20 Jahre Agiles Manifest – definitiv den Kinderschuhen entwachsen

Veröffentlicht von
Jutta Eckstein
Geschätzte Lesezeit
7 min
Hinzugefügt am
2021-12-08
Adresse
https://www.heise.de/hintergrund/20-Jahre-Agiles-Manifest-definitiv-den-Kinderschuhen-entwachsen-5049684.html?view=print

20 Jahre Agiles Manifest – definitiv den Kinderschuhen entwachsen

20 Jahre Agiles Manifest – definitiv den Kinderschuhen entwachsen

Jutta Eckstein

(Bild: Ward Cunningham)

Was vor 20 Jahren für Softwareentwickler gedacht war, ist mittlerweile unternehmens- und branchenübergreifend von Bedeutung. Eine Würdigung von Jutta Eckstein.

Am heutigen Tag feiert das Agile Manifest seinen zwanzigsten Geburtstag. Erstaunlicherweise gilt selbst nach all der Zeit noch immer, dass "agil" für viele Unternehmen gleichbedeutend mit "neu" und "trendy" ist. Die Pandemie hat gerade erst die Aktualität der Agilität aufgezeigt. Langfristige Pläne, die 2019 erstellt wurden, hatten bereits im März 2020 ihre Gültigkeit verloren. Das heißt, die Idee, Veränderungen anzunehmen und basierend auf Ergebnissen und der aktuellen Situation ständig dazuzulernen, wird zunehmend essenzieller.

Tops & Flops

In einer **nicht repräsentativen Umfrage auf Twitter [1]** stellte sich heraus, dass das Agile Manifest erstens immer noch von Bedeutung ist und es zweitens Prinzipien gibt, die favorisiert werden. Die folgenden Prinzipien gehören laut der Umfrage zu den Top 3:

- Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.
- Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen, und vertraue darauf, dass sie die Aufgabe erledigen.

- In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann, und passt sein Verhalten entsprechend an.

Im gleichen Zuge wurden ein paar Prinzipien genannt, die als besonders schwierig umzusetzen oder auch als nur schwer verständlich gelten. Diese sind:

- Fachexperten und Entwickler müssen während des Projekts täglich zusammenarbeiten.
- Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.

Up to date oder Update?

In regelmäßigen Abständen taucht die Frage auf, ob und wann das Agile Manifest aktualisiert werden sollte. Dazu gibt es eine einfache Antwort: nie. Im Gegensatz zu Scrum, das lediglich zwei Autoren entwickelt haben und von daher immer wieder, so auch 2020, aktualisiert wird, liegt das Urheberrecht des Agilen Manifests bei der Gruppe der Autoren. Und diese wird in dieser Konstellation nicht mehr zusammenkommen. Das liegt einerseits daran, dass die Autorengemeinschaft (und damit auch die agile Community) leider den schweren Verlust von Mike Beedle zu beklagen hatte, der 2018 in Chicago erstochen wurde. Andererseits gibt es teils schwere Verwerfungen unter den Autoren, allen voran zwischen Robert Martin und einigen anderen Manifest-Begründern, und zwar weil sich "Uncle Bob" wiederholt rassistisch und frauenfeindlich geäußert hat. Insofern ist es unmöglich, dass sich die Autorengemeinschaft nochmals auf etwas Gemeinsames verständigen wird.

Reichlich Alternativen und doch unersetzbar

In den letzten Jahren sind diverse andere Manifeste entstanden. Die einen versuchen, das Agile Manifest für Domänen zu definieren, die außerhalb der Softwareentwicklung liegen, etwa das **Agile Marketing Manifest [2]** oder das **Agile HR Manifest [3]**.

Die anderen sind der Versuch, das Agile Manifest zu aktualisieren. Dazu gehören **Agile 2 [4]**, **Heart of Agile [5]** oder **Modern Agile [6]**. Ersteres ist nicht viel mehr als eine gute Reflexion des Agilen Manifests. Bei anderen Aktualisierungen wie bei den letzten beiden entsteht jeweils eine begeisterte Sub-Community. Grundsätzlich aber gilt, dass alle alternativen Manifeste die Anzahl der agilen Manifeste erweitern, aber nicht das ursprüngliche Manifest ersetzen.

Darüber hinaus gibt es unterschiedliche Entwicklungen wie **Working out Loud [7]** oder **New Work [8]**, die sich auf einen Aspekt der Agilität fokussieren und ihn in den Mittelpunkt stellen. Auch diese Entwicklung zeigt, dass die agile Idee trotz der 20 Jahre immer noch aktuell ist.

Neues Verständnis

Auch wenn sich das Manifest von der Definition her nicht verändert hat, hat sich dennoch die Anwendung dieses Verständnisses der Agilität im Laufe der Zeit verändert. Einerseits handelt es sich dabei um kleinere Veränderungen, die eh zu konkret sind, als dass sie im Manifest stehen würden, und andererseits sind sie dem Erfolg der Agilität geschuldet, die unter anderem zur Erweiterung auf andere Anwendungsbereiche führte. Meiner Erfahrung nach erfordern die Veränderungen noch mehr Verantwortungsübernahme aller Beteiligter und damit ein noch tieferes Verständnis der Agilität. Nachfolgend ein paar Beispiele dieser Veränderungen:

- Es wird heute davon ausgegangen, dass es keine Projekte gibt, da selbst die Unterfangen, die als Projekte starten, weiterentwickelt und gewartet werden. Von daher sollte man all diese Unterfangen ehrlicherweise als Produktentwicklungen betrachten, behandeln und bezeichnen.
- Es wird kein Mehrwert mehr im Schätzen gesehen. Wenn Teams darauf achten, dass Stories immer ungefähr gleich groß geschnitten sind, wird es unnötig, jede einzelne Story zu schätzen.
- Stories wurden lange Zeit nur als solche erkannt oder akzeptiert, wenn sie dem Connextra-Format entsprechen ("Als ein ... möchte ich ..., sodass ..."). Vielmehr beruft man sich jetzt wieder darauf, dass Stories lediglich als eine Art Merkzettel für ein Gespräch über den Inhalt der Story dienen (und für Gesprächsnotizen gibt es kein Format).
- Flow gewinnt zunehmend an Bedeutung. Dazu gehört, dass oftmals die Sprints beziehungsweise Iterationen keine allzu große Rolle mehr spielen. Das heißt, dass Teams nicht alle zwei Wochen mehr planen, sondern eher im Kanban-Modus täglich abstimmen, was sich fertigstellen lässt und was dann aus dem Backlog nachrückt. Das heißt, es wird eher kontinuierlich geplant, und genauso werden auch die Retrospektiven oftmals zu einer kontinuierlichen Reflexion und Korrektur.
- Vor allem während des Höhepunkts der Black-Lives-Matter-Demonstrationen wurde gefordert, die Rolle des Scrum *Master* umzubenennen (da der Begriff "Master" historisch zur Abgrenzung zum Slave bzw. Sklaven diente). Die Anregung hat die Scrum-Community jedoch ausgesessen. (Im Gegensatz zu GitHub, wo im Zuge der Diskussion der *Master* in *Main* umbenannt wurde.)
- Agile Entwicklung ohne DevOps ist kaum mehr vorstellbar. Allerdings verstehen viele die zugrunde liegende Idee von DevOps nicht immer. Das wird daran ersichtlich, dass mehr und mehr Unternehmen ein DevOps-Team neben die Entwicklungsteams stellen oder auch die Rolle eines DevOps-Ingenieurs besetzen. Dabei übersehen sie, dass DevOps eine Haltung, eine Kultur ist (nämlich dass Dev und Ops eng verzahnt zusammenarbeiten) und keine Funktion, Rolle oder spezielle Aufgabe.

Die wenigsten "leichtgewichtigen" Methoden, die die Grundlage für das Manifest waren, spielen heute noch eine Rolle. Dazu gehören beispielsweise die Crystal-Methoden oder auch die adaptive Softwareentwicklung. Dafür hat Extreme Programming (XP) in den letzten Jahren einen Aufschwung erfahren unter anderem aus dem Grund, weil sich Kent Beck wieder in der Community zurückgemeldet hat, nachdem er viele Jahre aus persönlichen Gründen nicht mehr in Erscheinung getreten war. Bei einem seiner ersten öffentlichen Auftritte auf der XP 2018 in Porto meldete er sich bezeichnenderweise wie folgt zurück: "I'm Kent and I'm Back." Die neu aufgeflammte Begeisterung für XP führt dazu, dass zunehmend die Meinung vorherrscht, dass nur die Verbindung von Scrum und XP wirklich zum Erfolg führe.

Was kommt?

Agilität wird vielfach nicht mehr nur im Rahmen der Softwareentwicklung verstanden, auch wenn das Manifest eindeutig genau den Fokus hatte. Heute ist man davon überzeugt, dass die agilen Prinzipien auch unternehmensweit von Vorteil sind. So ist speziell Business Agility ein Thema, das mehr und mehr Momentum erfährt.

Für die Zukunft muss sich das Agile Manifest beweisen, inwiefern es auch für Diversifizierung und Nachhaltigkeit tragfähig ist. Auf die Frage, was wohl der Unterschied der Werte und Prinzipien wäre, wenn die Begründer des Manifests eine diverse Gruppe gewesen wäre, **meinte Ron Jeffries, einer der Co-Autoren des Manifests [9]**:

"Schwer zu sagen, insbesondere wenn man sich auf vier Werte beschränken muss. Ich denke, der Fokus hätte stärker auf den Menschen und auf Testen gelegen. Man glaubt, dass das Manifest besser geworden wäre, aber vielleicht wäre es auch gar nicht entstanden. Wäre es überhaupt möglich gewesen, dass eine diverse Gruppe sich trifft? Wollen wir es hoffen ..."

Andere Stimmen unterstreichen den Punkt, dass das Manifest vermutlich weniger entwicklerzentriert gewesen wäre und mehr den Menschen holistisch gesehen hätte, inklusive der Gefahren von Burnout und des Achtens auf mentale Gesundheit. Bei aller Spekulation darf man nicht außer Acht lassen, dass vor 20 Jahren andere Themen aktuell waren als heute. Von daher lässt sich aus heutiger Perspektive über den Einfluss einer größeren Diversifizierung nur schlecht mutmaßen.

Glückwünsche willkommen!

Wann haben Sie Ihre ersten Erfahrungen mit agilem Softwareentwicklung oder agilem Projektmanagement gemacht? Waren sie gut, was gestaltete sich schwierig. Wir freuen uns über Ihren Kommentar im Forum zu diesem Artikel. Gerne können Sie auch die kontaktieren.

Was die Nachhaltigkeit angeht, so steckt dieses Thema bezogen auf agile Entwicklung noch in den Anfängen. Es bleibt zu beobachten, inwiefern die Betrachtung des ökologischen Fußabdrucks und der verwendeten Energiequellen einer Software in die agile Entwicklung Einzug halten werden.

Jutta Eckstein
arbeitet als Business-Coach, Change-Managerin, Beraterin und Trainerin im In- und Ausland. Weltweit verfügt sie über große Erfahrung bei der erfolgreichen Umsetzung agiler Prozesse in mittleren bis großen, verteilten, unternehmenskritischen Projekten.

Veranstaltungen im Rahmen des Jubiläums

Es gibt eine Reihe von Veranstaltungen, die in diesem Monat das 20-jährigen Jubiläums zum Anlass nehmen, um über die Aktualität und Auswirkung des Agilen Manifests zu reflektieren:

- **Agile 20 Reflect Festival [11]**
- **Agile Manifesto 20th Anniversary [12]**
- **Agile 20 Conference [13]**

()

[6] https://modernagile.org/

[7] https://workingoutloud.com/

[8] https://en.wikipedia.org/wiki/New_Work

[9] https://twitter.com/RonJeffries/status/1354427259042615297

[10] mailto:developer@heise.de?subject=20%20Jahre%20Agiles%20Manifest

[11] https://www.agilealliance.org/celebrating-20-years-of-agility-at-the-agile20reflect-festival/

[12] https://www.scrumalliance.org/events/agile-manifesto-20th-anniversary

[13] https://a20dmv.org/

[14] mailto:ane@heise.de

Agile at 20: The Failed Rebellion

Veröffentlicht von
Unbekannt
Geschätzte Lesezeit
10 min
Hinzugefügt am
2021-12-08
Adresse
https://www.simplethread.com/agile-at-20-the-failed-rebellion/

Agile at 20: The Failed Rebellion

Worried you'll miss us?

Subscribe to get our articles and updates in your inbox.

The Agile Manifesto turned 20 this year, and there are two facts that seem self-evident:

1. Agile, as a label, won; nobody wants to be called *non-Agile*.

2. Agile, as it is practiced, falls woefully short of the revolutionary ideas of its founders.

How did we get to this point? **Everybody says they do Agile and yet almost nobody is Agile.**

Whence The Manifesto

In February, 2001, a group of seventeen expert software practitioners met at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah. Over the course of a few days of discussion and debate, they collaboratively wrote the “Agile Software Development Manifesto”.

* I don’t know the personal history of every signatory, but of those I know, they were all either still actively writing code or had a long and recent history of it.

The first point to highlight is that these were practitioners. They weren’t project managers or CTOs or VP Engs. They were developers, programmers, scientists, and engineers. They were still slingin’ code* and working with their stakeholders to solve problems. This is important.

The other point is that the Agile Manifesto wasn’t created in a vacuum. Many of these people already had a methodology they had created and/or were proselytizing. I might have my timing slightly off, but I think all of these methodologies pre-existed “capital A Agile”: Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming. I know Schwaber and Sutherland spoke publicly about Scrum in 1995, and Beck and Jeffries started talking about Extreme Programming (XP) in 1996, I think.

Everyone in this group had deep experience writing software, and they were all looking for an alternative to documentation-driven, heavyweight software development processes that ruled the day. The heart of the manifesto was four statements of value:

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

A New Hope

From our vantage point in 2021, it’s easy to take so much of modern software development practice for granted, but in 2001, these ideas were ***wildly radical.***

What do you mean you're going to start building the software before you've gathered all of the requirements and estimated every piece of functionality? That's insane!

The important piece that gets forgotten is that Agile was openly, militantly anti-management in the beginning. For example, Ken Schwaber was vocal and explicit about his goal to get rid of all project managers – not just get the people off his projects, eradicate the profession from our industry.

Agility & PMI

> We have found that the role of the project manager is counterproductive in complex, creative work. The project manager's thinking, as represented by the project plan, constrains the creativity and intelligence of everyone else on the project to that of the plan, rather than engaging everyone's intelligence to best solve the problems.
>
> **Ken Schwaber, manifesto signatory and Scrum co-creator**

Scrum Masters had almost no authority, no votes on issues. They were servant leaders, helping shield and unblock the team but not managing the team. XP was similar. If I recall correctly, XP originally had trackers and coaches, which had a similar facilitating, supportive vibe.

Alistair Cockburn, manifesto signatory and creator of the Crystal methodology and hexagonal architecture, had a marvelous, insightful thread on this recently, including this idea (paraphrased):

> Scrum struck a magnificent bargain in hostile territory:
>
> - *Management got 12 times per year to change direction in any way they wanted, after each sprint.*
> - *Teams got an entire month of total quiet time with no interruptions or changes of direction to do heavy thinking and working.*
> - *Teams got to announce what they could and couldn't do in the month, with no management interference in their bid.*
>
> No executives ever got a better deal.
> No development team ever got a better deal.

I'm a certified Scrum Master, working on Agile teams for 15+ years, and I've read many of the popular books in the space. I've never seen anyone frame the idea so explicitly and succinctly (again paraphrasing Cockburn):

> Scrum was invented to function in hostile environments. It's a contract between hard-pushing managers and developers needing time to think and explore.

The Empire Strikes Back

In some ways, Agile was a grassroots labor movement. It certainly started with the practitioners on the ground and got pushed upwards into management. How did this ever succeed?

It's partially due to developers growing in number and value to their businesses, gaining clout. But the biggest factor, in my view, is that the traditional waterfall approach simply wasn't working. As software got more complicated and the pace of business accelerated and the sophistication of users rose, trying to plan everything up front became impossible. Embracing iterative development was logical, if a bit scary for managers used to planning everything.

I remember meetings in the mid-2000s where you could tell management wasn't really buying it, but they were out of ideas.

What the hell, let's try this crazy idea the engineers keep talking about. We're not hitting deadlines now. How much worse can it get?

Then to their surprise, it started working, kind of, in fits and starts. Teams would thrash for a while and then slowly gain their legs, discovering what patterns worked for that individual team, picking up momentum. After a few sprints, you'd start to see the real power of prioritizing working software, collaboration, taking time to inspect and adapt, and all the rest.

In the course of about 5 years, Agile went from a methodology that *you'd heard of but probably not used* to something **everybody did**. In 2005, I was switching jobs, and I remember the fact that I knew a bit about Agile and TDD was a real differentiator. By 2010, it was assumed modern software teams were doing some flavor of Agile. At least, this was true for my bubble in the consulting world; large enterprises always move slower.

??? We did it! We won! Congratulations everybody! ???

And that's the end of the story. You can go ahead and close the browser tab.?

> Winning was easy, young man. Governing's harder.
>
> **George Washington, as portrayed in Hamilton**

Unfortunately, like many revolutions, the history of Agile didn't unfold how the founders envisioned.

- It turns out that prioritizing individuals and interactions is a hard concept to sell. It's much easier to sell processes and tools.
- It turns out that working software is harder to produce than unrealistic plans and reams of documentation.
- It turns out that collaborating with customers requires trust and vulnerability, not always present in a business setting.
- It turns out that responding to change often gets outweighed by executives wanting to feel in control and still legitimately needing to make long-term plans for their businesses.

It turns out that Agile done poorly often feels like chaos.

That doesn't mean the four values are wrong. It just means this whole thing takes some effort to get right, some courage to accept that software is inherently messy and chaotic at times. You have to understand and believe that if you keep learning and adapting and improving **and shipping**, you'll eventually get to a much better place, a much more honest and realistic **and productive** place than you ever could with waterfall.

> The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term hacker.
>
> **Jim Highsmith, History: The Agile Manifesto**

Those are important points. We still need to plan and document and have rigor in Agile. **It's about balance.** However if your organization is struggling with an Agile transformation, drowning in chaos, you're going to leap when someone offers you a lifeboat in the form of certifications and processes and tools. Executives understand processes and tools much more than they grok self-organizing teams.

Return of the Rogue One?

This is where my three-act structure breaks down a bit, because unfortunately, I don't see the plucky rebels coming back on this one, at least not under the Agile label.

It's been overrun with tool vendors and process consultants and experts making promises that can never be delivered. This is how we've ended up with SAFe and Scaled Scrum and all of the other enterprise Agile flavors. These frameworks weren't created with malicious intent, and they probably even have some value in the right context. But I wouldn't call them Agile. Trying to scale a methodology that focuses on individuals and interactions will inevitably lead to problems – and erode the original value of the methodology.

This is how we've ended up with this famous 2018 piece by Ron Jeffries, manifesto signatory and XP co-creator.

Developers Should Abandon Agile

> When "Agile" ideas are applied poorly, they often lead to more interference with developers, less time to do the work, higher pressure, and demands to "go faster". This is bad for the developers, and, ultimately, bad for the enterprise as well, because doing "Agile" poorly will result, more often than not, in far more defects and much slower progress than could be attained. Often, good developers leave such organizations, resulting in a less effective

> enterprise than prior to installing “Agile”.

This is how we’ve ended up with the famous 2014 piece by Dave Thomas, manifesto signatory and Pragmatic Programming co-creator.

Agile is Dead (Long Live Agility)

> The word “agile” has been subverted to the point where it is effectively meaningless, and what passes for an agile community seems to be largely an arena for consultants and vendors to hawk services and products… Once the Manifesto became popular, the word agile became a magnet for anyone with points to espouse, hours to bill, or products to sell. It became a marketing term.
>
> So I think it is time to retire the word “Agile.”

Retro

The really sad part of this for me is seeing young developers denigrate Agile and think of it as a means for management to extract unrealistic promises and push dev teams to work crazy hours. I get it. The only Agile they’ve ever known has been a mechanism of control imposed on them, not a tool of self-empowerment they joyously embraced. But I hope kicking off some discussions around the history and original vision can help us remember how things were supposed to go.

The good news in all of this is that the principles of the Agile Manifesto are as wise and relevant today as they were 20 years ago. And even supposed apostates like Jeffries and Thomas still think that.

Jeffries in the article mentioned above, says, “However, the values and principles of the Manifesto for Agile Software Development ***still offer the best way I know to build software***, and based on my long and varied experience, I’d follow those values and principles no matter what method the larger organization used.”

I agree.

It’s not hip or cool to talk about Agile now. Agile is boring. *Everybody does Agile, right?* Now is the perfect time to reflect on the last 20 years and ask ourselves a few questions:

- What went right?
- What went wrong?
- What do we want to do differently next time?

A few of us at Simple Thread who lived through the revolution plan to reflect on each of the original 12 Agile Principles over the coming months, contextualize their original meaning, and consider their value in the current milieu of software development.

My hope is that by studying the founding principles, we can learn from the past, and in the words of Dave Thomas, we can retain our *agility* even if we choose to abandon “Agile”.

Loved the article? Hated it? Didn’t even read it?

We’d love to hear from you.

Reach Out

Agile Softwareentwicklung: Entwickler und Controller gehören an einen Tisch

Veröffentlicht von
Stefan Adolf
Geschätzte Lesezeit
11 min
Hinzugefügt am
2021-12-08
Adresse
https://www.heise.de/hintergrund/Entwickler-und-Controller-gehoeren-an-einen-Tisch-6144036.html?view=print

Agile Softwareentwicklung: Entwickler und Controller gehören an einen Tisch

Agile Softwareentwicklung: Entwickler und Controller gehören an einen Tisch

Stefan Adolf

Mit agiler Softwareentwicklung und Controlling sollen Projekte reibungslos ablaufen. In der Praxis geht das aber häufig schief – warum eigentlich?

Klassisches Controlling und agile Development-Methoden entstammen unterschiedlichen Welten, verfolgen aber grundsätzlich dasselbe Ziel: Softwareentwicklungsprojekte detailliert planen und flexibel aussteuern. In der Praxis erweist sich das Zusammenspiel aufgrund der unterschiedlichen Ansätze aber häufig als schwierig. Viel zu oft laufen Aufwand und Kosten aus dem Ruder, obwohl das Coding in kleinen, agilen Schritten stattfindet und dank regelmäßiger Meetings scheinbar jedes Problem besprochen wird. Was läuft da schief und wie lässt sich das verhindern?

Dass Projekte egal welcher Art und Größe von einer soliden Planung profitieren, wissen sowohl Projektmanager als auch Entwicklerinnen und Entwickler. Jedoch ist die jeweilige Herangehensweise durchaus verschieden. Die Anforderungen kommen aus unterschiedlichen Richtungen, und nicht zuletzt gehen auch die Meinungen darüber auseinander, wann ein erreichtes Ergebnis als gut bewertet werden kann. Das führt häufig zu einer unüberschaubaren Lage – ein Blick auf die Details verrät, warum.

Trotz agiler Entwicklung werden Probleme zu spät erkannt

Agile Development-Methoden [1] haben die Art und Weise, wie Software entwickelt wird, in den

letzten Jahren entscheidend verändert. Auch wenn nicht in jedem Unternehmen ausschließlich Scrum Master und Agile Coaches für die Strukturierung der Entwicklungsprojekte verantwortlich sind, ist die von Major Releases getriebene Softwareentwicklung in der unternehmerischen Praxis kaum noch anzutreffen. Und das hat gute Gründe: Agile Methoden sind effizienter, transparenter und führen schneller zu verwertbaren Ergebnissen.

Im Wesentlichen geht es darum, durch iterative Schritte, Code Reviews und Tests potenzielle Fehlerquellen früher zu erkennen und entsprechend gegenzusteuern. So werden zu Beginn Projektvisionen klar ausformuliert, Anforderungen katalogisiert und Aufgaben priorisiert. In regelmäßigen Meetings diskutiert das Team möglichst alle Details der in kleinere Stücke zerlegten Projekte – von Akzeptanzkriterien über Integrationsanforderungen bis hin zu Umsetzungsschwierigkeiten.

Wieso passiert es trotzdem, dass die Verantwortlichen oft erst spät im Verlauf des Projektes Budget- und Zeitengpässe erkennen? Das Problem liegt meist nicht bei der Entwicklung, sondern vielmehr an deren ungenügender Integration ins Gesamtprojekt. Denn während DevOps-Initiativen versuchen, Entwicklung und Betrieb schon früh eng miteinander zu verzahnen, um spätere Operations-Anforderungen von Beginn an in den Entwicklungsprozess einfließen zu lassen, finden sich entsprechende Initiativen auf der Business-Seite eher selten. Mit anderen Worten: Entwicklerinnen und Entwickler gestalten Software heute zumeist mit klar definiertem Scope, maßgeschneidert für die Kundenanforderungen. Jedoch sitzen sie in den seltensten Fällen von Beginn an mit am Planungstisch – obwohl das sinnvoll wäre, denn ihre Einschätzungen und bevorzugten Herangehensweisen können sowohl Zeit- als auch Kostenpläne entscheidend beeinflussen. Allzu oft beschränkt sich das gewünschte Feedback auf die Anzahl der benötigten Sprints. Das greift allerdings zu kurz, denn die Softwareentwicklung ist ein kreativer Prozess, der sich schwer planen oder in enge Vorgaben pressen lässt – zumindest dann nicht, wenn der Anspruch an die Qualität der Software hoch ist.

Lesen Sie auch

Agile Softwareentwicklung: Auf welche Arten Agilität schiefgehen kann

[2]

Technical Debt definieren und einpreisen

Es ist unvermeidlich, dass Softwareprojekte im Laufe der Zeit **Technical Debt (technische Schulden) anhäufen [3]**. Bedingt durch Projektziel und Geschwindigkeit lassen sich im Prozess trotz überzeugender Vorteile nicht immer vollständige Testabdeckung, automatisiertes Testing, vollautomatische Main-Deployments und eine clevere Architektur sowie die reibungslose Orchestrierung von Updates aller Komponenten gewährleisten. Um ein schnell wachsendes Projekt später zu stabilisieren, müssen solche Aufgaben unweigerlich nachgeholt werden.

Da sich technische Schulden nicht vermeiden lassen, gilt es für das Controlling, sie einzupreisen und bei der regelmäßigen und mittelfristigen Planung zu berücksichtigen. Im Idealfall räumen Projektverantwortliche Entwicklerinnen und Entwicklern regelmäßige Sprints ein, um den Schuldenberg abzutragen. Wer das nicht tut, riskiert ein zunehmend instabiles System, das sich schwerer verändern und erweitern lässt. Je konsequenter und liberaler das Controlling technische Schulden behandelt, desto leistungsfähiger agiert das Entwicklerteam im langfristigen Projektverlauf. Auftretende Probleme lassen sich beseitigen, ohne das gesamte Projekt aus der Bahn zu werfen.

Lesen Sie auch

Fachliche Schulden als Kontrapunkt zu "Technical Debt"

[4]

Developer gehören ins Planungsteam

Entwicklerinnen und Entwickler sind längst mit dafür verantwortlich, dass Kunden die gewünschten Features in der Form bekommen, die sie möchten. Der agile Ansatz umfasst die frühe und kontinuierliche Auslieferung von Software, Offenheit gegenüber Änderungswünschen in jeder Projektphase – wenn diese vorteilhaft für die Kunden sind –, ständige Rücksprache mit allen Beteiligten, wie beispielsweise den Fachabteilungen, die die Software als Kunden in Auftrag gegeben haben.

In der Praxis werden Entwicklerteams meist aber erst später im Prozess eingebunden, sodass bei ihnen häufig das Gefühl aufkommt, die Planung erfolge von oben herab und die Kunden könnten sich wünschen, was sie wollen – weil sie ja schließlich dafür bezahlen. Unter solchen Bedingungen neigen auch Entwicklerinnen und Entwickler, die sich agile Methoden auf die Fahnen geschrieben haben, dazu, nur noch Dienst nach Vorschrift zu leisten und stumpf alle Aufgaben abzuarbeiten. Das führt in der Regel zu Frustration und wenig zufriedenstellenden Ergebnissen.

Die wichtigste Konsequenz aus diesen Überlegungen ist, das Development-Team von Beginn an vollständig miteinzubeziehen. Wenn diejenigen, die das Projekt maßgeblich umsetzen sollen, bereits Zeit- und Budgetplanung mitverantworten, ergeben sich klarere Rahmenbedingungen für die Umsetzungsphase. So können außerdem die Erkenntnisse der agilen Meetings des Dev-Teams in die Gesamtplanung zurückfließen. Wenn sich beispielsweise die Herausforderung ergibt, weitere Sprints oder Code Reviews einzufügen oder unvorhergesehene Inkompatibilitäten auflösen zu müssen, kann sich das gesamte Projekt verzögern. Erfolgt jedoch die Rückkopplung aus dem Entwicklerteam direkt mit den Business-Verantwortlichen, lässt sich unmittelbarer nachsteuern oder zumindest rechtzeitig umplanen.

Was kann Controlling und was nicht?

Die Rückkopplung sollte fachlicher Natur sein: Wo muss beispielsweise nachgebessert werden, damit sich die Anforderungen der Kunden effizient umsetzen lassen? Welche technischen Spezifikationen sind zu überdenken? Softwareprojekte unterliegen allerdings wie andere Projekte dem anfangs definierten Zeit- und Budgetrahmen. Dessen Einhaltung zu gewährleisten, ist die Kernaufgabe des Controllings, die üblicherweise über die reine "Kontrolle" hinausgeht. Viele Unternehmen verstehen Controlling jedoch nur als Kontrollmechanismus, der an bestimmten Meilensteinen prüft, wie viel Budget das Projekt bereits verschlungen hat und ob der Zeitplan eingehalten werden kann. Das ist jedoch zu kurz gedacht, denn Controlling kann mehr.

Projektbeteiligte sollte das Controlling eher als Steuerungsfunktion verstehen und nutzen. Nach der Anfangsplanung gehört es zu den Aufgaben des Controllers, den Projektfortgang laufend zu analysieren und, falls sich generelle Engpässe oder Veränderungen ergeben, entgegenwirkende Maßnahmen vorzuschlagen. Wichtig dabei ist, nicht nur den Status quo des Budget- und

Zeitverbrauchs zu betrachten, sondern auch weitere Informationen zu sammeln. Hier schließt sich der Kreis: Je enger das Entwicklerteam und alle anderen Projektbeteiligten mit dem Controlling zusammenarbeiten, umso effizienter lässt sich der Projektverlauf steuern.

Ist beispielsweise frühzeitig absehbar, dass die Entwicklung wegen unterschätztem Integrationsaufwand länger dauert als geplant, muss dies nicht zwingend in mehr Zeitdruck für die Entwicklerinnen und Entwickler oder gar eine Verschiebung der Auslieferung münden. Das Controlling könnte auch zu dem Ergebnis kommen, dass es insgesamt vorteilhafter ist, zusätzliche Entwicklerkräfte in das Projekt einzubeziehen – oder andere Projekte mit geringerer Priorität vorübergehend zurückzustellen. Auch lassen sich die Kosten für ein benötigtes Tool gegebenenfalls als Effizienz-Maßnahme begreifen, die das gesamte Projekt voranbringt, und nicht als zusätzlichen Kostenaufwand. Solche Entscheidungen können Controller aber nur treffen, wenn sie frühzeitig und umfassend informiert sind.

Microcontrolling, ohne den kreativen Raum einzuengen

Wie kleinteilig muss das Controlling sein? Es gilt, die Balance zwischen harten Zahlen, Fakten und Steuerungsmaßnahmen auf der einen Seite und kreativem Freiraum für die **Softwareentwicklung [5]** auf der anderen Seite zu wahren. Während zu viel Planung behindert, engt zu viel Standardisierung ein. Manche Probleme lassen sich nur mit kreativem Coding und flexiblen Methoden lösen. Die Bereitschaft, Dinge auch auf andere Weise anzugehen, gehört zum modernen Software-Engineering. Gerade in Zeiten der Containerisierung und des Cloud Deployments stehen viele Tools, Optionen und Workarounds zur Verfügung. Warum nicht auch einmal in der Gruppe vor dem Bildschirm sitzen und über Einzelheiten im Quellcode oder die richtige Interpretation von StackOverflow-Antworten diskutieren – wie es beim Mob-Programming vorgesehen ist. Den dafür nötigen Freiraum dürfen Controlling-Kriterien nicht zu stark einengen oder gar ausschließen.

In den agilen Entwicklungsprozess integriertes Microcontrolling empfiehlt sich als praktikabler Lösungsansatz. Die Idee dabei ist, die Variablen, die insbesondere für das Controlling wichtig sind, beim täglichen Ausbalancieren des Entwicklungsprozesses einzubeziehen. Wenn also das Dev-Team im Scrum-Meeting Sprints umplant oder Änderungen am Product-Backlog vornehmen muss, sollten auch die Auswirkungen auf die Zeit- und Kostenplanung beziffert werden – und zwar möglichst genauso kleinteilig, wie es für den Coding-Prozess erfolgt. Denn wenn sich im Sprint-Review ergibt, dass ein zusätzlicher Sprint angehängt oder weitere Anpassungen umgesetzt werden müssen, dann verschiebt sich alles Nachfolgende. Sobald es Entwicklerinnen und Entwicklern gelingt, solche Aspekte standardmäßig in ihre Planung zu integrieren und mit dem Controlling abzustimmen, lassen sich Abweichungen früher erkennen. Das wiederum erleichtert sinnvolles Gegensteuern, ohne die gesamte Planung über den Haufen werfen zu müssen.

Erwartungs- und Anforderungsmanagement: den Auftraggeber stärker einbinden

Sprint-Reviews sind eine Gelegenheit – und im Sinne der Methode auch dafür vorgesehen –, sämtliche Stakeholder eines Projektes an einen Tisch zu bringen. Auftraggeber und Nutzer können Features testen und ihre Funktionalität sowie praktische Anwendbarkeit beurteilen. In den Reviews treten regelmäßig gerade jene Probleme zutage, die nur im Zusammentreffen der verschiedenen Disziplinen entdeckt werden. Ein typisches Beispiel dafür ist eine Funktion in der Software, die auf dem Papier alle definierten Kriterien erfüllt, bei Anwenderinnen und Anwendern aber durchfällt, weil sie den entsprechenden Button nicht finden. Denn den hat das Entwicklerteam zwar an einer

technisch sinnvollen, aber aus Sicht einer optimalen User Experience (UX) wenig geeigneten Stelle platziert. Dann entscheidet das Feedback in den Sprint-Reviews darüber, wie es weitergeht – mit den beschriebenen Auswirkungen auf die Dauer und die Kosten des Gesamtprojekts.

Lesen Sie auch

Andererseits haben dadurch auch Entwicklerinnen und Entwickler die Möglichkeit, ihrerseits auf erforderliche Anpassungen aufmerksam zu machen. Nicht selten schließen technische Notwendigkeiten bestimmte Funktionen gänzlich aus oder verändern sie so stark, dass die Nutzer unzufrieden sind. In der gemeinsamen Diskussion lässt sich Verständnis aufbauen und – in den meisten Fällen – eine angemessene oder sogar bessere, neue Lösung finden. Beinahe automatisch entwickelt sich dabei ein aktiv gelebtes Erwartungs- und Anforderungsmanagement: eine Balance zwischen dem Gewünschten und dem Möglichen – stets unter der Beobachtung des Controllings.

So früh wie möglich "positiv enttäuschen": Softwareprojekte wieder auf Kurs bringen

Dabei müssen Controller vor allem im Auge behalten, dass das **Projektmanagement [7]** nicht zu agil über alle Planungsgrenzen hinweg arbeitet, damit sich die Auswirkungen von Änderungen im Projekt möglichst frühzeitig quantifizieren und mit der Gesamtplanung in Einklang bringen lassen. Unumgänglich ist, dass das Controlling auch bremsend eingreift. Wenn beispielsweise der Aufwand für eine neue, erweiterte Funktion im Verhältnis zum Nutzen zu groß wird, muss es die Möglichkeit geben, abzubrechen und einen neuen Ansatz zu entwerfen.

Je früher im Projektverlauf das geschieht, umso besser: Auch wenn solche Einschnitte sich im ersten Moment wie eine Enttäuschung oder Einschränkung anfühlen, bewahren sie das Gesamtprojekt häufig vor schlimmeren Folgen. Oft treten im Projektalltag Probleme erst zutage, wenn sie bereits hohen Aufwand und entsprechende Kosten verursacht haben. Häufig ist es dann zu spät, die Dinge zurückzurollen. Viel Arbeit ist vergebens getan, schlimmstenfalls muss das Projekt eingestellt werden. Frühes „positives Enttäuschen“ kann das verhindern. Mit geeigneten strategischen, organisatorischen und technischen Maßnahmen – von der Anpassung der Anforderungen über die Aufstockung der Ressourcen bis hin zum Erwerb weiterer Tools oder geeigneter Infrastruktur – lässt sich das Projekt besser aussteuern.

Gemeinsam stärker

Agile Entwicklungsmethoden eignen sich gut, um das Projekt-Controlling granular und praxisnah umzusetzen. Voraussetzung ist allerdings, dass alle Stakeholder von Beginn an an einem Tisch sitzen und gemeinsam die Anforderungen und die Aufwandsplanung unter einen Hut bringen.

- Entwicklerinnen und Entwickler gehören ebenso wie die Fachabteilungen (beziehungsweise die Kunden) und Vertreter des Controllings ins Kernteam für die Planung und Budgetierung.
- Microcontrolling bedeutet, Controlling in die einzelnen Teilphasen des Entwicklungsprozesses einzubinden: Wenn in jedem Entwickler-Meeting die Auswirkungen auf Kosten und Ressourcen bedacht und geplant werden, lassen sich frühzeitig steuernde Maßnahmen ergreifen.

- Die technischen Schulden des Projekts müssen in der Planungsphase erhoben, eingeplant und budgetiert werden, damit sie nicht das Projekt in Verzug bringen.
- Die in der Praxis üblichen Zeiträume zwischen definierten Meilensteinen sind für ein sinnvolles Microcontrolling zu lang. Sinnvoller ist es, tägliche Status-Updates zu erstellen, um den Zeit- und Kostenaufwand für alle Teil- und Unteraufgaben zu erfassen und den Restaufwand zu kalkulieren.
- Auftraggeber sollten ebenso eng in alle Phasen eingebunden sein. Das schafft Verständnis für die Abhängigkeiten des jeweils anderen, vermeidet Missverständnisse und erleichtert Anpassungen des Produkts.
- Eingefahrene Kommunikationsprozesse gilt es regelmäßig kritisch zu hinterfragen. Kollaborations-Tools sind in der Praxis oft weniger sinnvoll als gedacht. Es lohnt sich, zu überprüfen, ob die bestehenden Prozesse effizient sind, und ob sich Tools so nutzen lassen, dass sie Zeit sparen, statt zusätzliche Kosten zu verursachen.

Ein agiler Prozess ist immer nur so agil wie sein am wenigsten dynamischer Teilnehmer. Der Sinn hinter **Werkzeugen wie Scrum [8]** ist es nicht, aus Entwicklerarbeit Burndown-Charts zu erstellen. Es geht vielmehr um die Beteiligung aller Stakeholder, mit Rücksicht auf die jeweiligen Bedürfnisse und Aufgaben: Entwicklerinnen und Entwickler brauchen Freiräume, um schwierige Probleme zu lösen – sowohl im zeitlichen Ablauf als auch bei der Wahl ihrer Werkzeuge.

Softwareentwicklung bleibt bei allem notwendigen Controlling eine kreative, schwer planbare Aufgabe. Zugleich müssen Auftraggeber und Projektcontroller in der Lage sein, Zwischenergebnisse detailliert nachzuvollziehen, um Anforderungen gegebenenfalls zu überdenken und neu zu priorisieren. Dann gelingt es, gemeinsam eine Software zu entwickeln, die ein bestmöglicher Kompromiss aus Erwartetem und technisch Machbarem ist und die weder den Zeit- noch den Budgetrahmen sprengt.

Stefan Adolf
ist Developer Ambassador bei Turbine Kreuzberg. Als Fullstack-Entwickler mit Schwerpunkt auf Applikationen, IoT und Integration ist die Kommunikation mit der Developer-Community seine primäre Aufgabe.

()

URL dieses Artikels:
https://www.heise.de/-6144036

Links in diesem Artikel:

[1] https://www.heise.de/thema/Agile-Softwareentwicklung

[2] https://www.heise.de/ratgeber/Agile-Softwareentwicklung-Auf-welche-Arten-Agilitaet-schiefgehen-kann-5054837.html

[3] https://www.heise.de/developer/artikel/Technische-Schulden-entstehen-einfach-so-3969279.html

[4] https://www.heise.de/hintergrund/Fachliche-Schulden-als-Kontrapunkt-zu-Technical-Debt-4284588.html

[5] https://www.heise.de/thema/Softwareentwicklung

[6] https://www.heise.de/hintergrund/Requirements-Engineering-in-der-agilen-Softwareentwicklung-4617665.html

[7] https://www.heise.de/thema/Projektmanagement

[8] https://www.heise.de/thema/Scrum

[9] mailto:map@ix.de

Agile Softwareentwicklung: Die vielen Hüte der Product Owner

Veröffentlicht von
Cornelia Seraphin
Geschätzte Lesezeit
15 min
Hinzugefügt am
2021-12-08
Adresse
https://www.heise.de/hintergrund/Agile-Softwareentwicklung-Die-vielen-Huete-der-Product-Owner-6072808.html?view=print

Agile Softwareentwicklung: Die vielen Hüte der Product Owner

Agile Softwareentwicklung: Die vielen Hüte der Product Owner

Cornelia Seraphin

(Bild: iChzigo/Shutterstock.com)

Product Owner müssen je nach Kontext verschiedene Aufgaben übernehmen. Die sechs Kernrollen lassen sich als Hüte darstellen – eine Vorstellung.

Product Owner ist mehr als nur eine Rolle in Scrum. Je nach Kontext müssen sie verschiedene Aufgaben aus unterschiedlichen Bereichen übernehmen. Als Hüte dargestellt ergeben sich sechs konkrete Aufgabengebiete für Product Owner. Der Artikel stellt sie vor und gibt Tipps für den Umgang damit.

Der **Scrum Guide [1]** liefert auf seinen rund 13 Seiten nur wenige Aussagen zu den Tätigkeiten eines Product Owners (die Seitenzahl variiert je nach Sprache). Um das volle Potenzial dieser in Scrum essenziellen Rolle zu entfalten, sollten deshalb alle Beteiligten die Aufgaben und Verantwortlichkeiten von Product Ownern kennen und auf geeignete Weise in ihre jeweilige Projekt- und Organisationsstruktur integrieren. Um dabei nicht den Überblick über die vielfältigen Aufgaben zu verlieren, hilft es, sich die Product Owner mit unterschiedlichen Hüten vorzustellen.

Hut des Produktmanagers

Als Produktmanager übernehmen Product Owner die inhaltliche und wirtschaftliche Verantwortung für das Produkt. Sie sind Visionäre und Wertsteigerer, gleichzeitig müssen sie den Erfolg des

Produkts sicherstellen.

Inhaltliche Verantwortung bedeutet: Sie sorgen dafür, dass das passende Produkt für ihre Kunden entwickelt wird. Um das zu bewerkstelligen, müssen sich Product Owner sehr gut in ihrer Domäne und im Markt auskennen und sollten die aktuellen Trends und ihre Mitbewerber kennen. Die Analyse des Marktes und der Kundenzufriedenheit ist eine durchgängige Tätigkeit während der gesamten Produktentwicklung.

Product Owner: Definition aus dem Scrum Guide

"The Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team. How this is done may vary widely across organizations, Scrum Teams, and individuals."

– Scrum Guide, November 2020

Product Owner entwickeln und kommunizieren das Produktziel. Damit es in den für den jeweiligen Kontext richtigen Schritten umgesetzt wird, erstellen Product Owner eine Roadmap für die Produktentwicklung. Davon leiten sie Release-Pläne bis hin zu den Sprint-Zielen ab – jedes Release soll schließlich einen Mehrwert für die Kunden schaffen. Je kürzer die Release-Zyklen ausfallen, desto schneller erhalten Product Owner die Rückmeldung der Kunden darüber, ob das Scrum-Team bei der Produktentwicklung den richtigen Weg eingeschlagen hat.

Gleichzeitig müssen Product Owner bei ihrer Release-Planung beachten, wie aufwendig es für ihre Kunden ist, ein neues Release einzusetzen. Sollte es nicht den erhofften Kundennutzen liefern, oder falls sich die Bedürfnisse beziehungsweise die Rahmenbedingungen ändern, passen Product Owner ihre ursprüngliche Planung kurzfristig an, prüfen wiederum diesen neu eingeschlagenen Weg und geben so den optimalen Fahrplan für die Produktentwicklung vor. Schritt für Schritt entsteht also genau das Produkt, das die Kunden wirklich wollen und brauchen.

Product Owner sind obendrein für das Marketing ihres Produkts verantwortlich. Sie verbreiten ihr Produktziel im Unternehmen und bringen zudem die Produktentwicklung mit den Unternehmenszielen in Einklang.

Wirtschaftliche Verantwortung für das Produkt

Mit ihrer wirtschaftlichen Verantwortung für das Produkt müssen sie dafür Sorge tragen, dass sich die Investition der Produktentwicklung für ihr Unternehmen lohnt. Sie nutzen geeignete Kennzahlen zum Monitoring, um den Return on Investment (ROI) im Auge zu behalten und den Nutzen für Kunden und Unternehmen sicherzustellen.

Es ist wichtig, dass Product Owner das Motto "Inspect and Adapt" leben (Überprüfen und Anpassen); und zwar vom Produktziel über das Messen und Verfolgen des gelieferten Wertes mit jedem Release bis hin zur wiederholten Validierung des Produkts mit den Stakeholdern und dem Marktplatz.

Sich bei den Aufgaben helfen lassen

Product Owner dürfen sich bei ihren vielfältigen Aufgaben helfen lassen und müssen notwendige Analysetätigkeiten des Marktes und der Wettbewerber nicht selbst erledigen. Sie können diese Arbeit an Mitarbeiter des Unternehmens delegieren, etwa an die Marketingabteilung, an andere Teammitglieder oder auch an externe Partner. Sie müssen allerdings die Daten und aktuellen Trends kennen, um die bestmöglichen Entscheidungen treffen zu können. Außerdem sorgen sie dafür, dass sie ihr Wissen darüber an das Scrum-Team weitergeben.

Strategische Aktivitäten des Produktmanagements

Viele der Aktivitäten des Produktmanagements sind nicht direkt mit Scrum verknüpft. Dennoch müssen Product Owner diese Aktivitäten kennen und in ihrer Arbeit als Produktmanager und "Value Maximizer" berücksichtigen.

Strategische Aktivitäten für das Produktmanagement:

- Industrie und Wettbewerber analysieren
- Return on Investment maximieren
- Machbarkeitsprognose erstellen
- Produktstrategie entwickeln
- Releases planen

- Kunden und ihre Bedürfnisse identifizieren
- Fahrplan für die Produktentwicklung erstellen
- Ergebnisse überwachen
- Externe Kommunikation erstellen

- Produkt weiterentwickeln
- Releases einführen
- Business Case erstellen
- Produktanforderungen identifizieren
- Markteinführung des Produkts

- Strategie zur Kundenbindung entwickeln
- Funktionalitäten für das Produkt definieren
- Produktlebenszyklus managen
- Produkt einstellen
- Marketing und Branding

Hut des Requirements Engineers

Als Requirements Engineer stellen Product Owner sicher, dass das Produkt die Anforderungen und Erwartungen aller Stakeholder befriedigt. Sie verantworten, dass die Anforderungen im Product Backlog beschrieben, verstanden sowie sortiert sind und dass das Backlog für jeden Stakeholder zugänglich ist.

Ein gutes Produkt lässt sich nur entwickeln, wenn Product Owner die Wünsche und Bedürfnisse der Kunden und aller anderen Stakeholder kennen. Diese Stakeholder gilt es entsprechend zu identifizieren und während der gesamten Produktentwicklung aktiv zu betreuen. Mittels Methoden des Requirements Engineering ermitteln, klären und priorisieren sie die sich ergebenden Anforderungen sämtlicher Stakeholder an das Produkt. Das bedeutet auch, dass sie mit zuweilen widersprüchlichen Anforderungen und Bedürfnissen konfrontiert sind. Zu den Sprint Reviews laden sie die geeigneten Stakeholder ein, um Feedback einzuholen und die weitere Produktentwicklung in eine angemessene Richtung zu steuern.

Die konkreten Anforderungen – detaillierte ebenso wie grobe Ideen und lose Hypothesen – erfassen Product Owner im Product Backlog. Das Backlog sortieren sie anhand des Business Values, des Risikos sowie anhand der Abhängigkeiten zwischen den Product Backlog Items (PBI). Auch die Schätzgröße und somit die Kosten für die Umsetzung einer Anforderung fließt in die Sortierung mit ein. Product Owner verantworten auch, dass die PBIs in angemessener Detailtiefe beschrieben sind und die Developer die Items verstehen. So stellen Product Owner sicher, dass Developer die Items im bevorstehenden Sprint auch tatsächlich umsetzen können.

Definition of Done

Die Definition of Done ist ein Artefakt aus dem Scrum Guide. Product Owner und die Developer erstellen sie gemeinsam. Sie enthält alle Qualitätskriterien, die ein Product Backlog Item (PBI) erfüllen muss, um als fertig umgesetzt zu gelten und in das aktuelle Produktinkrement integriert werden zu können.

Scrum Guide: "The Definition of Done is a formal description of the state of the Increment when it meets the quality measures required for the product. […] The Definition of Done creates transparency by providing everyone a shared understanding of what work was completed as part of the Increment."

Die PBIs für die nächsten zwei Sprints sollten deshalb "Sprint-ready" sein und alle Informationen enthalten, die für die Implementierung notwendig sind, etwa Abnahmekriterien, Akzeptanztests oder UI-Entwürfe. Ferner dürfen die Items nur so groß sein, dass sie gemeinsam mit einigen anderen Items innerhalb eines Sprints entsprechend der Definition of Done umgesetzt werden können. Dabei greifen Product Owner gegebenenfalls auf Techniken wie das Story Splitting zurück, um zu große Items in kleinere zu zerlegen.

Anforderungen, die erst später oder eventuell gar nicht umgesetzt werden, weil sich die Rahmenbedingungen geändert oder neue Erkenntnisse ergeben haben, werden im Backlog nur als Idee vorgemerkt. In die Ausarbeitung dieser Ideen stecken Product Owner dann aber keine Energie, getreu dem Motto einer "Just-in-time Specification". Der Vorteil dieser Herangehensweise ist, dass sich unklare Anforderungen, Annahmen und Hypothesen zunächst anhand von echtem Kundenfeedback prüfen lassen, bevor Aufwand in die Spezifikation fließt.

Lesen Sie auch

Dritter Product Owner Day geplant: Referenten können sich ab sofort bewerben

[2]

Product Owner sind wiederum auch für alle Entscheidungen verantwortlich, die sich im Product Backlog widerspiegeln. Das Product Backlog ist die einzige Anforderungsquelle, alle Anforderungen sind in diesem zentralen Scrum-Artefakt enthalten. Ausschließlich an ihr orientieren sich die Developer. Dabei sollten Product Owner auch stets prüfen, ob ihre Einschätzung des Business Value der einzelnen PBIs noch passt. Dafür eignen sich die verschiedenen **Metriken, die im Evidence Based Management beschrieben sind [3]**.

Nicht alle PBIs wie User Stories müssen Product Owner selbst schreiben. Sie können sich von Mitgliedern des Entwicklungsteams oder von den Stakeholdern helfen lassen. Oft unterstützen Business-Analysten den Product Owner[A1] [A2], die Stories zu schreiben, weil eine Person allein dies für Produkte mit komplexer Fachlichkeit kaum bewältigen kann. Die Helfer arbeiten sich in die Details der Story ein, der Product Owner behält den Überblick. Aber die Entscheidungshoheit und die Priorisierung des Backlogs bleiben weiterhin die Aufgabe des Product Owners.

Hut des User-Experience-Experten

Als Experten für User Experience (UX) stellen Product Owner sicher, dass das Produkt die Bedürfnisse und Wünsche seiner Anwenderinnen und Anwender befriedigt und ihnen einen Mehrwert bringt.

Damit das Produkt am Markt erfolgreich ist, muss es die Kundenbedürfnisse erfüllen. Daher ist es als Product Owner essenziell, die Wünsche und Bedürfnisse der Kunden, die das Produkt einsetzen, genau zu kennen. Entsprechend gilt es, die zukünftigen Kunden und ihre Bedürfnisse zu identifizieren. Mittels UX-Methoden entwickeln Product Owner die notwendige Empathie für ihre Kundinnen und Kunden, sodass sie stellvertretend für sie sprechen können. Für den weiteren Verlauf der Produktentwicklung erarbeiten die Product Owner unter anderem Personas und Customer Journey Maps, damit die Kundenbedürfnisse der wichtigsten Nutzergruppen für alle Beteiligten permanent präsent sind

In regelmäßigen Reviews und Usability Tests sammeln Product Owner das notwendige Feedback ein und lassen es in die Produktentwicklung einfließen. Zum Nachvollziehen der Kundenzufriedenheit setzen Product Owner geeignete Metriken ein und passen ihr Vorgehen gegebenenfalls an. Auch können sie Elemente zur Kundeninteraktion direkt in das Produkt integrieren, etwa Feedback-Möglichkeiten oder Monitoring-Mechanismen.

Dabei sind Product Owner aber nicht auf sich allein gestellt, sondern sie können sich Hilfe von UX-Experten holen. Sinnvollerweise gehören deshalb auch sie zu den cross-funktionalen, interdisziplinären Scrum-Teams. Als Experten für die User Experience unterstützen sie dann nicht

nur bei der Entwicklung von anwenderfreundlichen Bedienoberflächen, Workflows und Styleguides, sondern helfen Product Ownern auch, die Kunden und ihre Bedürfnisse zu verstehen, sie während der Entwicklung transparent zu halten und Feedback einzuholen.

Hut des Leaders

Als Leader geben Product Owner die Richtung der Produktentwicklung vor, inspirieren und motivieren das Scrum-Team, das Produktziel zu verwirklichen. Sie evaluieren und kommunizieren aktiv den Fortschritt der Produktentwicklung.

Damit ein Produkt erfolgreich entwickelt werden kann, brauchen Product Owner ein gut funktionierendes Team. Daher sind sie für die Zusammenstellung des Teams selbst zuständig. Außerdem tragen sie dafür Sorge, dass das Scrum-Team das Produktziel, das aktuelle Produkt, die Roadmap und die Kunden des Produkts kennt. Indem Product Owner das gesamte Scrum-Team einbeziehen, kann sich jedes Teammitglied als Teil von etwas Größerem fühlen, kennt den Zweck und das Ziel seiner Aufgaben und weiß, wohin die gemeinsame Reise gehen soll. Hierdurch kann jeder im eigenen Arbeitsbereich die besten Entscheidungen treffen, um das Produktziel Schritt für Schritt umzusetzen. Das schafft ein motivierendes Arbeitsumfeld für die Developer.

Product Owner behalten den Fortschritt der Produktentwicklung im Auge und klären ab, wie viel Arbeit für das gesetzte Ziel noch zu leisten ist. Sie nehmen Reporting-Aufgaben wahr und informieren die Stakeholder wie auch das Management ihres Unternehmens über den Status der Produktentwicklung.

Als Erweiterung der Product Owner Days richten *heise Developer* und *dpunkt.verlag* **am 24. Juni 2021 den ersten Agile Leader Day (online) [4]** aus. Wer teilnimmt, erfährt

- wie man selbstorganisierte Teams im Gegensatz zu Einzelpersonen führt,
- wie man Mitarbeiter beurteilt (wenn die Teamleistung im Vordergrund steht) und

- in welcher Menge und Form es dann noch disziplinarisches Führen braucht.

Zielgruppe sind die Leiterinnen und Leiter von Gruppen, Teams oder Abteilungen sowie erfahrene Scrum Master und Agile Coaches. Einzeln oder im Team können sie rings um den Online-Konferenztag auch **in Workshops ihr Wissen praktisch vertiefen [5]**.

Hut des Entscheiders

Als Entscheider hat der Product Owner das Sagen über alle Aspekte, die das Produkt und dessen Entwicklung betreffen – aus inhaltlicher wie aus wirtschaftlicher Sicht.

Product Owner sind für das Produkt und dessen Erfolg verantwortlich. Damit sind sie auch diejenigen, die die Entscheidungen für die Produktentwicklung treffen. Die gesamte Unternehmensorganisation sowie auch die Stakeholder müssen hinter dem Product Owner stehen, ihm das Mandat geben, die inhaltlichen und wirtschaftlichen Entscheidungen zu treffen. Nur so können Product Owner das volle Potenzial ihrer Rolle entfalten. Wenn Product Owner etwa wegen fehlender Budgetverantwortung nicht befugt sind, Entscheidungen zu treffen und zunächst Gremien wie Lenkungsausschüsse kontaktieren müssen, gerät die gesamte Entwicklung des Produkts ins Stocken. Entsprechend sinkt womöglich die Effizienz der Developer, weil sie nicht am aktuellen Thema weiterarbeiten können und somit die weitere Entwicklung blockiert ist.

Akzeptanzkriterien formulieren

Auch wenn sich Stakeholder nicht einigen können oder es mehrere Entwicklungsalternativen gibt, bestimmen Product Owner, was entwickelt wird und was nicht. Über Akzeptanzkriterien definieren sie, was sie bei der Umsetzung eines PBIs erwarten und was nicht. Den Developern geben sie Feedback über die abgelieferte Arbeit und zeigen den Weg auf, der noch zu gehen ist. Sie evaluieren mit jeder Iteration die Produktentwicklung und entscheiden, ob das Produktinkrement veröffentlicht wird.

Alle Entscheidungen eines Product Owners sind durch die Sortierung des Product Backlogs transparent. Die ganze Unternehmensorganisation muss diese Entscheidungen respektieren und darf nicht versuchen, Mitglieder des Scrum-Teams dazu zu bringen, an etwas anderem als dem Backlog zu arbeiten. Falls ein Stakeholder Änderungen am Backlog wünscht, muss dieser den Product Owner überzeugen, weil er die Entscheidungshoheit über das Product Backlog trägt.

Hut des Teammitglieds

Als Teil des Scrum Teams arbeiten Product Owner gemeinschaftlich mit den anderen Teammitgliedern daran, innerhalb der Sprints ein wertvolles Produktinkrement zu erarbeiten und gemeinsam kontinuierlich besser zu werden.

Um das zu erreichen, halten sich Product Owner an das Scrum-Framework, leben die Scrum-Werte und greifen auf das empirische Vorgehen zurück, um neue Erkenntnisse in die Produktentwicklung

einfließen zu lassen. Sie respektieren die Selbstorganisation (engl. self-management) der Developer und des gesamten Scrum Teams. Sie nutzen das Coaching und die Unterstützung des Scrum Masters hinsichtlich Scrum, Product-Backlog-Techniken und der empirischen Produktentwicklung sowie seine Hilfe bei der Organisation von Scrum-Events. Product Owner helfen auch dabei, Hindernisse zu beseitigen – unabhängig davon, ob sie das Scrum Team, die Organisation oder den Markt und die Kunden betreffen. Sie sehen sich nicht als Proxy zwischen Developern und Stakeholdern, sondern als "Matchmaker". Ziel des Matchmakings ist es, dass die Developer mit den Stakeholdern effektiv zusammenarbeiten können, um einerseits das Feedback und andererseits den Product Value zu maximieren.

Damit im Sprint ein wertsteigerndes Produkt entsteht, definieren die Product Owner im Sprint Planning gemeinsam mit den Developern das Sprintziel. Zu diesem Zeitpunkt stellen Product Owner sicher, dass das Scrum-Team das Sprintziel, das Produktziel sowie wichtige Informationen zur Produktentwicklung kennt. Sie sind dafür verantwortlich, dass den Developern ausreichend fertige PBIs bereitstehen, um das Sprint Backlog zu füllen und eine Vorhersage für den Sprint abgeben zu können. Während des Sprints klären sie gegebenenfalls aufkommende Fragen zu den PBIs und geben Feedback zu den bereits umgesetzten. Notfalls verhandeln sie den Scope (Funktionsumfang) neu oder akzeptieren Abstriche beim Sprint.

Sprint Review mit den Stakeholdern

Zur Sprint Review laden Product Owner die Stakeholder ein. Die Absicht dabei ist, konstruktives Feedback zur aktuellen Produktentwicklung zu erhalten und die nächsten Entwicklungsschritte aufzuzeigen. Der aktuelle Zustand und die Sortierung des Backlogs stehen während des Sprint Reviews zur gemeinsamen Diskussion. Gemeinschaftlich überlegen das Scrum-Team und die Stakeholder auch, wie sich der Wert des Produkts zukünftig für Kunden und Unternehmen noch weiter steigern lässt. Dieses wertvolle Feedback lassen Product Owner schnellstmöglich in das Product Backlog einfließen. Insbesondere Feedback und Änderungen, die die Spitze des Backlogs betreffen, müssen sie unverzüglich und noch vor dem Beginn des nächsten Sprints einarbeiten.

In der Sprint-Retrospektive erarbeiten Product Owner gemeinsam mit den anderen Mitgliedern des Scrum Teams Maßnahmen, um sich als Team kontinuierlich zu verbessern. Sinnvollerweise planen sie in jeder Sprint-Retrospektive für jede der Scrum-Rollen, also Scrum Master, Product Owner und Developer, eine konkrete Verbesserungsmaßnahme für den nächsten Sprint ein. Hilfreiche Beispielfragen sind: "Funktioniert die Kommunikation im Team gut?", "Ist der Product Owner ausreichend oft verfügbar?", "Lassen sich die PBIs noch besser beschreiben?" oder "Sind andere als die angewandten Methoden zielführender?"

An den Dailys müssen Product Owner nicht teilnehmen. Sie bieten ihnen aber eine gute Möglichkeit, sich über den aktuellen Stand der Entwicklung und womöglich aufkommende Probleme oder Fragestellungen zu informieren. Hierdurch können sie schnell Feedback geben und eingreifen, falls die Entwicklung eines PBIs in eine andere Richtung läuft als vorgesehen.

Schätzen und Verfeinern von Product Backlog Items

In Refinement- und Estimation-Aktivitäten präsentieren Product Owner neue PBIs und Erkenntnisse. Sie stellen damit sicher, dass alle Teammitglieder die bevorstehenden Aufgaben verstanden haben. Fragen und offene Punkte klären sie gemeinsam, damit das Team eine Gelegenheit für Rückfragen hat. Durch die Diskussion finden die Developer geeignete Lösungsideen und können sinnvolle Aufwandschätzungen abgeben. Gleichzeitig erhalten Product Owner Kenntnis über Risiken oder

Abhängigkeiten, die sie in das Product Backlog einarbeiten müssen. Eine grobe Schätzung des gesamten Backlogs durch die Developer hilft Product Ownern außerdem bei der Releaseplanung.

Product Owner aktualisieren das Product Backlog zeitnah mit allen neuen Informationen und Erkenntnissen, sodass es jederzeit den aktuellen Wissens- und Planungsstand widerspiegelt. Hierdurch ist für jedes Mitglied des Scrum Teams sowie für die Stakeholder ersichtlich, wie die geplante Reise der Produktentwicklung aussieht und was als Nächstes ansteht.

Product Owner tragen Verantwortung – müssen aber nicht alles alleine stemmen

Santa Clause Rule: Team Spirit

"Santa Clause Rule: No matter how busy Santa gets, he does not bring on other Santas. How does he cope?"

- Elves

Product Owner tragen die Verantwortung für den Produkterfolg und haben zahlreiche damit verknüpfte Aufgaben zu bewältigen. Je größer und komplexer ein Produkt wird, desto häufiger müssen sich Product Owner helfen lassen, um den Blick auf das Wesentliche richten zu können. Indem sie taktische Tätigkeiten wie das Schreiben von User Stories oder Rechercheaufgaben delegieren, können sie sich auf strategische Aspekte des Value-driven-Development konzentrieren. Damit kommen sie ihrer wichtigsten Aufgabe nach: den Product Value zu maximieren.

Der Auftrag des Product Owners bleibt es dabei immer, die Verantwortung für das Produkt und dessen Erfolg zu übernehmen, die Richtung vorzugeben und die notwendigen Entscheidungen zu treffen. Aber je strategischer Product Owner unterwegs sind, desto stärker müssen sie auch darauf achten, nicht die Tuchfühlung zum Scrum Team zu vernachlässigen, sondern mit dem Team eng in Kontakt zu bleiben.

Dipl.-Ing. Cornelia Seraphin
arbeitet seit über 15 Jahren im Software Engineering. Als Senior Business Consultant bei der msg leitet sie das Center of Competence Business Analysis, führt Anforderungsanalysen durch und begleitet Projekte als Product Owner. Ihr Wissen gibt sie unter anderem in Schulungen weiter, etwa in der zum Certified Professional Scrum Product Owner.

()

URL dieses Artikels:
https://www.heise.de/-6072808

Links in diesem Artikel:

[1] https://scrumguides.org/

[2] https://www.heise.de/news/Dritter-Product-Owner-Day-geplant-Referenten-koennen-sich-ab-sofort-bewerben-6072147.html

[3] https://www.scrum.org/resources/evidence-based-management

[4] https://ald.inside-agile.de/

[5] https://ald.inside-agile.de/#workshops

[6] mailto:sih@ix.de