtzeitig fertig wird. Da er ein guter Produkt-Owner ist, hilft er jedem in seinem Team dabei, sich mit diesem Ziel zu identifizieren. Die frühen Schätzungen sind sein Abgabe-Budget.

Managt euer Budget

Mike und Aaron haben mit Leuten zusammengearbeitet, denen sie vertrauen, um ihre erste Schätzung des Zeitbedarfs abzugeben. Den Zeitbedarf betrachten sie als ein Budget. Und sie managen es aktiv.

Jedes Mal, wenn das Team einen kleinen Teil fertigstellt, messen sie, wie lange die Herstellung gedauert hat. Das betrachten sie als eine Ausgabe aus ihrem Zeitbudget. Sie könnten feststellen, dass sie schon die Hälfte ihres Budgets verbraucht, aber nur ein Drittel der Aufgaben erledigt haben. Wahrscheinlich haben sie das nicht erwartet, aber nun sind sie sich der Problematik bewusst und können etwas dagegen tun. Sie könnten zum Beispiel Budget von anderen Features leihen, an denen sie arbeiten, oder vielleicht können sie kleine Veränderungen vornehmen, die den Outcome für den User nicht substanziell verändern. Oder sie könnten sich der Sache stellen und ihr Bestes tun, die Ewartungen der Leute, an die sie liefern müssen, zu verändern.

Je nachdem, wie schlimm es ist, brauchen sie alle womöglich mehr Bier.

Wenn man eine Entwicklungsstrategie festlegt, versucht man vor allem, das in den Griff zu bekommen, was dem Budget am meisten schaden kann, und zwar so früh wie möglich. Das sind die riskanten Sachen. Und im Austausch mit dem ganzen Team hat man die Chance, diese Dinge frühzeitig zu erkennen.

Risiken in der Story Map offenlegen

Chris Shinkle, SEP

Eine große Sicherheitsfirma wollte ein mittelpreisiges kabelloses Zugangssystem für Gebäude mittlerer Größe haben (z.B. Schulen, Arztpraxen, Ladengeschäfte etc.). Man zog SEP hinzu, um sowohl die Firmware mitsamt Schließsystemen als auch das kabellose ZigBee-Gateway zu erstellen, mit dem alles kommunizieren sollte.

Technisch war die Sache sehr aufregend, aber auch mit dem Potenzial, im großen Umfang zu scheitern – ein sehr knappes Budget,

eine enge Timeline, Wechsel von Führungskräften mitten im Projekt, ungetestete Technik und ein sich massiv aufblähender Umfang.

Natürlich ging die Sache schnell baden. Das Projektteam verpasste mehrere Meilensteine. Der Kunde war unzufrieden, und die Stimmung im Team entsprechend. Bei einer rückwirkenden Analyse stellte das Team fest, dass eine der entscheidenden Ursachen für den Misserfolg nicht geplante Aufgaben waren, hauptsächlich durch Unsicherheiten und eingetretene Risiken. Etwas musste sich ändern.

Wie jede Gruppe von klugen Ingenieuren ging das Team dieses Problem direkt an. Die Lösung? Die Story Map anzupassen.

Auf der obersten Ebene erhöhten sie die Frequenz und Genauigkeit ihrer Map. Durch die erhöhte Frequenz des Story Mapping bei jedem Zwischenrelease hofften sie, die Wahrscheinlichkeit zu erhöhen, dass sie weitere Risiken identifizieren konnten. Mit der Erhöhung der Genauigkeit der Map und dem Einbinden von »Risk Stories« (zusätzlich zu normalen Aktivitäten, Tasks und Details) wollten sie Risiken besser visualisieren, diskutieren und schlussendlich managen.

Die Ergebnisse waren verblüffend.

Das Team wusste, dass Breite und Tiefe einer typischen Story Map ein Gefühl für den Umfang des Projektes vermitteln. Es wusste auch, dass die Zahl der möglichen Pfade durch die Map einen guten Indikator der Komplexität darstellt. Aber da Unsicherheiten und Risiken in der ursprünglichen Map nicht berücksichtigt worden

waren, zeigte die Map auch nicht den tatsächlichen Umfang der Arbeiten (einschließlich Lernen), der erledigt werden musste.

Die neue Map mit ihren Risk Stories war eine bessere Abbildung der Größe und Komplexität des Weges, der vor ihnen lag. Projektgröße und Komplexität wurden besser abgebildet, weil die Map sowohl aus den ursprünglichen Stories als auch aus den neuen, »unbekannten« Stories zusammengesetzt war, und auch dem Wissen, dass das Team noch sammeln musste, um sicher mit den vorhandenen Stories arbeiten zu können.

Wie zu erwarten, wurde die Map damit als Planungsinstrument viel nützlicher. Sie zeigte nun Unsicherheiten und Risiken auf, die das Team zusätzliche Zeit kosten würden. Dadurch, dass sie diese Zeit nun planen konnten, wurde das Team deutlich berechenbarer und zuverlässiger.

Zu den positiven Nebenwirkungen gehörte, dass es einen besseren Weg besaß, Stakeholder in den Lernprozess einzubinden. Neben dem traditionellen Feature-Burn-down-Chart gab es nun auch einen Burn-Down-Chart der Risiken. Es war sehr hilfreich, dem Kunden die Visualisierung der Risiken zeigen zu können, wenn der Feature-Burn-down gerade nicht gut aussah.

Die Lektion aus dieser Sache: Fügt man seinen Maps Risk Stories hinzu und erhöht die Frequenz des Story Mapping, kann die Map besser die Realität abbilden.

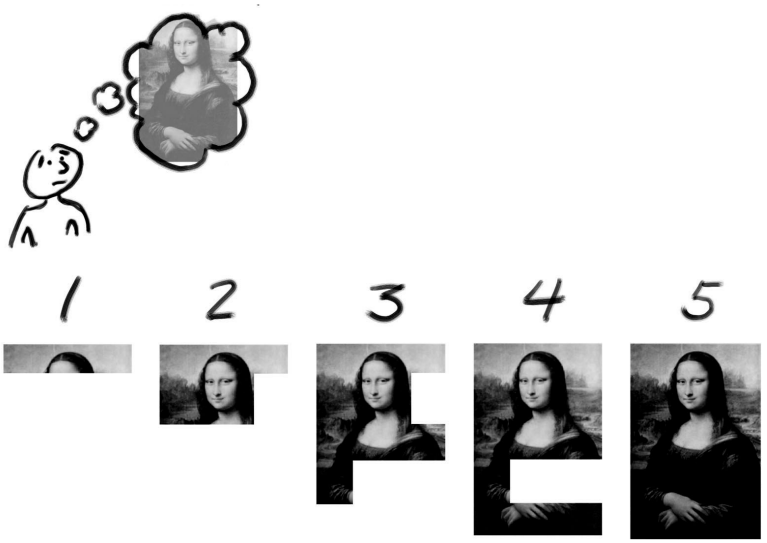
Was würde da Vinci tun?

Ich frage mich das oft. Naja, eigentlich nicht. Aber vielleicht sollte ich es tun.

Mike und Aaron benutzten hier eine Strategie, die von Künstlern eingesetzt wird, um etwas rechtzeitig fertigzustellen. Ich benutze sie schon seit Jahren im Zusammenhang mit Software. Und als ich meine Freunde bei Globo.com kennenlernte, stellte ich fest, dass auch sie sie einsetzen, denn – wie ich bereits sagte – wenn sie ihr cooles interaktives Zeug für die Olympischen Spiele nicht rechtzeitig fertig haben, wird das Olympische Kommittee kaum die Spiele für sie verschieben. Vermutlich benutzt ihr den Ansatz auch regelmäßig, ohne euch dessen bewusst zu sein.

Lasst mich zunächst erklären, was da Vinci *nicht* tun würde. Unglücklicherweise ist das sehr oft das, was Leute in der Softwareentwicklung zu tun versuchen.

Nehmen wir an, ihr wärt da Vinci und wolltet ein Gemälde anfertigen, und zwar so, wie es ein naives Team von Softwareentwicklern machen würde.[1] Ihr beginnt mit dem, wovon ihr annehmt, es sei eine klare Vision des fertigen Gemäldes. Dann zerlegt ihr das Gemälde in seine Teile. Sagen wir, ihr habt fünf Tage, das Gemälde anzufertigen. Jeden Tag malt ihr weitere Bestandteile des Bildes. Am Ende von Tag 5 habt ihr – Hurra! – ein fertiges Gemälde. Was könnte einfacher sein?



Nur ... so funktioniert es eben nicht. Zumindest nicht in der Kunst. Diese Art des Arbeitens geht davon aus, dass unsere Vorstellung des fertigen Produkts richtig und akkurat ist. Sie beinhaltet auch Annahmen über die Fähigkeiten der Schöpfer und deren Fähigkeit, Teile präzise zu definieren, ohne sie im Gesamtzusammenhang zu sehen. In der Softwarewelt nennen wir das eine *inkrementelle Strategie*. Auf diese Art baut ein Maurer eine Mauer. Und es funktioniert, solange

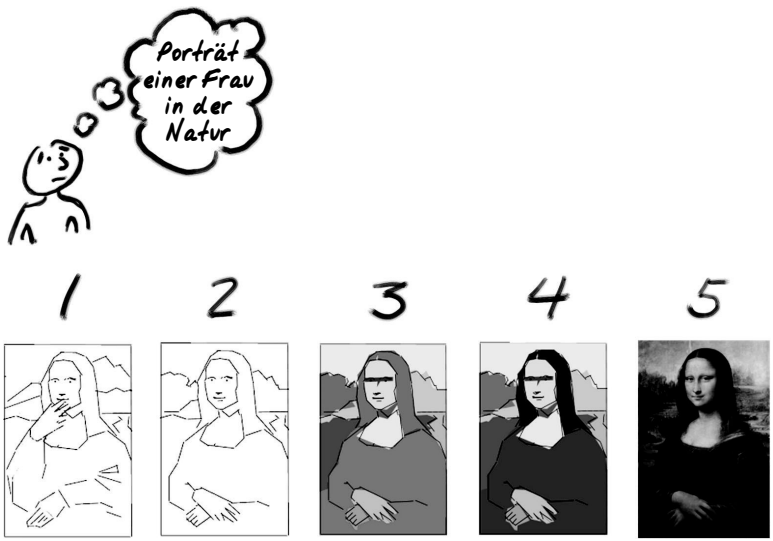
1 Die Idee für diese einfache Visluaisierung habe ich aus John Armitages Artikel »Are Agile Methods Good for Design?« aus dem Jahr 2004. John beschreibt darin, wie man sich damit an User Experience Design annähern kann. Ich denke, wir sollten die Methapher auch auf unsere Methode anwenden, Builds zu erstellen.

jedes Teilchen so gleichmäßig groß und genau definiert ist wie ein Backstein.

Als Kind hatte ich dieses Problem häufig beim Malen. Ich begann etwa, ein Tier zu zeichnen, und fing am Kopf an, und wenn ich den perfekt hinbekam, war der Rest des Körpers dran – Beine, Schwanz und so weiter. Wenn ich dann fast fertig war, konnte ich sehen, dass die Proportionen des Tieres nicht stimmig waren. Der Kopf war zu groß oder zu klein für den Rest des Körpers. Die Beine knickten in eigenartigen Winkeln ab. Und die ganze Pose des Tieres wirkte steif. Zumindest sah ich das als talentierter sechsjähriger Künstler so. Und alle Sechsjährigen sind talentierte Künstler.

Später lernte ich, dass es besser funktionierte, wenn ich zunächst eine Skizze der Gesamtkomposition anfertigte. Damit konnte ich die Proportionen optimieren und auch Veränderungen an der Haltung des Tieres vornehmen. Ich konnte mir sogar überlegen, doch etwas anderes zu malen.

Ich weiß nicht, wie da Vinci vorging – aber ich vermute, ganz ähnlich.



Auch ein da Vinci hätte vermutlich eingestanden, dass seine Vision nicht perfekt war und er beim Erschaffen des Gemäldes noch etwas dazulernen würde. Ich denke mir, am ersten Tag hätte er die Komposition als Skizze umrissen oder eine dünne Untermalung ange-

fertigt. An diesem Punkt hätte er vermutlich bereits die ersten Veränderungen an der Komposition vorgenommen. »Hey, das Lächeln wird sicher ein wichtiger Bestandteil der Komposition. Ich nehme die Hand mal vom Mund weg. Und diese Berge da im Hintergrund sind zu viel des Guten.«

In der Mitte der Woche ist da Vinci damit beschäftigt, jede Menge Farben und Formen zu seinem Bild hinzuzufügen, und er nimmt immer noch direkt im Arbeitsprozess Veränderungen vor. Am Ende der Woche wird die Zeit knapp, also konzentriert er sich darauf, das Ding zu verfeinern. Man kann sich fragen, ob die fehlenden Augenbrauen der *Mona Lisa* ein bewusstes Gestaltungsmerkmal sind oder da Vinci einfach die Zeit fehlte, noch ein fein ausgefeiltes Feature hinzuzufügen.

Ein Kunstwerk wird niemals fertig, höchstens aufgegeben. – Leonardo da Vinci

Das Zitat wird da Vinci zugeschrieben, und es drückt aus, dass wir ad infinitum Dinge verbessern und hinzufügen könnten, aber irgendwann das Produkt abliefern müssen. Und wenn wir die Arbeiten von da Vinci und anderen Künstlern als Beispiel nehmen, können wir sagen, dass wir, die wir die Kunstwerke bewundern, nicht sagen können, dass sie vorzeitig abgebrochen wurden. Für uns sehen sie fertiggestellt aus.

Iterativ UND inkrementell

Künstler und Autoren arbeiten so. Auch die Leute, die die Morgenzeitung oder die Abendnachrichten zusammenstellen, arbeiten so. Leute, die Theateraufführungen produzieren, arbeiten so. Jeder, der irgendwas mit einer Deadline abliefern muss und beim Arbeiten Neues hinzulernt, kennt diese Vorgehensweise.

Benutzt iteratives Denken zur Evaluation und überarbeitet das, was ihr bereits hergestellt habt.

In der Softwarewelt hat *iterieren* zwei Bedeutungen. Aus der Perspektive des Prozesses bedeutet es, dasselbe immer und immer wieder zu tun; deswegen bezeichnen wir die Entwicklungs-Zeitbox in der agilen Entwicklung auch oft als *Iteration*. Wenn ihr das Wort aber benutzt, um zu beschreiben, was ihr mit der Software tut, die ihr gerade entwickelt, bezeichnet der Begriff das Überprüfen und Ver-

ändern bzw. Verbessern. Software nachträglich bearbeiten zu müssen, sehen viele als Fehlschlag an; da werden dann Vorwürfe wie »lausige Anforderungen« oder »scope creep« in den Raum geworfen als Rüge für diejenigen, die die Entscheidungen für den Build treffen mussten. Aber wir wissen alle, dass Veränderungen eine notwendige Folge des Erkenntnisprozesses sind.

Benutzt inkrementelles Denken, um Ergänzungen vorzunehmen.

Es ist einfach, im Mahlstrohm der Iteration verloren zu gehen. Also müssen wir den Kalender im Blick behalten und inkrementell neue Dinge zu unserem Produkt hinzufügen. Künstler erweitern ihre Gemälde nicht nur, indem sie neue Sachen hinzufügen, sondern auch, indem sie Sachen, die bereits da sind, weiter ausbauen.

Ihr könnt in der Softwareentwicklung ähnlich vorgehen, indem ihr zunächst eine ganz einfache Version der Funktionalitäten erschafft, ohne die Extras. Betrachtet diese als eure Skizze. Nachdem ihr die simple Version ausprobiert habt, könnt ihr sie ausbauen und mehr Funktionalität ergänzen. Nach und nach wächst so das Ganze zu der fertigen Vision heran, die ihr euch ursprünglich mal vorgestellt habt. Wenn alles wirklich gut läuft, wird es etwas ganz anderes als das, was ihr ursprünglich dachtet – nämlich etwas Besseres, das dem zu verdanken ist, was ihr dazugelernt habt.

Strategien: Eröffnungs-, Mittel- und Endspiel

Es könnte euch ein paar Kopfschmerzen verursachen, aber ich werde hier ein paar Metaphern vermischen. In der Softwareentwicklung benutze ich persönlich eine Strategie, die an eine Metapher aus dem Schachspiel angelehnt ist. Ich bin ein lausiger Schachspieler, der nicht viel Ahnung davon hat – wenn ich die Metapher hier also verkorkse, braucht ihr mir nicht extra zu schreiben, um mich zu korrigieren. Egal, wie klein ein Produkt- oder Feature-Release auch ist, ich unterteile mein Release-Backlog stets in drei Gruppen:

Eröffnungsspiel
: Konzentriert euch auf die grundlegenden Features oder Userschritte, die sich durch das gesamte Produkt ziehen. Konzentriert euch auf Sachen, die technisch herausfordernd oder risikobehaftet sind. Überspringt optionale Dinge, die User tun könnten. Überspringt die komplizierten Geschäftsregeln, die ihr vor dem

Release noch braucht. Produziert *gerade eben* genug, um das Produkt von Anfang bis Ende laufen zu sehen.

Mittelspiel
: Ergänzt und erweitert Features. Fügt Dinge hinzu, die optionale Schritte der User ermöglichen. Implementiert diese leidigen Geschäftsregeln. Wenn ihr euer Eröffnungsspiel richtig gemacht habt, könnt ihr nun euer Produkt durchgehend auf Dinge wie Performance, Skalierbarkeit und Benutzbarkeit hin testen. Hier verstecken sich mögliche qualitative Mängel, die man schwer erfassen kann. Wir müssen uns dessen bewusst sein und sie regelmäßig überprüfen.

Endspiel
: Veredelt euren Release. Macht ihn sexier, und effizienter in der Benutzung. Da ihr die Software nun mit echten Daten und maßstabsgerecht einsetzen könnt, werdet ihr noch Verbesserungsmöglichkeiten entdecken, die ihr anhand des Prototyps nicht sehen konntet. Nun habt ihr auch Feedback von Usern, das ihr umsetzen könnt.

Markiert eure Entwicklungsstrategie in einer Map.

Wenn ihr etwas habt, von dem ihr denkt, dass es ein praktikabler erster Release für Kunden und User sein könnte, solltet ihr im Team euer erstes öffentliches Release ein weiteres Mal in Slices aufteilen, in Eröffnungs-, Mittel- und Endspiel-Stories. Das Team, welches das Produkt entwickelt, ist am besten dafür geeignet, Risiken und Lernmöglichkeiten zu identifizieren. Die Teammitglieder werden sich am stärksten mit einem Plan identifizieren, den sie gemeinsam erstellt haben.

Das ist das, was Aaron und Mike mithilfe ihres gesamten Entwicklerteams taten. Seht euch noch mal an, wie glücklich sie sind.

Es geht um das Risiko.

In Kapitel 3 musste sich Eric dem Risiko stellen, eventuell das falsche Produkt zu identifizieren. Seine Release-Strategie mit Slices ermöglichte es ihm, den Kunden vollständige Produkte vorzulegen.

In diesem Kapitel haben Aaron und Mike sich auf technische Risiken konzentriert, auf Sachen, die ihren Lieferplan zum Platzen bringen oder mehr Kosten verursachen können als erwartet. Sie zeigen das, was sie am Ende jedes Durchlaufs haben, nicht ihren Kunden oder Usern, denn sie wissen bereits, dass es nicht ausreicht. Aber sie schauen es sich gemeinsam mit ihrem Team sehr genau an und benutzen das, was sie daraus lernen, um die Entwicklung ihres Features sicher zu steuern.

Es wurde nur kurz erwähnt, aber vielleicht erinnert ihr euch, dass Eric in Kapitel 2 zwei Zwei-Wochen-Sprints veranschlagte, in denen er sein nächstes MVPe, das nächste Experiment, herstellen wollte. Er musste entscheiden, was im ersten und was im zweiten Sprint gemacht werden sollte. Er traf seine Entscheidungen mit einer Strategie wie der hier vorgestellten. Er und sein Team machten sich zuerst an die riskanten Teile – die Dinge, von denen sie rechtzeitig wissen wollten, ob sie funktionierten, so dass sie notfalls ihren Kurs hätten korrigieren können, ehe sie damit an den Kunden herantraten.

Wie geht es weiter?

Ich habe euch vier ziemlich gute Beispiele dafür vorgestellt, wie Maps für unterschiedliche Zwecke erstellt und eingesetzt werden. Es gibt noch viel mehr Arten, Maps zu benutzen, und dazu kommen wir in späteren Kapiteln. Aber ehe wir zu weit abschweifen, will ich euch noch meinen Lieblingstrick dafür verraten, wie man anderen Leuten beibringt, zu mappen. Wenn ihr das ausprobiert, werdet ihr wie die Experten mappen, versprochen!

Wie es geht, zeige ich euch in Kapitel 5.