

Volume
12

entwickler**spezial**

AGILE

DER STATUS QUO DER FLEXIBILITÄT

METHODEN

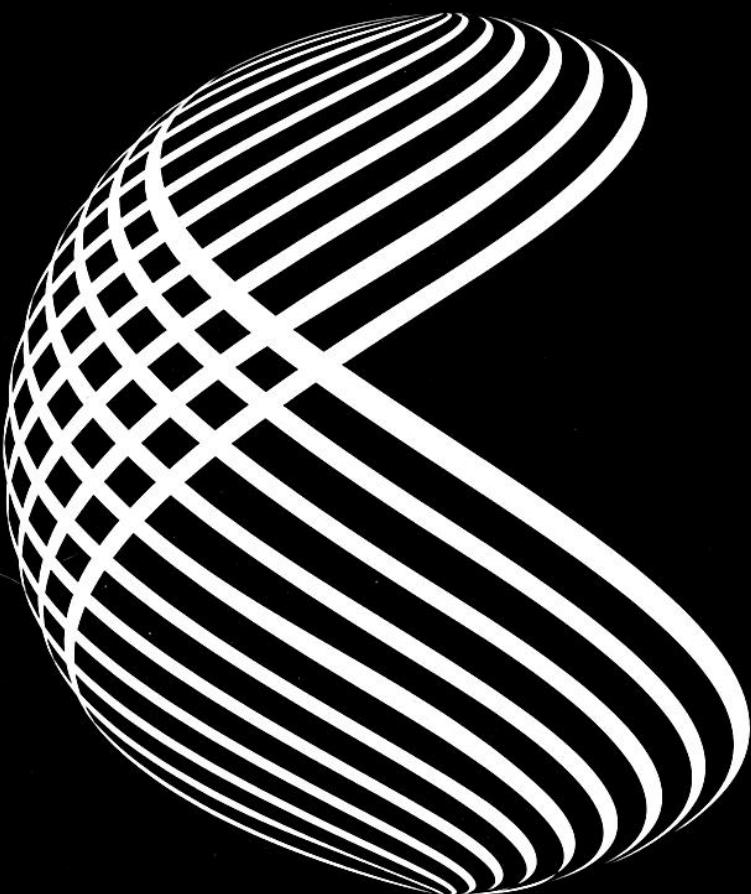
Neue Wege zur Kreativität

CODING PRACTICES

Never code alone

TEAMWORK

Crossfunktionale Teams



DE € 9,80
AT € 10,80
CH sFr 19,50
L € 11,00

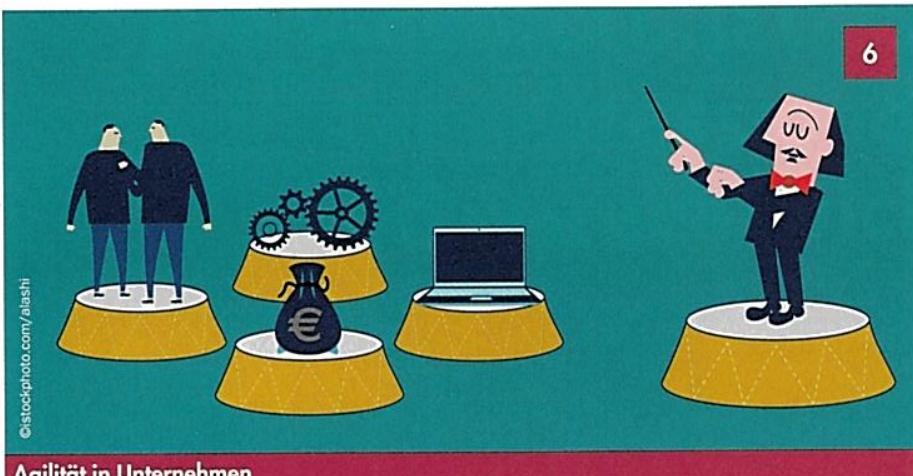


www.entwickler-magazin.de

+++ Scrum +++ Digitalisierung +++ Prototyping +++ Code Reviews +++ Projektleitung +++

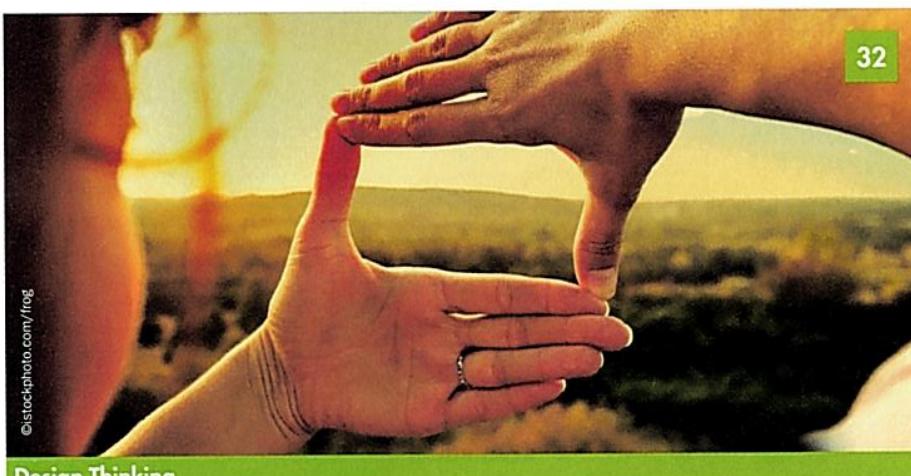
sokis0/Shutterstock.com

INHALT



Agilität in Unternehmen

Die Vorgehensweise, Software zu entwickeln, zu betreiben und zu nutzen, verändert sich in den Betrieben grundlegend. Ideen, die noch vor wenigen Jahren Start-ups vorbehalten waren, halten mittlerweile Einzug in etablierte Großunternehmen. Um Software immer schneller produzieren zu können, müssen nicht nur die Organisation eines Betriebs, sondern auch die Systeme verändert werden.



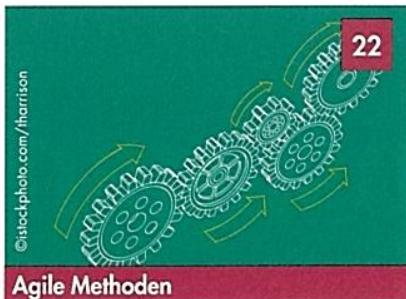
Design Thinking

Ob Softwareentwicklung eine ingenieurmäßige Disziplin ist oder ein großes Maß an Kreativität erfordert, ist eine seit den Anfängen der Programmentwicklung geführte Diskussion. Wir sind der Meinung, dass beide Fähigkeiten notwendig sind, um gute und vor allem kundengerechte Software zu erstellen. Der Ansatz des Design Thinkings ist nicht neu, hält aber gerade Einzug in die IT-Welt.



Wissenstransfer

Wie gehen wir schlau mit dem vorhandenen Wissen im Team um? Stichworte gibt es viele dazu: Cross-funktionalität, Interdisziplinarität, Generalisten, Spezialisten oder ein T-Shaped-Skill-Set. Diese Stichworte lassen sich auf folgende Fragen zurückführen: Welche Fähigkeiten benötigen wir, und bekommen wir das Wissen sinnvoll verteilt?



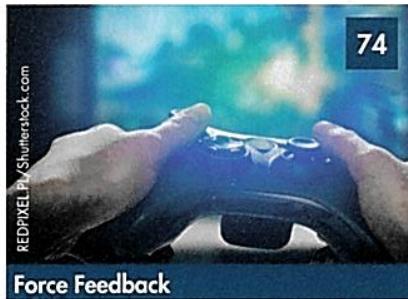
Agile Methoden

Wenn man über Softwareentwicklungsprojekte spricht, kommt man in der heutigen Zeit auf keinen Fall um das Thema Agilität herum. Seit einigen Jahren haben sich agile Herangehensweisen bei vielen Unternehmen etabliert – und das zu Recht. Nicht nur lassen sich Performance, Durchsatz und Qualität steigern, auch das Verständnis für moderne Entwicklungsmethoden lässt sich dadurch enorm verbessern.



Prototyping

Prototypen gehören zum guten Ton einer hippen Produktentwicklung. Am meisten Wirkung haben Prototypen, die in allen Phasen der Produktentwicklung das Feedback der Nutzer abholen. Trotzdem arbeiten noch wenige Entwickler systematisch mit Prototypen. Ein Grund ist möglicherweise, dass wir unter Prototypen oft schwergewichtige Softwareprototypen mit mehreren Wochen Entwicklungszeit verstehen. Warum eigentlich?



Force Feedback

Mit Force Feedback Programming, einer Erweiterung für Visual Studio, soll unsauberer Code sichtbar gemacht werden. Die Erweiterung geht aber noch einen Schritt weiter: Sie zwingt Entwickler, sauberen Code zu schreiben. Force Feedback Programming nutzt die Microsoft .NET Compiler Platform für die Analyse des Codes.

AGILITÄT IN UNTERNEHMEN

6 Der Widerspenstigen Zähmung

Agilität ist in Großunternehmen angekommen

16 Stillstand ist der Tod

Agilität – eine Grundvoraussetzung für funktionierende Digitalisierung

22 Auf dem Vormarsch

Agile Methoden in IT-Projekten

26 Agil genug für die digitale Zukunft?

Agilität in der Organisationsstruktur

28 Gemischtes Doppel

Wie sich UX-Design erfolgreich in den Scrum-Prozess integrieren lässt

METHODEN

32 Neue Wege zur Kreativität

Kreative Lösungen suchen und finden mit Design Thinking

38 Messbar mehr Qualität

Optimierte Agilität durch Best Practices

46 Leichtgewicht statt unflexibler Klotz

Kontinuierlich Feedback mit Prototyping einholen

52 Im Rückwärtsgang Domänen erforschen

Mehr Dynamik mit Event Storming

58 Energieschub für müde Produktentwickler

Wie fünf Tage alles verändern können

62 Der Product Owner im Casino

Backlogs priorisieren mit Costs of Delay und Monte-Carlo-Simulationen

CODING PRACTICES

68 Weniger ist mehr

Minimalismus für Softwareunternehmen

71 Never code alone

Pair Programming als Ausweg aus der Qualitätsfalle?

74 Force Feedback Programming

Sauberer Code? Gezwungenermaßen

80 Die Entwickler machen lassen

Qualitätsstandards als Communityaufgabe

TEAMWORK

84 Crossfunktionale Teams

Aktiver Wissenstransfer im Team ist notwendig

90 Vier Augen sehen mehr als zwei

Effektiver Einsatz von Codereviews

94 Was sich am einfachsten ändert: Sie selbst!

Teamstruktur durch skalierte Agilität verbessern

Agilität ist in Großunternehmen angekommen

Der Widerspenstigen Zähmung

Die Vorgehensweise, Software zu entwickeln, zu betreiben und zu nutzen, verändert sich in den Betrieben grundlegend. Ideen, die noch vor wenigen Jahren Start-ups vorbehalten waren, halten mittlerweile Einzug in etablierte Großunternehmen. Um Software immer schneller produzieren zu können, müssen nicht nur die Organisation eines Betriebs, sondern auch die Systeme verändert werden.

von Prof. Dr. Volker Gruhn



©istockphoto.com/alashii

Ist Schnelligkeit inzwischen das oberste Gebot für IT-Abteilungen? Oder ist Planungssicherheit weiterhin unverzichtbar, wenn Unternehmen neue Software entwickeln wollen? Sollen Entwickler den Funktionsumfang Schritt für Schritt während des Projekts erarbeiten, damit sie exakt passende Lösungen programmieren? Oder sollen sie genauen Spezifikationen folgen, damit alle Beteiligten von Anfang an wissen, was am Ende herauskommt? Der Blick auf agile und klassische Methoden der Softwareentwicklung offenbart viel Schwarz-Weiß-Denken in Unternehmen. Dabei liegt die Wahrheit auch in diesem Fall in der Mitte. Denn mit planorientierten Modellen, die auf der Annahme basieren, dass Spezifikationen weitgehend vollständig sind und die Projektverantwortlichen späte Anforderungen vermeiden sollten, werden Unternehmen kaum den schnellen Innovationstakt mitgehen können, den Start-ups mit ihren Angeboten inzwischen vorgeben. Auf der anderen Seite steht die agile Lehre in Reinkultur, der zumindest der Ruf vorausseilt, auf viele Projektstandards, wie eine saubere Dokumentation, gleich ganz verzichten zu wollen. Sie wird die Anforderungen des Managements an Liefertermin, Minimalfunktionalitäten oder Budgetplanungen nicht hundertprozentig erfüllen können.

Es gilt, die Vorteile der agilen Softwareentwicklung mit planerischer Sicherheit zu kombinieren. Unternehmen müssen Agilität zähmen, dann entfaltet sie ihre volle Wirkung. Den Entscheidern in Unternehmen steht dafür eine ganze Reihe von Instrumenten und Konzepten zur Verfügung. Die wichtigsten davon werden hier vorgestellt. Will das Management Agilität in den eigenen Reihen etablieren, kann es daraus die passenden Maßnahmen auswählen.

Das rechte Maß der Agilität

Wie das richtige Verhältnis von Planung und Flexibilität aussieht, ist von mehreren Faktoren abhängig. Selbst bei vergleichbaren Unternehmen – ähnliche Branche, ähnliche Anwendungslandschaften, ähnliche Vertriebswege, ähnliche Produkte – kann dieser Punkt auf unterschiedlichen Stellen der Skala liegen. Entscheidend ist die persönliche Einstellung wichtiger Projektbeteiligter zu dem Thema: Je agil-affiner sie sind, desto größer der agile Anteil in der Projektarbeit. Indikatoren sind auch die Größe des Projekts, seine Bedeutung, die Dynamik des Umfelds, die Unternehmenskultur und das Branchen-Know-how der Entwicklungsmannschaft.

Zunehmende Größe (von Projekt und Organisation), hoher Bedeutungsgrad der Anwendung und Klarheit der Anforderungen verschieben das optimale Verhältnis tendenziell in Richtung plangetriebener Verfahren. Darüber hinaus dürfen die Entscheider zwei weitere Faktoren nicht außer Acht lassen: Das sind der Zustand der Anwendungslandschaft und die Frage, ob ein anstehendes Projekt mit großer Wahrscheinlichkeit zu strukturellen Veränderungen der Anwendungslandschaft führt.

Die Suche nach dem optimalen Verhältnis von Planung zu Agilität kann zeitaufwändig und nervenaufreibend sein. Geeignete Werkzeuge und Konzepte können diese Suche abkürzen. Neben DevOps und Continuous Integration beziehungsweise Continuous Delivery bietet sich hier der Interaction Room als Konzept an.

Ein Raum mit grenzenlosen Möglichkeiten

Der Interaction Room ist ein Medium, über das Fach- und IT-Experten besser miteinander kommunizieren können. Das gelingt durch die offene, nicht IT-fixierte Darstellung von Prozessen. Sie erlaubt es auch den Vertretern aus den Fachabteilungen, sich in die Diskussionen einzubringen. Abbildung 1 zeigt ein Beispiel für so eine Art der Prozessbeschreibung. Im Interaction Room können Teams auch komplexe Prozesse mit einfachen Mitteln beschreiben und bewerten. Unternehmen können den Interaction Room grundsätzlich unabhängig von einem konkreten Vorgehensmodell einsetzen.

Der Interaction Room ist ein echter begehbarer Raum mit vier Wänden. Diese Wände haben tragende Funktionen: Auf ihnen visualisieren die Projektmitarbeiter Prozesse und stellen Projektdetails dar, hier können sie Probleme auf einen Blick erkennen. Im Interaction Room arbeitet ein interdisziplinäres Team aus Fach- und IT-Experten unter der Anleitung eines Moderators zusammen. Gemeinsam ermitteln sie in Abstimmungsrunden Lösungen für die zentralen Themen und Fragestellungen eines Projekts. Visualisiert wird dies durch die Zuordnung von Symbolen zu einzelnen Aspekten des Projekts. Mithilfe der Wertannotationen ergänzen die Beteiligten die erarbeiteten Modelle um weitere Informationen (Abb. 2).

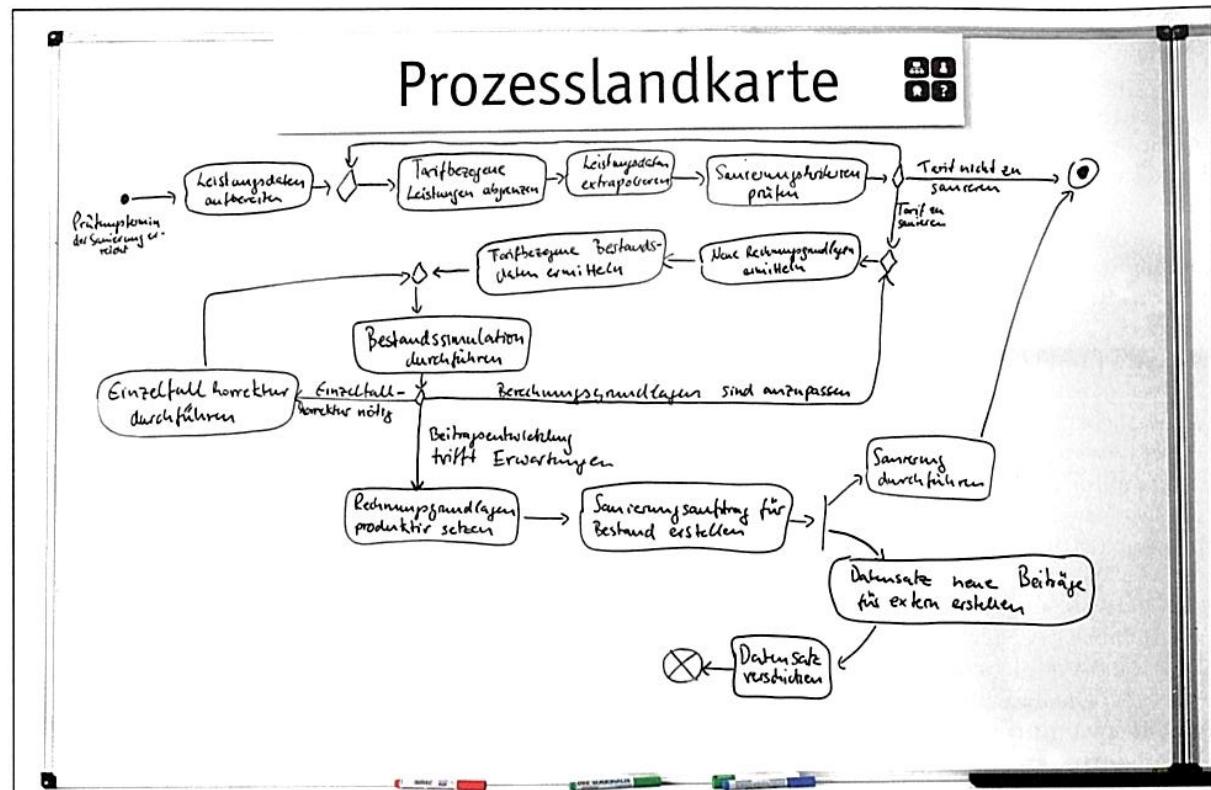
Der Interaction Room ist eine Methode, die das Interesse auf den Projektfortschritt lenkt und dazu beiträgt, dass die Vision von der zu entwickelnden Software kontinuierlich und durch alle Beteiligten gemeinsam weiterentwickelt wird.

Damit das Team im Interaction Room die gewünschten Ergebnisse erzielt, sind einige Voraussetzungen zu beachten: Die vier Wände repräsentieren jeweils einen zentralen Aspekt des Projekts. Eine Wand wird mit den

Agilität geht über Abteilungsgrenzen hinaus

Agilität fängt nicht erst hinter der Tür zur IT-Abteilung an, ganz im Gegenteil: Wenn Unternehmen neue Software für wettbewerbskritische Geschäftsprozesse entwickeln wollen, gleichgültig ob mit der unternehmensinternen IT oder mit externen Lieferanten, treffen Expertengruppen aus IT und Business aufeinander. Dies bedeutet nicht nur, dass sich hier unterschiedliches Fachwissen gegenübersteht; unterschiedliche Ziele, Arbeitsweisen und Vorstellungswelten erschweren häufig die Zusammenarbeit.

Abb. 1:
Prozessbe-
schreibung
(Quelle:
adesso AG)



Modellen der Geschäftsprozesse beschriftet, die das zu erstellende Softwaresystem unterstützen soll (die so genannte Prozesswand). Die zweite Wand dient zum Notieren fachlicher Objektmodelle (Objektwand). Die dritte nutzt das Team, um den Backlog zu notieren und den Projektfortschritt zu protokollieren (Statuswand). Auf der vierten Wand bilden sie die Integrationslandkarte ab (Integrationswand). Diese Landkarte gibt Auskunft darüber, welche existierenden Softwaresysteme mit dem zu erstellenden System integriert werden müssen. Damit die Projektmitglieder im Interaction Room problemlos zusammenarbeiten können, müssen sich

alle Beteiligten im Vorfeld über einige Grundsätze im Klaren sein.

Abstraktion: Wände sind, im Gegensatz zu elektronischen Dokumenten, endlich. Dies zwingt die Beteiligten dazu, sich auf das Wesentliche zu konzentrieren – ein gewollter Effekt der Arbeit im Interaction Room. So hat es sich in der Praxis bewährt, auf der Prozesswand nicht mehr als fünfzehn Prozessmodelle mit jeweils maximal fünfzehn Aktivitäten zu notieren. Details, die unnötig Ressourcen binden, werden ausgeblendet.

Wertorientierung: Eine entscheidende Frage für den Projekterfolg ist, wie die Projektmitarbeiter die wesentlichen und kritischen Teile eines zu erstellenden Softwaresystems identifizieren. In der Praxis gibt es eine Tendenz dazu, einfachen Zusammenhängen, die alle schnell verstehen, zu viel Zeit und Ressourcen zu widmen. Während alle gemeinsam die Personenstammdaten detailliert modellieren, schenken sie zentralen Geschäftsprozessen zu wenig Aufmerksamkeit. Die Antwort des Interaction Rooms auf dieses Thema ist die so genannte „Wertannotation“ von Prozess- und Objektmodellen. Abbildung 3 zeigt eine Auswahl der Symbole, die in verschiedenen Durchgängen von den Beteiligten an die Aktivitäten der Prozessmodelle geklebt werden. Development und Operations folgen zwar unterschiedlichen Leitlinien, müssen aber eng zusammenarbeiten.

Inkonsistenz: Ein unternehmensweit bedeutsames Projekt darf nicht zu einer IT-dominierten Veranstaltung geraten, in der der Beitrag aus den Fachbereichen unterrepräsentiert wird. Damit sich alle Beteiligten auf

	Ablösung		Nicht rechnergestützt
	Änderungshäufigkeit		Nutzungsspitze
	Baustelle		Rahmenbedingung
	Dublettengefahr		Schnelligkeit
	Externer Dienst		Schwierigkeit
	Geld		Sicherheit
	Häufigkeit		Ungewissheit des Teams
	Image		Ungewissheit
	Individuelle Ungewissheit		Unveränderlichkeit
	Mobilität		Wertschöpfung

Abb. 2: Wertannotationen (Quelle: adesso AG)

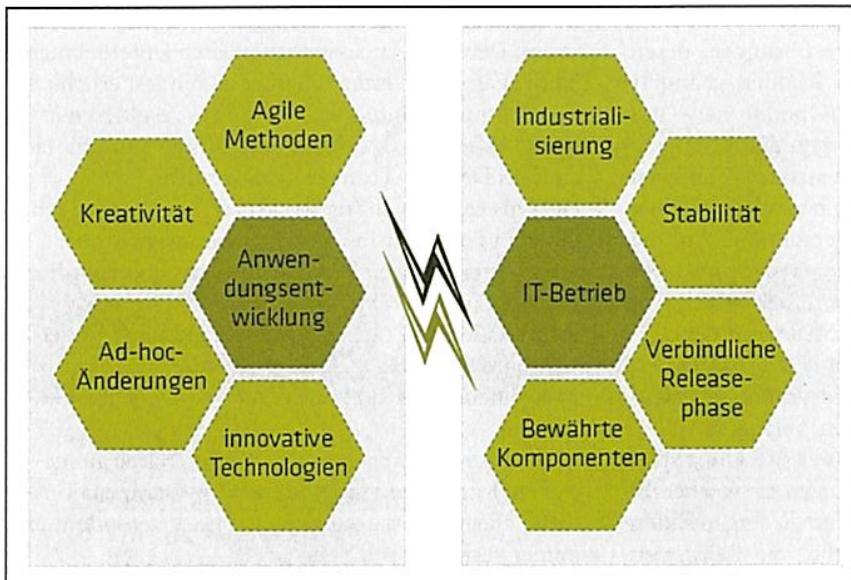


Abb. 3: Das Spannungsverhältnis zwischen Entwicklung und Betrieb (Quelle: adesso AG)

fachliche Inhalte konzentrieren, gibt der Interaction Room bei der Beschreibung der Prozessmodelle – ein Thema, bei dem IT-Experten häufig einen Know-how-Vorsprung haben – keine Syntax vor. Was immer den Beteiligten sinnvoll erscheint, wird in Abstimmung mit dem Interaction-Room-Moderator genutzt. Mögliche syntaktische Ungereimtheiten werden dabei in Kauf genommen.

Ungewissheit: In den frühen Phasen der Softwareentwicklung sind nicht alle fachlichen Zusammenhänge gleichermaßen klar, über einige Themen diskutiert das Team womöglich kontrovers. Damit die Beteiligten erkennen, welche Zusammenhänge sie noch nicht tief genug durchdrungen haben, müssen alle Teilnehmer des Interaction Rooms nach der einleitenden Diskussion die für sie kritischen Themen mit einem „Ungewissheitssymbol“ kennzeichnen. Auf diese Weise ermitteln sie, in welchen Bereichen zusätzliches Wissen nötig ist.

Der Interaction Room passt gut zu agilen Modellen aus dem Scrum-Umfeld. Das Fortschreiben von Aufwandsprognosen und das kontinuierliche Kontrollieren des Budgets (auf Basis von „Earned-Value-Analysen“) zähmen die Agilität. Er bildet einen verlässlichen kommerziellen Rahmen, innerhalb dessen die Verantwortlichen agile Softwareprojekte organisieren. Während der Interaction Room „vorne“ bei der Softwareentwicklung, im Zusammenspiel zwischen Anwender und Entwickler, sein Potenzial entfaltet, stehen jetzt Möglichkeiten im Fokus, die auch „hinten“, beim IT-Betrieb, für mehr Agilität sorgen.

Agil bis in den IT-Betrieb

Agil entwickeln bedeutet, in regelmäßigen kurzen Abständen Software zu produzieren: Software, die Unternehmen in den Betrieb – oder zumindest in produktionsnahe Umgebungen – überführen müssen. Dafür müssen sie die passenden Prozesse und Rahmenbedingungen schaffen. Im Folgenden wird der praktische Einsatz verschiedener Ansätze beschrieben, mit denen die in der Anwendungsentwicklung begonnene Agilität im Betrieb fortgesetzt wird. Dazu gehören die aus der Start-up-Welt stammenden DevOps – das Zusammenlegen von Entwicklung, Inbetriebnahme und Betrieb zu einer Organisationseinheit, aber auch der Einsatz weitestgehend automatisierter, kontinuierlicher Integrations- und Deployment-Prozesse wie Continuous Integration (CI) beziehungsweise Continuous Delivery (CD).

Der Begriff DevOps – „Development“ und „Operations“ – steht für eine neue Zusammenarbeit dieser beiden IT-Bereiche. DevOps sind im ersten Schritt ein organisatorisches Thema. Um die Vorteile zu realisieren, müssen IT-Verantwortliche vorhandene Strukturen und Verantwortungen anpassen.

webinale

29.05. – 02.06.2017 | Berlin
www.webinale.de



Int. PHP Conference

29.05. – 02.06.2017 | Berlin
www.phpconference.com



DevOpsCon

12.06. – 15.06.2017 | Berlin
www.devopsconference.de/de



BASTA!

25.09. – 29.09.2017 | Mainz
www.basta.net



JavaScript Conference

23.10. – 27.10.2017 | München
www.javascript-conference.com



Int. PHP Conference

23.10. – 27.10.2017 | München
www.phpconference.com



W-JAX

06.11. – 10.11.2017 | München
www.jax.de

Die einen – Softwareentwicklung (Development) – entwickeln unter Zeitdruck kreative Lösungen, deren Spezifikationen häufig noch durch Kunden geändert werden. Die anderen – IT-Betrieb (Operations) – garantieren stabile IT-Prozesse und sorgen dafür, dass zu festen Terminen neue Software nach strikten Vorgaben in Betrieb genommen wird. Häufig herrscht zwischen diesen Abteilungen mangelndes Verständnis für die Arbeitsweise und Zielsetzungen des Gegenübers; ein Mangel, der für langsamere Prozesse und erhöhten Abstimmungsaufwand sorgt. An dieser Stelle setzt die DevOps-Idee an: Anwendungsentwicklung und IT-Betrieb sind eins. Das komplette Team wird daran gemessen, dass der Service, den sie verantworten, verfügbar ist.

Insbesondere bei der agilen Softwareentwicklung spielen DevOps ihre Stärken aus: Was nutzen die wöchentlichen Sprints eines Scrum-Projekts, wenn die Organisation darauf ausgerichtet ist, dreimal jährlich ein Release zu veröffentlichen? Geboren wurde DevOps in Start-up-Unternehmen: Unternehmen, die ohne große Rücksichtnahme auf ihre IT-Historie agieren können. Diese Herkunft macht es für die Verantwortlichen schwierig, das Konzept eins zu eins auf etablierte Unternehmen mit tradierten Strukturen zu übertragen. Denn häufig arbeiten Entwicklung und Betrieb seit Jahrzehnten auf Basis derselben Aufgabenteilung mehr oder weniger zusammen.

Welche Organisationsform beziehungsweise welche Mischung von Organisationsformen für ein Unternehmen geeignet sind, können die Verantwortlichen nur durch genaue Betrachtung des Einzelfalls entscheiden. Aber es gibt einige Indikatoren, die anzeigen, ob die Tendenzen eher in Richtung DevOps oder in Richtung klassische CIO-Organisation gehen:

- Systeme, die sich nur langsam verändern und die kaum strukturellen Anpassungen ausgesetzt sind, sind kein Auslöser für die Einführung von DevOps-Strukturen.
- Das Gleiche gilt für Anwendungen, die Teil einer hochintegrierten IT-Landschaft sind. Hier ist es in al-

ler Regel schwierig, Änderungsreichweiten vorherzusagen. Denn bei Anpassungen müssen Unternehmen häufig ganze Landschaften oder zumindest erhebliche Teile komplett einführen. Die daraus resultierende Komplexität der Gesamtverantwortung kann für ein einziges DevOps-Team zu groß werden.

- DevOps eignen sich für Strukturen mit klaren Trennlinien in Form nur lose gekoppelter, asynchroner integrierter Dienste. Diese Voraussetzungen sind häufig in jüngeren Systemen gegeben. Dies trifft insbesondere auf oberflächenintensive und kundensichtbare Systeme am Anfang ihres Lebenszyklus zu. In so einem Umfeld können DevOps bestens funktionieren.
- Mobile Anwendungen sind von ihrer Natur aus geeignete Kandidaten für eine DevOps-Organisationsform; allein schon, weil sie sich schnell verändern und immer verfügbar sein müssen.

Die Herausforderung für viele IT-Verantwortliche liegt darin, innerhalb ihrer etablierten Strukturen ein neues Konzept der Zusammenarbeit zu implementieren. In der Praxis hat sich ein dreistufiges Vorgehen bewährt, mit dessen Hilfe Unternehmen DevOps schrittweise einführen, ohne die Anpassungsfähigkeit einer Organisation zu überfordern:

Stufe 1 – säen: „Gemeinsame Serviceverantwortung entwickeln“ – das ist das Motto der frühen DevOps-Einführung in Unternehmen. Die Grundlage bilden ein einzelner oder einige wenige geeignete Services. Hierfür bieten sich innovative Anwendungen aus den Bereichen Internet und Mobile Computing an, die mit marktgängigen Technologieplattformen (.NET, JEE) realisiert sind und etablierte Softwarearchitekturmuster verwenden. Komplexe Mainframe- und Legacy-Anwendungen sind eher ungeeignet.

Stufe 2 – pflegen: Auf Basis des in Stufe 1 konzipierten und erprobten DevOps-Blueprints rollen die Teams das Konzept weiter aus und verankern es tiefer in der Organisation. Dazu wählen die Verantwortlichen einen repräsentativen Querschnitt aus dem Portfolio der IT-Services aus. Für diese Auswahl werden Teams aufgebaut. Geeignet sind Prozesse, die von DevOps besonders profitieren. Das Geschäftsmodell und die Untersuchung der IT-Organisation liefern Anhaltspunkte für die Suche: An welchen Stellen müssen Entwickler häufiger Änderungen an der Unternehmenssoftware umsetzen? Wo muss das Unternehmen schnell mit Lösungen auf dem Markt sein? Die Antworten auf solche Fragen liefern Hinweise darauf, wo sich DevOps-Potenzial verbirgt.

Stufe 3 – ernten: Die Umstrukturierung des verbleibenden Teils des Serviceportfolios, der für das DevOps-Konzept geeignet ist, ist ein wichtiges Element der letzten Phase. Ziel ist nicht das hundertprozentige Durchdringen, sondern die am Geschäftsnutzen orientierte Umstellung mit Augenmaß. Auf der technischen Seite wird die Automation der Abläufe flächendeckend ausgebaut.

BizDevOps

DevOps-Strukturen sind, so mutig der Schritt weg von herkömmlichen CIO-Organisationen wirkt, erst der Anfang. Denn die zunehmende Bedeutung der IT, die wachsende IT-Kompetenz in den Fachabteilungen und die Kommoditisierung des Betriebs sorgen dafür, dass Umstrukturierungen nicht an den Grenzen der IT-Abteilung aufhören. Im nächsten Schritt wird es notwendig sein, Business, Development und Operations – BizDevOps – bei Softwareentwicklung und -betrieb zusammenarbeiten zu lassen. Schon heute ist diese Organisationsform in Start-ups verbreitet, die auf die Lean-Startup-Methode setzen. Über diese jungen Unternehmen hinaus werden BizDevOps in Zukunft eine wichtige Rolle spielen. Im eingangs erwähnten Interaction Room können auch IT-Entwicklung, -Betrieb und Business zusammenarbeiten; er ist ein Instrument, mit dem BizDevOps schon heute in der Praxis gelebt werden kann.

Die IT-Experten führen auf Basis einer standardisierten IT-Infrastruktur Self-Services für das Deployment von Entwicklungs- und Testumgebungen, einschließlich der erforderlichen Testdaten, ein.

Gewohnte Arbeitsweisen in Frage stellen, über Jahre erlernte Konzepte aufgeben, das reflexartige Denken in den Kategorien „wir“ und „die“ über den Haufen werfen: Die Einführung und das Leben von DevOps verlangen Mitarbeitern einiges ab. Mit dem beschriebenen Konzept zur Einführung gelingt Unternehmen der Übergang.

Continuous Integration und Continuous Delivery: Agilität ein System geben

Damit Unternehmen agil entwickelte Software überhaupt in der Frequenz in Betrieb nehmen können, in der die Entwicklung sie veröffentlicht, müssen die Verantwortlichen nicht nur die Organisation, sondern auch die Systeme anpassen. Ziel ist es, sowohl die Software als auch die Entwicklungs-, Test- und Produktivumgebungen automatisiert zu produzieren und bereitzustellen. Ein solch hohes Maß an Automatisierung war bisher in vielen Organisationen nicht notwendig. Für Unternehmen, die nur wenige Releases pro Jahr veröffentlichen, hat sich der damit verbundene Aufwand nicht gerechnet. Ein weiteres Argument, das gegen einen höheren Automatisierungsgrad sprach: Wenn zwischen zwei Softwareupdates mehrere Monate liegen, verändert sich mit hoher Wahrscheinlichkeit die Struktur der Zielumgebung. Die Experten müssten den automatisierten Prozess also erneut überarbeiten, damit er zu den neuen Strukturen passt.

Eher statische Rahmenbedingungen in der IT fördern also die „Handarbeit“. Diese personalintensiven Prozesse bringen einige Nachteile mit sich. Insbesondere sind sie stark von einzelnen Mitarbeitern abhängig. Unvorhergesehene Abwesenheiten, der Verlust von Kompetenzträgern oder unterschiedlich qualifizierte Fachleute sorgen für Probleme oder sogar für Prozessstillstand.

Wenn zwischen zwei Updates aber nicht mehrere Monate, sondern nur ein paar Tage oder sogar nur Stunden liegen, müssen die Verantwortlichen die IT-Betriebsprozesse grundlegend überdenken. Jetzt sind Installationsroutinen gefragt, die permanent prüfen, ob veränderte Software in Betrieb genommen werden kann. Diese Idee liegt Continuous Integration und Continuous Delivery zugrunde. Im Gegensatz zu den ersten Ansätzen der Automatisierung mit ihrem begrenzten Funktionsumfang bieten die aktuellen Lösungen inzwischen sehr vielfältige Möglichkeiten: von der Adressierung heterogener Infrastrukturen bis hin zum automatisierten Bereitstellen von Umgebungen und deren Integration in die vorhandene IT-Infrastruktur (Softwarelösungen wie Puppet oder Chef). Dazu gehören auch Tools für die Umsetzung von Monitoring- und Logging-Mechanismen, für das automatisierte Testen, für die Verwaltung von Softwarekomponenten (zum

Beispiel mit der Lösung Nexus) und für das Orchestrieren des gesamten Prozesses (beispielsweise das Softwaresystem Jenkins).

Automatisierung beschleunigt die Prozesse in der IT nicht nur, sie erleichtert auch die Fehlersuche. Sie vermeidet die Probleme, die durch das unterschiedliche Konfigurieren von Entwicklungs-, Test- und Produktivumgebungen entstehen können. Diese Fehlerquelle kann das Team also ausschließen. Das Bereitstellen von Infrastruktur wird zu einem Prozess, der mit den gleichen Methoden und zum Teil sogar mit den gleichen Werkzeugen wie die Softwareentwicklung unterstützt werden kann. Beispiele dafür sind die Versionierung von Code für die Infrastruktur oder das Entwickeln von Design Patterns für bestimmte Infrastrukturaufgaben, wie das Aufsetzen eines Webservers.

Mit den bisher geschilderten Konzepten und Werkzeugen gelingt es Unternehmensentscheidern, in der eigenen Organisation den richtigen Ausgleich zwischen agilen und traditionellen Entwicklungskonzepten zu finden. Aber häufig stehen sie auch vor der Aufgabe, Dienstleister in agile Projekte einzubinden. Der nächste Abschnitt zeigt, wie das gelingen kann.

Budgets in agilen Projekten: die Quadratur des Kreises?

Agiles Denken reicht nicht nur über Abteilungs-, sondern auch über Unternehmensgrenzen hinaus. Denn setzen Unternehmen bei der Softwareentwicklung auf externe Kapazitäten, ist es unabdingbar, dass diese in gleicher Weise an der agilen Entwicklung schlanker Software interessiert sind. Agile Entwicklungsverfahren reduzieren zugunsten einer rasch einsetzbaren Software die Spezifikationen auf ein Minimum und passen sich flexibel an neue Erkenntnisse, neue Prioritäten und neue Anforderungen an. Damit bieten sie zwar die Chance auf besonders effiziente Projekte, bereiten aber den Einkaufs- und Controlling-Abteilungen Kopfzerbrechen: Wie sollen sie etwas budgetieren, das gar nicht genau spezifiziert ist? Wie sollen sie auf dieser Grundlage die Kosten für ein ganzes Softwareprojekt planen? Eine Berechnung nach Aufwand ist für die Auftraggeber äußerst problematisch, da die Kosten dabei theoretisch ins Unendliche steigen können. Auf einen Festpreis wiederum wird sich kaum ein Lieferant einlassen, denn dieses Modell funktioniert für ihn nur, wenn während des Projekts keine oder zumindest nur sehr wenige Änderungen erforderlich sind. Dieses Dilemma ist der Grund dafür, dass es in der Praxis de facto nur sehr selten zu echten agilen Softwareprojekten kommt. Diese gibt es fast nur bei internen Projekten, die ganz anderen Regeln unterliegen. Beauftragen die Verantwortlichen dagegen einen Dienstleister mit der Softwareentwicklung, sind Planungs- und Sicherheitsbedürfnisse der involvierten Parteien meist stärker als alle guten agilen Absichten.

Das Thema Budgetierung steht also der agilen Idee im Weg. Erst wenn es zur Zufriedenheit aller Beteilig-

ten gelöst ist, kann sich die agile Idee weiter ausbreiten. Es klingt ein bisschen wie die Quadratur des Kreises. Gefragt sind intelligente Konzepte, die die unterschiedlichen Interessenlagen von Fach- und IT-Abteilungen des Auftraggebers und seines Lieferanten ausbalancieren. Einen Ansatz dafür bieten die so genannten „Shared pain/Shared gain“-Modelle. Die zentrale Idee dahinter ist, dass sich Auftraggeber und Dienstleister als Projektpartner die budgetären Risiken und Chancen der Agilität teilen.

Ein Beispiel dafür ist das Vorgehensmodell „adVANTAGE“, das es erlaubt, komplett agile Projekte zu budgetieren und die Budgets zu kontrollieren. Dazu skizziert der Auftraggeber zunächst seine Erwartungen an die Software, indem er unter anderem ihren Zweck, ihr Einsatzgebiet und ihre Anwender beschreibt. Daraus abgeleitet werden dann in so genannten User Stories die zu entwickelnden Features definiert – unscharf, aber ausreichend, um den Aufwand und die Dauer ihrer Entwicklung schätzen zu können. Die Zusammenfassung aller Schätzungen ergibt ein vorläufiges grobes Gesamtbudget und umreißt einen Zeitrahmen. Sind alle Anforderungen geschätzt, muss der Auftraggeber sie priorisieren: Welche Stories sind unbedingt nötig, um live gehen zu können? Und welche lassen sich zur Laufzeit nachreichen?

Die so priorisierte Anforderungsliste bearbeitet der Dienstleister anschließend in Iterationen, den Sprints. Vor dem zweiten und jedem weiteren Sprint treffen sich Auftraggeber und Dienstleister und nehmen den letzten Sprint ab. Sie legen fest, welche User Stories in der nächsten Iteration umgesetzt werden sollen. Bringt der Auftraggeber dabei neue User Stories ein, so schätzt das Planungsteam den Aufwand dafür gemeinsam ein. Soll das Projektbudget dafür nicht erhöht werden, ersetzt die neue Story eine niedriger priorisierte mit gleichem Umfang. Den Streichkandidaten bestimmt dabei der Auftraggeber. Die Rahmenbedingungen wie Zielbudget, Zieltermin und Schätzung einzelner Features sind während des gesamten Projektverlaufs für alle Beteiligten transparent und dienen ihnen als Leitfaden. Kommt es zu Änderungen durch die Hinzunahme oder das Weglassen eines Features, werden diese Daten entsprechend angepasst.

Wie aber sollen die Partner mit Abweichungen vom ursprünglich geschätzten Budgetrahmen umgehen? Hierfür bauen sie in das Projekt einen „Unwissenheitskorridor“ ein; ein spezieller Mechanismus, mit dem sich Auftraggeber und Auftragnehmer das Risiko teilen, das mit den groben Schätzungen einhergeht. Wird die angesetzte Zeit überschritten, schlagen für die zusätzlich nötigen Tage deutlich geringere Sätze zu Buche. Diese sind für den Dienstleister gerade noch kostendeckend, lohnen sich aber nicht mehr wirklich. Mit diesem Konstrukt ist sichergestellt, dass Auftraggeber und Auftragnehmer gemeinsam gewinnen, wenn sie möglichst nahe am Budgetrahmen bleiben, und gemeinsam verlieren, wenn sie ihn überschreiten.

IT sitzt jetzt mit am Tisch

Die Ausführungen zeigen: Die Art und Weise, wie Unternehmen Software entwickeln, betreiben und nutzen, verändert sich im Augenblick grundlegend. Ideen, die noch vor wenigen Jahren hippen Start-ups vorbehalten waren, halten Einzug in etablierte Großunternehmen. Damit ändert sich nicht nur die Arbeitsweise von IT-Abteilungen, sondern auch die Bedeutung, die IT-Themen und damit der CIO innerhalb eines Unternehmens einnehmen.

Für junge Firmen mit vergleichsweise neuen Geschäftsmodellen ist IT häufig das zentrale Produktionsmittel. Im Gegensatz dazu wird in alteingesessenen und etablierten Unternehmen IT oft noch auf die Rolle des reinen Servicebringers beschränkt. Hier wissen die einzelnen Fachbereiche genau, was sie wollen. Sie sind es, die die Merkmale der Lösung festlegen. Die EDV-Experten müssen dann nur noch bauen oder beschaffen.

Dieses Servicebringermodell funktionierte auch mehr oder weniger gut. Zumindest solange es nicht erforderlich war, disruptive Innovationen auf der Basis von IT umzusetzen. Aber neue digitale Produkte, neue Vertriebswege, neu fragmentierte Geschäftsprozesse, neue Anforderungen der Kunden an das Tempo, in dem das Unternehmen Innovationen umsetzt, oder neue Formen der Kooperation mit Partnern erfordern genau das: IT-gestützte Innovationen, die etablierte Geschäftsmodelle ablösen oder zumindest ergänzen. Dann wird IT zum Business Enabler. Das bedeutet, dass IT-Abteilungen noch enger mit Fachbereichen zusammenarbeiten müssen, um das Potenzial von Technologien auszuschöpfen. Sie müssen sich in Märkte einarbeiten, um essenzielle Anforderungen zu verstehen und zu bewerten. Auf der anderen Seite sind Fachbereiche dazu aufgefordert, die Chancen neuer Technologien zu beurteilen.

Das gemütliche Modell der langsamen und kontinuierlichen Verbesserung von Produkten und Prozessen kommt immer mehr an seine Grenzen. Die Bereitstellung von IT muss flexibler und dynamischer werden. Damit wird der Einfluss des CIO und seines Teams immer weniger an der Tür zur IT-Abteilung aufhören. Denn die Technologien und die Infrastruktur, die die Experten bereitstellen, werden immer entscheidenderen Einfluss über Erfolg oder Misserfolg haben. Entsprechend wird sich die Rolle des CIO weiter ändern: vom reinen Erbringer zum Gestalter von Geschäftsprozessen.



Prof. Dr. Volker Gruhn gründete 1997 die adesso AG mit und ist heute Vorsitzender des Aufsichtsrats. Er ist Inhaber des Lehrstuhls für Software Engineering an der Universität Duisburg-Essen. Seine Forschungsschwerpunkte in diesem Bereich liegen auf mobilen Anwendungen und der Auseinandersetzung mit den Auswirkungen der digitalen Transformation, insbesondere der Entwicklung und des Einsatzes von Cyber-Physical Systems. Prof. Dr. Gruhn ist Autor und Koautor von über dreihundert nationalen und internationalen Veröffentlichungen und Konferenzbeiträgen.

Agilität – eine Grundvoraussetzung für funktionierende Digitalisierung

Stillstand ist der Tod

Software agil entwickeln – ja oder nein? Es ist tatsächlich merkwürdig, dass sich Unternehmen diese Frage immer noch stellen. Um es vorweg zu nehmen: Eigentlich bleibt ihnen keine Wahl, denn Agilität ist schon lange kein Trend und keine Mode mehr. Der Artikel zeigt an einem Beispiel, wie ein Unternehmen agil werden kann.

von Martin Schmitz-Ohrndorf

Unternehmen, die die notwendigen Weichenstellungen bisher nicht vorgenommen haben, verzeichnen bereits jetzt spürbare Nachteile. Der Grund dafür ist das, was aktuell unter dem Stichwort „digitale Transformation“ in der Fachwelt und der breiten Öffentlichkeit diskutiert wird. Auch die digitale Transformation ist – ähnlich wie agile Entwicklung – kein Hype, sie passiert bereits.

Neue Aufgabe der Unternehmen

Das betrifft nicht nur technische Aspekte wie das „Internet of Things“ oder „Cyber-Physical Systems“. Inzwischen hängen alle maßgeblichen Geschäftsprozesse von IT ab. Auch Produkte und Dienstleistungen sind mehr und mehr rein digital oder haben zumindest entscheidende digitale Komponenten, was fundamentale Auswirkungen auf Unternehmen hat. Bisher konnten sie ihren Handlungsbedarf strukturiert bewerten und den eigenen Fahrplan an Optimierungspotenzialen, Änderungsbereitschaft und den eigenen Geschäftsmodellen orientieren; aber jetzt mischen neue Player die Märkte auf, und mit ihnen verändern sich die Angebote auf dem Markt und die Ansprüche der Kunden. Diese Entwicklung erfordert ein grundsätzlich anderes Verhalten von den Unternehmen. Das zeigt sich an vielen Schnittstellen zwischen Kunden und Unternehmen, beispielsweise im Kundenservice. Von Kunden zu verlangen, auf unternehmensinterne Anforderungen und Prozesse Rücksicht zu nehmen – den Service auf wenige Kommunikationskanäle oder kurze Öffnungszeiten zu beschränken – wird zunehmend schwerer durchzusetzen sein. Produkte und Dienstleistungen müssen sich klar an den Bedürfnissen der Kunden orientieren. Ändert der Kunde seine Anforderungen, muss sich das Unternehmen flexibel und dynamisch anpassen. Passt es sich nicht an und bringt es kein überzeugendes Angebot auf den Markt, wird der Kunde sich Alternativen suchen. Vorbei sind also die

Zeiten, in denen Unternehmen von ihren Kunden Verständnis für die eigenen internen Strukturen einfordern konnten.

In vielen Branchen machen es Start-ups vor. Sie haben ihre Kunden mit kleinem Aufwand viel besser verstanden als die etablierten Unternehmen. Ihre Erfolge zeigen: Der direkte Kontakt zum Kunden, sein direktes Feedback und die schnelle Reaktion darauf sind erfolgskritisch. Eine große Anzahl Start-ups scheint sich langfristig zu etablieren. Natürlich gibt es Konsolidierung. Aber: Viele Ideen scheinen auch Bestand zu haben; das Angebot fächert sich auf. Das ist gut für den Kunden, gut für den Markt und auch gut für etablierte Unternehmen – denn es gerät etwas in Bewegung. Dieser Handlungsbedarf ist im Wesentlichen im Business-to-Consumer-Bereich entstanden, er wirkt sich aber unmittelbar auch auf Business-to-Business-Märkte aus.

Zurück zur Agilität. Software ist – mehr denn je – der Schlüssel zu digitalen Services und Produkten. Die IT muss maßgeblich gestalten und ihr Wissen um dynamische Lösungen in die Entwicklung neuer Angebote einfließen lassen. Die gängigen Grenzen zwischen Fachbereichen und IT verschwinden endgültig. Der Kunde beziehungsweise Nutzer wird in Projekte einbezogen; wirksame Customer Experience und daraus folgende User Experience entsteht im direkten Kontakt mit den Anwendern. Dabei ist Zeit ein kritischer Faktor. Unternehmen müssen Neues schnell auf den Markt bringen und bei fehlender Wirksamkeit auch schnell wieder „einstampfen“ können. Diese Anforderungen an das „neue Verhalten von Unternehmen“ sprechen eindeutig für agile Konzepte.

Die Rolle der Methoden

Welche Rolle spielen in diesem Zusammenhang dann konkrete agile Methoden wie Scrum? Ein Framework wie Scrum gibt Unternehmen, die sich mit digitalen Services und Produkten (also mit Softwareentwicklung)

beschäftigen, einen Rahmen. Dieser ist eine nahezu optimale Grundlage für die Auseinandersetzung mit den Herausforderungen der digitalen Transformation: Schnelligkeit, Dynamik, Flexibilität, Orientierung am Kunden und dem Nutzererlebnis sowie Übereinstimmung mit gemeinsam verstandenen Zielen. Das sind grundlegende agile Werte (wie sie beispielsweise im Agilen Manifest definiert wurden), und sie beschreiben auch die richtige Einstellung für den Umgang mit der digitalen Transformation.

Viele der großen Player quer durch alle Branchen haben diese Zusammenhänge inzwischen verstanden – und doch sind entsprechende Change-Projekte nur schwer umzusetzen. Der Grund dafür: Viele Unternehmen befinden sich in einer Phase, in der die mühevoll erlerten und etablierten klassischen Prozesse gerade erst Wirkung zeigen; eine Phase, in der Qualität und Verlässlichkeit der IT gerade steigt. Zementierte Prozessvorgaben, bindende Quality Gates und vorgegebene Toollandschaften sind jedoch blockierende Faktoren in einem Change-Projekt hin zu mehr Agilität. Die Experten müssen sie deshalb konstruktiv in Frage stellen und in großen Teilen überarbeiten. Es ist nachvollziehbar, dass in den Unternehmen auf allen Ebenen die Angst vor totalem Kontrollverlust in der IT immens ist. Wie die Verantwortlichen ein Change-Projekt in einem etablierten Unternehmen gestalten können, welche Rolle dabei ein Framework spielen kann und wie es gelingt, 150 IT-Experten auf agile Prinzipien einzuschwören, zeigt folgender Exkurs.

Exkurs: Change-Projekt in einem Großunternehmen

Zu den Anforderungen des auf Dienstleistungen spezialisierten Unternehmens passte lange Zeit die Softwareentwicklung nach „klassischen“ Vorgaben: ausführliche Spezifikationen, phasenorientiertes Vorgehen, große Releases. Aber alle Beteiligten stellten fest, dass dieses Vorgehen immer häufiger zum Hemmschuh wurde. Aufwendige Abstimmungsprozesse, zahlreiche Änderungen der Anforderungen, unzureichende Koordination zwischen den nach IT-Kompetenzen aufgestellten Abteilungen und eine geringe Termintreue sorgten für Unzufriedenheit und damit Handlungsbedarf. Agile Prozesse, abgestimmt auf die Anforderungen einer großen Organisation, sollten Abhilfe schaffen. Aber wie können 150 IT-Experten auf agile Methoden eingeschworen werden? Wie können komplexe Prozesse mit eng verzahnten Systemen, die mehrere Teams betreffen, agil gestaltet werden? Unterstützt durch den IT-Dienstleister adesso machten sich die IT-Verantwortlichen daran, Agilität im großen Maßstab umzusetzen.

Individualentwicklung für ein individuelles Geschäft

Die IT-Teams fungieren aktuell noch als interner Dienstleister des ganzen Konzerns. Ihre Auftraggeber sind Fachabteilungen oder auch das Marketing. Für vie-

In den Unternehmen ist die Angst vor totalem Kontrollverlust in der IT auf allen Ebenen immens.

le der intensiv IT-gestützten Prozesse werden intern eigene Lösungen entwickelt, da auf dem Markt erhältliche Softwareprodukte weder den geforderten Funktionsumfang noch die benötigte Funktionstiefe erreichen. Dazu gehören fachliche Standardthemen wie Abrechnung und Service, das Kerngeschäft des Konzerns, aber auch neue innovative digitale Endkundenprodukte. Dabei hört der Service nicht an den Landesgrenzen auf. Auch für ausländische Gesellschaften entwickeln die Fachleute Lösungen neu beziehungsweise weiter.

Die IT-Organisation folgte dabei in der Vergangenheit einer weit verbreiteten Aufstellung: Anforderungs-, Entwicklungs- und Testbereich waren hintereinander geschaltet. Die Teams arbeiteten in Form eines Phasenmodells inklusive Rückkopplungsschleifen zwischen den Phasen Planung, Analyse, Entwicklung, Test und Produktion. Ungewöhnlich an der Organisation: Die Teams entwickelten bis zu fünf Releases gleichzeitig, die jeweils um eine Phase verschoben waren. Während das erste Release noch in der Planung steckte, war das zweite bereits in der Analyse, das dritte in der Entwicklung und so weiter. Über 150 IT-Fachleute waren so in einer Kette von der Anforderungsanalyse bis hin zum Testing organisiert.

In dieser Kette knirschte es jedoch. Dabei gab es nicht das eine Ereignis oder die eine Kennzahl, die eine Diskussion über den Softwareentwicklungsprozess anstieß. Eine ganze Summe von kleineren und größeren Faktoren ließ die Entscheider hellhörig werden. Insbesondere das Vertrauen der Mitarbeiter in die Releaseplanung war nicht so groß, wie sich das Management das wünschte. Häufig änderten sich die Anforderungen an das neue Release – Anpassungen, die die Teams aufgrund der engen Verzahnung der Abläufe vor große Herausforderungen stellten – und Termine mussten immer wieder angepasst werden.

Einerseits viel Aufwand, der in Planung und Abstimmung investiert wurde, andererseits ein Ergebnis, das von vielen als unbefriedigend empfunden wurde: Die IT-Verantwortlichen entschlossen sich dazu, die Arbeit ihrer Teams neu zu organisieren und trugen die Diskussion zu diesem Thema in das Management. In dieser Runde wurden Alternativen geprüft: noch mehr Planung, eine noch engere Begleitung der Projekte. Am Ende entschlossen sich die Entscheider aber dazu, einen auf den ersten Blick genau anderen Weg zu gehen: mehr dezentrale Abläufe und mehr Selbstverantwortung. Me-

Zwischen der ersten Idee und dem ersten agil entwickelten Release lag ein umfangreicher Change-Management-Prozess.

Methoden der agilen Softwareentwicklung sollten die vorhandenen Unzulänglichkeiten abstellen.

Von vornherein war klar: Für dieses Projekt können die Entscheider agile Methoden wie Scrum nicht eins zu eins umsetzen, denn diese Ansätze sind häufig auf die Teamebene zugeschnitten und führen hier auch zum Ziel. Aber wenn es darum geht, teamübergreifende Abhängigkeiten darzustellen, war das Vertrauen in diese Konzepte zu gering. Gesucht wurde deshalb eine Variante der Agilität, die auch im großen Maßstab Sicherheit vermittelt. Die Verantwortlichen entschieden sich für das Scaled Agile Framework (SAFe).

Agilität entsteht im Kopf, nicht auf dem Organigramm

Zwischen der ersten Idee und dem ersten Release, das agil entwickelt wurde, lag ein umfangreicher Change-Management-Prozess. Denn auch in diesem Punkt war sich das IT-Management einig, primär geht es bei Agilität darum, die Einstellung der Beteiligten zu verändern.

Bei der Gestaltung und Umsetzung dieses Prozesses arbeiteten die Verantwortlichen eng mit externen Consultants des IT-Dienstleisters adesso zusammen. Am Anfang stand adesso beratend zur Seite, mit fortschreitendem Projekt arbeiteten die Experten immer häufiger als Coaches und unterstützten die Teams in ihrer tagtäglichen agilen Arbeit.

Schon sehr früh konnten so externes Wissen und externe Erfahrung in das Projekt eingebracht werden. Noch bevor die Entscheider den eigentlichen Roll-out planten, wurden im Kreis der IT-Führungskräfte Workshops zum Thema Agilität durchgeführt. Ihr Zweck war es, auf der Ebene des mittleren Managements Verständnis aufzubauen und unterschiedliche agile Ansätze kennenzulernen. In dieser Frühphase besuchten Vertreter des Führungskreises auch andere Unternehmen, die bereits agil entwickelten.

Agilität muss agil eingeführt werden

Schnell kristallisierte sich bei den Projektverantwortlichen eine Vorstellung davon heraus, wie der Weg der Einführung aussehen kann. Es war aber auch allen bewusst, dass ein starrer, konsequent umgesetzter Plan nicht zur agilen Idee passt. Das Einführungsprojekt wurde deshalb auch als Möglichkeit betrachtet, die agile Idee vorzuleben. So wurden gezielt nur einige Meilensteine definiert, die das Team zeitlich und inhaltlich er-

reichen wollte; den Weg dazwischen passten sie flexibel an Situation und Personen an.

Dieser Weg begann für die über 150 IT-Teammitglieder mit einem gemeinsamen Event, das den Teilnehmern nachhaltig im Gedächtnis blieb. Hier kamen alle Beteiligten außerhalb der Tagesroutine und auch außerhalb der eigenen Räumlichkeiten zusammen, hier wurde der Grundstein der agilen Offensive gelegt. Der Aufwand, der für die Organisation und Durchführung der Veranstaltung betrieben wurde, zeigte allen von Beginn an deutlich: Das Unternehmen meint es mit der Agilität ernst.

Nach dieser zentralen Veranstaltung ging es daran, ein Bewusstsein für Agilität zu entwickeln. Welche Ideen, Werte und Konzepte verbergen sich dahinter? Wie kann ich agil arbeiten? Die Antworten auf diese Fragen erarbeiteten die Teams auf ungewöhnliche Art und Weise: spielerisch in Workshops. Im ersten Schritt ging es nicht um Softwareentwicklung oder Releaseplanung, sondern um das Bauen von LEGO-Türmchen oder das Planen von Haushaltstätigkeiten. Anhand von einfachen Aufgaben – ohne die Belastung des täglichen Arbeitsens – setzten sich die Teams mit agilen Prinzipien auseinander. So lernten die Experten „live“, dass sogar ein Turm aus Bauklötzchen sehr agil aufgebaut werden kann, oder ganz klassisch nach der Wasserfallmethode. Am Ende hatten die Beteiligten die Unterschiede zwischen den einzelnen Konzepten direkt vor Augen. Zahlreiche solcher Übungen, die Aspekte der Agilität vermittelten, dachten sich die internen Projektverantwortlichen gemeinsam mit den adesso-Beratern aus. Pro Termin wurden zirka fünfzehn Mitarbeiter geschult, die Gruppen waren ganz bewusst bunt gemischt. Java, Host und SAP, Tester, Entwickler und Requirements Engineers – die Projektorganisation achtete darauf, dass jeder einzelne Kollege möglichst mit Personen in einer Schulung saß, mit denen er in der täglichen Arbeit bisher wenig zu tun hatte. Wissensvermittlung und Teambuilding gingen so Hand in Hand.

Auf die Workshops folgten Basisschulungen, die die Grundlagen realer agiler Methoden vermittelten. Diese Schulungen waren für alle Teilnehmer gleich, die Spezialschulungen für Rollen wie Scrum Master, Teammitglied oder Product Owner bauten dann anschließend darauf auf. Dort ging es darum, den Beteiligten Details über Scrum auf Teamebene und das übergeordnete Scaled Agile Framework zu vermitteln. Dieses Framework war die Antwort auf die eingangs gestellte Frage, wie komplexe Prozesse, die mehrere Teams betreffen, agil gestaltet werden können. SAFe ist ein Rahmenwerk für skalierte Agilität und liefert dem Konzern einen bewährten Rahmen, der dafür sorgt, dass Agilität auch in Größenordnungen von 150 und mehr Mitarbeitern funktioniert. Diesen Rahmen passten die Projektbeteiligten so an, dass er optimal zu den Unternehmensforderungen passte.

Bei diesen Spezialschulungen unterstützten die adesso-Experten nicht nur die Vorbereitung, sie waren auch bei den Terminen dabei. So schulten erfahrene adesso-

Coaches gemeinsam mit internen Trainern zukünftige Product Owner, Scrum Master oder Entwickler aus DBT-Teams für ihre neuen Rollen – ein Konzept, das sich rückblickend bewährt hat. So konnten die Projektverantwortlichen während der Schulungsphase die Brücke zwischen agiler Theorie und Praxis schlagen.

Auch wenn das Schulungsprogramm prall gefüllt war: Neben all diesen Terminen ging die Arbeit für das IT-Team weiter, Releases mussten geplant und neue Funktionen entwickelt werden. Den zur Verfügung stehenden Schulungszeitraum nutzte die gesamte IT-Führungsmannschaft dafür, in enger Abstimmung mit den einzelnen Mitarbeitern und dem Betriebsrat, Teams zu planen und zu besetzen. So entstanden die Strukturen, in denen Agilität künftig umgesetzt werden sollte.

Während die Schulungen zu Ende gingen, fing die Zeit der konkreten agilen Softwareentwicklung an. Dabei wählten die Manager keine Big-Bang-Strategie, sondern entschieden sich dafür, Team für Team auf Agilität umzustellen. Dies hatte zwar zur Folge, dass in der IT-Abteilung für einen längeren Zeitraum zwei Konzepte – Agilität und „Phasenorientierung“ – gelebt und koordiniert werden mussten. Auf der anderen Seite verschaffte dieses schrittweise Vorgehen den Verantwortlichen aber den nötigen Spielraum, um gemeinsam mit externen Coaches den Weg jedes einzelnen Teams intensiv zu begleiten.

Um diesen Übergang von Phasenorientierung zu Agilität auf Teamebene erfolgreich zu gestalten, setzten die Experten auf das Konzept des so genannten „Preparation Sprint“. Mit diesem ersten Sprint verfolgten sie dabei zwei Ziele: Einerseits diente er als weitere Übung und sorgte dafür, dass jeder Einzelne die Scrum-Grundlagen noch einmal auffrischen konnte. Andererseits sollte sich das ganze Team während dieser ersten Sprint-Phase so finden und koordinieren, dass es im Anschluss agil durchstarten kann. Somit diente dieser erste Schritt der Selbstorganisation. Alle Teammitglieder suchten die Antwort auf die Frage: „Was ist noch zu tun, um ab dem nächsten Sprint ‚echt‘ zu arbeiten?“

Grenzen überwinden – Fortschritt erzielen

Noch läuft nicht alles rund, aber vieles läuft deutlich besser als vorher. Dank der agilen Teams konnte der Konzern die Grenzen zwischen den Abteilungen innerhalb der IT überwinden – ein wichtiger erster Schritt. Es heißt nun nicht mehr: „die Tester und wir“ oder „die Entwickler und wir“. Stattdessen arbeiten alle zusammen auf ein gemeinsames Ziel hin.

Ein wichtiger Baustein sind auch die Release-Planning-Events. Auf ihnen treffen sich alle Beteiligten und planen das kommende Release. Die Teams koordinieren ihre Arbeit eigenständig, und jeder Einzelne wird zu seiner Einschätzung gefragt; Gegenmeinungen werden explizit berücksichtigt. Am Ende entsteht so eine Planung, die von allen verstanden und getragen wird.

Dieses Projekt zeigt auch, dass agile Softwareentwicklung nicht nur in homogenen technologischen Umge-

bungen mit nur einem Produkt sinnvoll ist. Der Konzern arbeitet mit einer sehr heterogenen IT-Landschaft und hat große SAP-, Java- und Mainframe-Technologie-Teams. Aber Agilität zahlt sich für alle Bereiche aus.

Fazit

Der aktuelle Zwischenstand lautet: Es existiert eine agile IT-Abteilung in einem nicht agilen Unternehmen. Das Projektteam hat also erreicht, was erreicht werden sollte. Die Optimierung der IT-internen Abläufe ist voll wirksam. Viel wichtiger jedoch: Bei den Verantwortlichen hat ein Veränderungsprozess begonnen. Review-meetings ohne Nutzer oder gar ohne Kunden werden von allen Beteiligten laut beklagt und deren Anwesenheit eingefordert. Anforderungen der Fachabteilungen erörtern die Experten untereinander konstruktiv und auf Augenhöhe. Das Thema der „agilen Führung“ ist nun auf Managementebene – auch über die IT hinaus – angekommen. Die Grenze zwischen IT und Fachbereichen wurde aber noch nicht aufgelöst, hier besteht noch Handlungsbedarf.

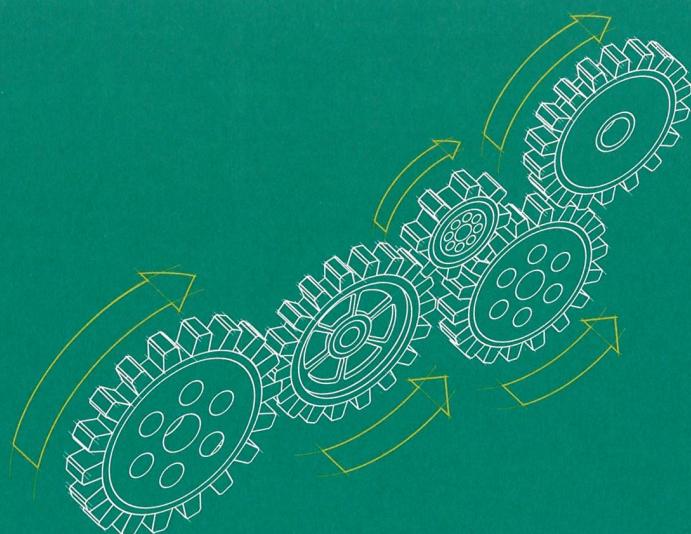
Und wie steht es mit der skalierten Agilität? Ist SAFe ein sinnvolles Modell? Der Exkurs zeigt: Es hilft beim Start. Die Einführung eines klassischen Scrums als teambezogenes Framework hätte nicht wie SAFe alle Beteiligten mit ins Boot geholt, der Beginn der Transition wäre um ein Vielfaches schwieriger gewesen und der agile Ansatz höchstwahrscheinlich auf der Teamebene stecken geblieben. Genau so war es bei dem Unternehmen des beschriebenen Beispiels bei einem vorangegangenen Versuch schon einmal geschehen. Stand-up-Meetings und Boards waren bereits da, wirkliche Agilität jedoch nicht.

Die Antwort ist aber auch: SAFe baut neue Hürden auf, da Konformität mit einem Modell für ein digitalisiertes Unternehmen eigentlich kein Wert mehr sein kann. Es ist für ein Change-Projekt hin zur Agilität aber eben auch entscheidend, den Schwung nicht zu verlieren und möglichst viele der Protagonisten zu überzeugen. Das ist mit SAFe, aber auch LeSS (Large Scale Scrum) oder irgendeinem anderen Ansatz möglich, der bei der Selbstorganisation hilft – solange es von innen kommt.

Und die Digitalisierung? Auch hier gibt es erste Erfolge. Neue Services wurden in sehr kurzen Zeiträumen geschaffen, beim Nutzer nicht Erfolgreiches wurde schnell wieder reuelos verworfen, und wichtige Erkenntnisse wurden gesammelt. Die Teams verstehen zunehmend besser, welche Bedeutung die Softwareentwicklung für die Marktchancen von Unternehmen hat und beteiligen sich aktiv. Aber es bleibt viel zu tun – und der Druck zur Digitalisierung steigt stetig.



Martin Schmitz-Ohrndorf ist Principal Consultant und Competence Center Leiter Consulting beim IT-Dienstleister adesso AG.



Agile Methoden in IT-Projekten

Auf dem Vormarsch

Wenn man über Softwareentwicklungsprojekte spricht, kommt man in der heutigen Zeit auf keinen Fall um das Thema Agilität herum. Seit einigen Jahren haben sich agile Herangehensweisen bei vielen Unternehmen etabliert – und das zu Recht. Nicht nur lassen sich Performance, Durchsatz und Qualität steigern, auch das Verständnis für moderne Entwicklungsmethoden lässt sich dadurch enorm verbessern.

von Benjamin Lanzendörfer

Nicht nur in der IT-Branche sind Vorgehensmodelle wie Scrum erfolgreich, auch im Marketing oder in der Automobilindustrie kommt Scrum nachweislich mit äußerst positiven Zielen zum Einsatz. Vor allem der Scrum-Rahmen lässt sich aber nicht nur in technischen Projekten wunderbar einsetzen. Beispielsweise gibt es mittlerweile sogar in der Gastronomie erste Ansätze und Vorgehen, die etwa beim Kochen von Gerichten eine tragende Rolle spielen. Tatsächlich gibt es in Hildesheim eine Küche, in der ein Backlog mit den jeweiligen User Stories aufgeführt und die jeweiligen Tasks in definierten Sprints abgearbeitet werden. Warum? Weil es sich nachweislich bewährt und sich die Zusammenarbeit und die Qualität enorm verbessert haben.

Der klassische Scrum-Prozess

Der klassische Scrum-Prozess ist in Abbildung 1 dargestellt. Interessant sind vor allem die beiden Pla-

nungsschritte, die als Sprint Planning 1 und als Sprint Planning 2 bekannt sind. Beide dienen dazu, mit dem Product Owner die Prioritäten der einzelnen Product-Backlog-Items zu besprechen. Daraus werden dann im zweiten Schritt Tasks erstellt, die in mit festen Laufzeiten definierten Sprints erarbeitet werden.

Das Priorisieren der User Storys (der Anforderungen) nimmt ausschließlich der Product Owner vor, denn er allein kennt die Produktvision und hat den direkten Kontakt zu den jeweiligen Stakeholdern – also denjenigen, die das Projekt finanzieren oder beauftragen. Nach dem Priorisieren entscheidet das Scrum-Team im Sprint Planning 2 dann jedoch selbst, wie die einzelnen Tasks entwickelt und im Sprint-Backlog konsolidiert werden. Während des Sprints, der üblicherweise oft im Abstand von zwei Wochen durchgeführt wird, gibt es täglich ein Daily Scrum. In diesem Meeting setzt sich das Scrum-Team für 15 Minuten zusammen. Jeder Teilnehmer beantwortet die folgenden drei Fragen:

1. Was habe ich seit dem letzten Daily Scrum getan?
2. Was plane ich, bis zum nächsten Daily Scrum zu tun?
3. Was hat mich bei der Arbeit behindert?

Vor allem bei einem täglichen Meeting lässt sich schnell erkennen, wie sinnvoll es sein kann, sich regelmäßig miteinander auszutauschen. Jedes Teammitglied ist dann auf dem neuesten Stand, kann Hindernisse mitteilen und beschreiben, was es geschafft hat. Eine unglaublich wichtige Rolle hat hierbei der Scrum Master. Er ist dafür verantwortlich, dass der gesamte Prozessablauf zielgerecht verläuft und das Team ungestört seinen Aufgaben nachgehen kann. Viele nennen ihn auch die „Mutter“ in Scrum-Projekten.

Die offene Feedbackkultur

Das Wunderbare am agilen Ansatz des Scrum-Rahmens ist die feste Situation, in der sämtliche Prozessbeteiligte zusammenkommen und sich in Form einer Retrospektive über positive sowie negative Erkenntnisse, die während des Sprints auftraten, austauschen. Das hat den großen Vorteil, dass die Beteiligten Fehler direkt ansprechen und gemeinsam Lösungen diskutieren können.

Diese Retrospektive sorgt dafür, dass ab einer gewissen Sprint-Anzahl die Fehler drastisch abnehmen. In der Praxis zeigt sich oft, dass mit dem gelebten Scrum-Vorgehen eine positive Einstellung zum gemeinsamen Ziel entsteht. Jeder Beteiligte fühlt sich anerkannt und als Teil einer wichtigen Organisation, in der es darum geht, durch wiederkehrende Iterationen die Qualität und den Durchsatz zu optimieren. Auch gibt es bestimmte Abwandlungen und Anpassungen des Scrum-Prozesses, da viele Unternehmen sich den Rahmen so definieren, dass er für die internen Prozesse und Richtlinien optimal funktioniert. Außerdem setzen die Beteiligten zusätzlich Kanban-Boards ein, um noch mehr Transparenz und Nachhaltigkeit einzubringen.

Einführung von agilen Methoden in Unternehmen ohne Hindernisse?

In der Theorie verhält sich die Einführung von agilen Methoden oder eines klassischen Scrum-Prozesses für die Softwareentwicklung eher unkompliziert, kann jedoch in der Praxis bei vielen Unternehmen oft zu enormen Hindernissen und Schwierigkeiten führen.

Im ersten Schritt sollten die Beteiligten nicht versuchen, den Scrum-Prozess auf die internen Richtlinien und das genutzte Framework anzupassen. Ziel sollte zunächst sein, immer erst ein gemeinsames Verständnis für den Nutzen und den daraus entstehenden Mehrwert zu schaffen. Danach können sich die Beteiligten auf die Vorgehensweise einigen. In der agilen Welt nennt man das Commitment. Das heißt, die Prozessbeteiligten bekennen beziehungsweise verpflichten sich dazu, das gemeinsame Ziel mithilfe der gemeinsam erarbeiteten Vorgehensweise zu erreichen. Bereits in dieser Phase kommt es immer wieder zu Missverständnissen und fehlendem Verständnis. Schuld ist nicht eine bestimmte Rolle oder eine bestimmte Person, sondern die Verfahrensweise der Einführung an sich. Oftmals gibt es erhebliche Kommunikationschwierigkeiten, sowohl zwischen den Fachabteilungen und der IT als auch zwischen den Entwicklern und den Administratoren. Als Instrument für die erfolgreiche Einführung einer agilen Methodik kann es hilfreich sein, sie anhand eines Beispielworkshops zu präsentieren – am besten jeweils in einem fremden Thema. Für Entwicklerteams werden hierfür oft Scrum-Workshops in Form von Cocktails mixen oder – wie oben beschrieben – in einer Küche angeboten. Ziel ist es, erst einmal nur ein Verständnis für die Herangehensweise zu bekommen, unabhängig von Themen, Entwicklungsinhalten sowie Rahmenbedingungen. Sehr häufig lassen sich so die Beteiligten für Agilität besser sensibilisieren, als Scrum direkt und „übers Knie gebrochen“ im eigenen Entwicklungs- und Deployment-Prozess einzuführen.

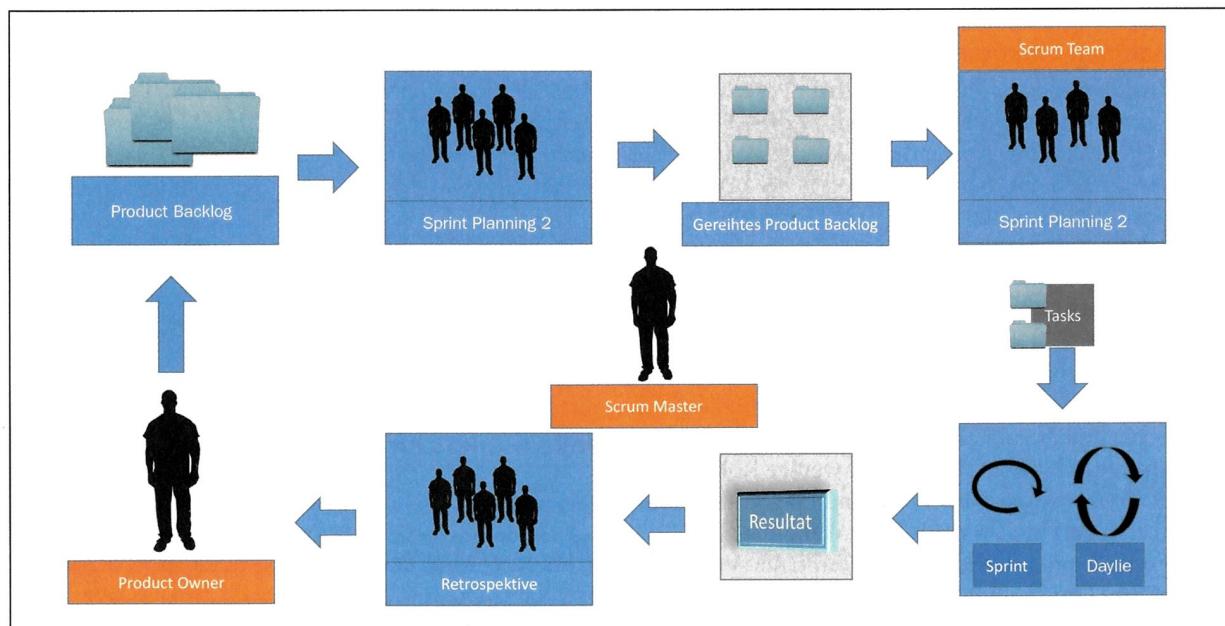


Abb. 1:
Der klassische Scrum-Prozess

Selbstverständlich ist die Einführung von Scrum oft nicht einfach, denn sie ist mit vielen Veränderungen verbunden. Neue Wege zu gehen, alte Gewohnheiten abzulegen und ständiges Optimieren fällt vielen Beteiligten schwer. Neben dem Optimieren von Qualität und Performance sollten deshalb immer auch die Menschen, die in den Prozess involviert sind, im Vordergrund stehen. Eine gelebte agile Methodik ist auch eine gute agile Methodik. Deshalb steht die klare Empfehlung im Vordergrund, die Betroffenen zu Beteiligten zu machen und durch Kommunikation, Transparenz und Sensibilisieren gezielt die Basis dafür schaffen, dass das neue Prinzip einen großen Vorteil für alle haben kann.

Vorteile durch transparentes Vorgehen und definierte Methoden

Als größter Mehrwert überhaupt steht das Fördern der Eigeninitiative im Vordergrund. Im Scrum-Prozess müssen und können die Entwickler innerhalb des Scrum-Teams selbstständig die Anforderungen schätzen und in Tasks herunterbrechen, was dazu führt, dass die Entwickler sich viel stärker mit ihrer getätigten Arbeit identifizieren. Auch kann es sie beflügeln, nach jedem Sprint aus ihren Fehlern zu lernen und so ihre Tätigkeit noch weiter zu verbessern. Die Qualität der entwickelten Features verbessert sich hierdurch automatisch, da die Sensibilisierung für das gemeinsame Ziel viel stärker ist als bei klassischen Vorgehen. Weiterhin zeigt sich in der Praxis, dass Entwickler und Scrum-Teams in ihren Tätigkeiten einen viel höheren Spaßfaktor aufweisen als klassisch organisierte Teams. Dafür sorgt vor allem das eigenverantwortliche Arbeiten, das eine gewisse Flexibilität mit sich bringen kann. Die Entfaltungsmöglichkeiten steigen und die Mitarbeiter sind viel zufriedener mit ihrem Umfeld. Auch die schnellen Ergebnisse und die täglichen Meetings mit den Kollegen können zusätzlich motivieren.

Scrum und Agilität können in jedem Unternehmen funktionieren

Viele Diskussionen widmen sich dem Potenzial von Scrum, zum Beispiel seinem Potenzial für die Allgemeinheit. Viele praktische Anwender von agilen Methoden schwören darauf, dass Scrum – egal ob in der originellen Form oder individualisiert – in jedem Unternehmen seine Berechtigung findet. Egal ob Hersteller von Software, Auto oder Waschmaschine: Sobald es darum geht, etwas zu erschaffen und ein Produkt zu entwickeln, kann Scrum eine interessante und wunderbare Möglichkeit sein, die Performance und die Qualität innerhalb des Herstellungszyklus zu verbessern. Entscheidend hierfür sind die Metaebenen, die Scrum als unkomplizierte und iterative Methode mit sich bringt. Es gibt nur wenige Rollen: den Product Owner, das Scrum-Team und den Scrum Master, und erweitert hierfür noch die Stakeholder.

Neben dem pragmatischen und iterativen Vorgehen sowie einer Transparenz, die zu jedem Zeitpunkt innerhalb des Projekts besteht, bietet Scrum vor allem die

Möglichkeit, sich konzentriert auf qualitative Arbeit zu fokussieren. Selbst komplexe Projekte können durch die Einhaltung eines agilen Rahmens schneller zum Erfolg und der gewünschten Qualität führen als klassische Methoden. Durch die Befähigung eines jeden Beteiligten innerhalb des Scrum-Prozesses und der daraus gewonnenen Flexibilität für jeden Einzelnen entsteht eine dynamische Vorgehensweise, die auch bei kurzfristigen Änderungen oder Anpassungen stets das Ziel im Blick hat.

Warum Machen oft besser ist als Planen

Eine gewisse Planung sollte selbstverständlich sein. Sie wird bei Scrum und agilen Methoden ebenfalls durchgeführt, allerdings in einer viel schmaleren Form als in klassischen Vorgehensmodellen. Das klassische Wasserfallmodell, das phasenorientiert aufgebaut ist, definiert sich so, dass sämtliche Spezifikationen zu Beginn eines Projekts detailliert und vollumfänglich festgelegt werden – getreu dem Motto: „Erst planen und dann arbeiten“. Diese Vorgehensweise ist schlichtweg unrealistisch. Die Tatsache, dass sich Anforderungen während eines IT-Projekts ändern oder auch wachsen, ist ebenso wahrscheinlich wie der Fakt, dass am nächsten Morgen wieder die Sonne aufgeht.

Die Gründe hierfür: Große IT-Projekte sind einfach zu undurchschaubar, um von Anfang an sämtliche Eventualitäten sowie äußere Einflüsse zu berücksichtigen. Der Projektplan sowie das Konzept, das man für ein komplexes Projekt erstellt, ändert sich im Laufe eines Projekts immer wieder – das ist eine natürliche Gegebenheit. Dadurch, dass innerhalb des Scrum-Rahmens der Fokus auf dem zügigen Voranschreiten liegt, kann schnell mit dem „Doing“ begonnen werden. Der gesamte Scrum-Prozess mit den Rollen sowie Schritten ist so einfach, dass er nachweislich auf einen Bierdeckel passt [1]. Grundlage für das Verständnis der Scrum-Philosophie ist das Agile Manifest, das unter [2] nachzulesen ist. Im Zug der Digitalisierung von Prozessen und Vorgehen ist es wichtig, jederzeit auf Veränderungen zu reagieren und Kollaboration und Kommunikation dafür intensiv zu nutzen. Agile Methoden können mit einfachen Mitteln dabei helfen, die Technologien zu erschaffen, die den Menschen im täglichen Leben unterstützen und eine Arbeitserleichterung herbeiführen.



Benjamin Lanzendorfer ist Senior Consultant im Bereich Business Productivity bei der adesso AG. Schwerpunkte seiner Arbeit sind agile Themen und Methoden.

✉ Benjamin.Lanzendoerfer@adesso.de

Links & Literatur

[1] Roock, Stefan: „Scrum – auf dem Bierdeckel erklärt“, Open Book, dpunkt.Verlag, 2016, 30 S.: <https://www.dpunkt.de/buecher/12551/-scrum-auf-dem-bierdeckel-erkl%C3%A4rt.html>

[2] Agiles Manifest für agile Softwareentwicklung: <http://agilemanifesto.org/>

Agilität in der Organisationsstruktur

Agil genug für die digitale Zukunft?

Welche Rolle spielt Agilität in Ihrer Organisationsstruktur? Tim Neugebauer gibt in seinem Gastbeitrag eine Checkliste für die digitale Transformation. Sieben verschiedene Stichworte rund um Innovationen und neue Werte.

von Tim Neugebauer

Die digitale Transformation bringt Unternehmen eine Vielfalt an neuen Möglichkeiten. Doch um diese überhaupt nutzen zu können und daraus Wettbewerbsvorteile zu generieren, braucht es eine neue und agile Unternehmensorganisation – inklusive neuer Denkmuster, Strategien und Handlungsweisen. Doch wie agil und damit gut gerüstet ist mein Unternehmen eigentlich für die digitale Zukunft? Welche Fragen sich Unternehmen diesbezüglich jetzt im Einzelnen stellen sollten, zeigt die folgende Checkliste.

Welche Innovationen erfordert die digitale Transformation?

Es sind nicht nur technologisch induzierte Innovationen, die im Rahmen der digitalen Transformation auf Unternehmen zukommen. Vielmehr gilt es, die komplette Organisation neu aufzustellen, um in Zeiten der Digitalisierung überhaupt Schritt halten zu können. Das erfordert veränderte Prozesse für eine höhere Entscheidungsgeschwindigkeit, neue Strukturen im Sinne flexiblerer Formen der Zusammenarbeit und autonomer Entscheidungsfindung sowie generell digitalzentrierte Strategien. In der Summe gilt es, die Organisationsstruktur nach agilen Gesichtspunkten zu erneuern.

Welchen Stellenwert haben Innovationen in meinem Unternehmen, und welche Reaktionen könnten kommen?
Um zu analysieren, welchen Stellenwert Innovationen im eigenen Unternehmen haben, ist es sinnvoll zu ergründen: Welche Innovationen haben wir in den letzten Jahren bereits herbeigeführt? Um welche Art von Innovationen handelte es sich dabei? Und wie haben unsere Mitarbeiter auf diese Innovationen reagiert? Waren sie offen gegenüber Veränderungen oder hatten sie eher eine ablehnende Haltung? Die Antworten auf diese Fragen geben eine Orientierung, wie agil und flexibel ein Unter-

nehmen in der Vergangenheit im Hinblick auf Veränderungsprozesse war. Und vermitteln daher auch eine Idee davon, welche Reaktionen in Zukunft kommen könnten.

Was verhindert eventuell Innovationen in meinem Unternehmen?

Das größte Hindernis für Innovationen im Unternehmen ist das Tagesgeschäft. Eine zündende Idee entsteht durchaus mal in der Kaffeepause, jedoch erfordert deren Umsetzung – also das Entwickeln von echten Innovationen – zeitliche, personelle, finanzielle und materielle Ressourcen. Weitere Hindernisse sind zum Beispiel starre Strukturen und Hierarchien sowie die fehlende Bereitschaft der Arbeitnehmer, Neues auszuprobieren. Das kann mit der Angst zusammenhängen, Fehler zu begehen. Daher ist eine gesunde Fehlerkultur im Unternehmen eine wichtige Voraussetzung für alle Arbeitnehmer. Aber auch dann ergeben sich oftmals Widerstände gegenüber Veränderungen, einfach weil neue Konzepte am Anfang tendenziell zu Unsicherheiten führen und somit im deutlichen Gegensatz zur sicheren Routine des Tagesgeschäfts stehen. Komplexe digitale Innovationsprozesse müssen erst erprobt werden und sich dabei bewähren.

Wie muss ich mein Unternehmen strukturieren, um Innovationen möglich zu machen? Und welche Rolle spielt dabei das operative Tagesgeschäft?

Im operativen Tagesgeschäft ist nicht viel Zeit, um Innovationen voranzutreiben. Daher empfiehlt es sich, das Unternehmen im Sinne einer dualen Organisation in zwei Organisationsteile zu splitten: Neben dem operativen Tagesgeschäft – der so genannten Performance Engine – gibt es dann einen weiteren Organisationsteil, der dann tatsächlich innovative Ideen entwickelt und voranbringt – die so genannte Innovation Engine. Die operativ im Tagesgeschäft tätige Performance Engine zeichnet sich durch hohe Verlässlichkeit und einen Effizienzfokus aus, Innovationshandeln findet nur eingeschränkt im Sin-

ne inkrementeller Optimierung des Status quo statt. Die Innovation Engine hingegen ist der Organisationsteil, in dem digitale Projekte mit großer Innovationshöhe und gegebenenfalls auch disruptivem Potenzial erprobt werden. Dieser Teil zeichnet sich durch verstärkte Systemoffenheit und hohe Fehlertoleranz aus und hat das Auffinden neuer digitaler Strategien und Geschäftsmodelle im Mittelpunkt. Er ist personell schlank aufgestellt und arbeitet nach agilen Grundsätzen. Damit ein Unternehmen langfristig zu einem digitalen Unternehmen wird, müssen beide Organisationsteile nebeneinander koexistieren und vom anderen lernen: Die Strukturen und Finanzmittel der Performance Engine sind notwendig, um die Aktivitäten in der Innovation Engine überhaupt zu ermöglichen. Umgekehrt ist die Innovation Engine dafür da, durch Innovationen neue Alleinstellungsmerkmale zu generieren und so das operative Tagesgeschäft nach und nach für die digitale Zukunft umzuwandeln [1].

Was kann ich dafür tun, dass mein Unternehmen offen gegenüber Innovationen ist?

Die Digitalisierung erfordert ein intensives Innovations- und Change Management. Hier gilt es, Mitarbeiter frühzeitig über die anstehenden Veränderungsprozesse zu informieren und ihnen so die Angst vor dem Neuen zu nehmen. Neue Arbeitszeitmodelle, mehr organisatorische Freiheiten und strukturelle Flexibilität ermöglichen es den Mitarbeitern, sich mit digitalen Innovationen auch beschäftigen zu können. Das sollte, wie bereits erwähnt, so früh wie möglich erfolgen, damit Mitarbeiter – und generell die ganze Unternehmenskultur – mit den Innovationen mitwachsen können. Mitarbeiter müssen schließlich über digitales Wissen verfügen, um die Herausforderungen der Digitalisierung bewerkstelligen zu können. Hilfreich sind kompetente Ansprechpartner – so genannte Digital Champions – im Haus, die bereits Experten für die digitale Transformation sind, sowie der Austausch mit Branchennetzwerken und -organisationen außerhalb des eigenen Unternehmens und mit der digitalen Start-up-Szene.

Wer in meinem Unternehmen ist überhaupt für Veränderungsprozesse verantwortlich?

Digitale Innovations- und Veränderungsprozesse benötigen Antreiber, die diese leiten und begleiten. Dazu gehören die Entscheider aus dem Topmanagement, die das große Vorhaben initiieren und vorleben müssen. Hilfreich kann es ebenso sein, neue Rollen zu schaffen. Ein Beispiel ist der Chief Digital Officer (CDO). Er ist der oberste Digitalisierungsbeauftragte des Unternehmens, der den digitalen Wandel im Unternehmen leitet, neue Geschäftsmodelle entwickelt, innovative Technologien einführt und das vernetzte Arbeiten im Unternehmen fördert. Dabei agiert er strategisch, arbeitet agil und teamorientiert. Weiterhin lässt sich externes Expertenwissen in einem so genannten Digital Advisory Board bündeln. Dieses Board berät die Geschäftsleitung bei der agilen Transformation des Unternehmens und der

Ausbildung digitaler Kapazitäten. Es ermöglicht zudem einen ungefilterten Außenblick auf die eigene Organisation und hilft damit, mögliche Barrieren und Widerstände im Change-Prozess frühzeitig zu erkennen sowie Marktpotenziale besser zu erschließen. Außerdem bietet es sich an, ganze Innovationsteams aufzustellen: Dabei handelt es sich um Teams, die sich jeweils mit einem digitalen Vorhaben des Unternehmens intensiv auseinandersetzen, neue Ideen entwickeln und deren Umsetzung begleiten. Diese Personen werden nicht vom laufenden Tagesgeschäft abgelenkt, sondern fokussieren sich explizit auf neue digitale Geschäftsoptionen.

Welche Werte sind im Rahmen des digitalen Wandels und in Bezug auf die Unternehmenskultur von großer Bedeutung?

Ein Unternehmen, das sich für die digitale Zukunft wappnet, sollte besonderen Wert auf Teamarbeit, Agilität und Nutzerzentrierung legen. Das bedeutet: Die starren Strukturen im Sinne von getrennten Abteilungen lösen sich auf und werden durch die Zusammenarbeit in funktionsübergreifenden Teams ersetzt. Diese Teams zeichnen sich durch Interdisziplinarität aus. Das bedeutet, dass jeder sein spezielles Wissen in das einzelne Projekt einbringt und eine gewisse Eigenverantwortung im Rahmen der Zusammenarbeit trägt, um das gemeinsam gesetzte Ziel zu erreichen. Eine projektbezogene Zusammenarbeit dieser Art ermöglicht, dass man schneller, flexibler und gezielter auf Kundenanforderungen eingehen kann: Welche Anforderungen hat unser Kunde? Haben sich diese Anforderungen zum Zeitpunkt X geändert, und muss ich deshalb meine Lösung anpassen? Probleme werden durch eine hohe Prozesstransparenz frühzeitig erkannt, Änderungen am Produkt lassen sich rechtzeitig vornehmen und Lerneffekte kontinuierlich erzeugen.



Tim Neugebauer (Jg. 1980) ist Mitgründer und Geschäftsführer der DMK E-BUSINESS GmbH. Der Diplom-Kaufmann verantwortet in der Geschäftsführung die Bereiche Strategie, Beratung und Vertrieb. Er lehrt zudem an der Beuth Hochschule für Technik Berlin in den Bereichen Dienstleistungsmarketing sowie Innovationsmanagement und ist Sprecher des Forums Digital Business im Verband der Softwareindustrie Berlin-Brandenburg (SIBB e.V.).

Links & Literatur

- [1] Govindarajan, Vijay; Trimble, Chris: „The Other Side of Innovation – Solving the Execution Challenge“, Harvard Business Review Press, 2010

Buchauszug

Diese Checkliste ist eine Zusammenfassung aus dem Buch „Digital Business Leadership. Digitale Transformation – Geschäftsmodell-Innovation – agile Organisation – Change-Management“. Dort erhalten Sie noch mehr Informationen und Anregungen, die Ihnen zeigen, wie Sie Ihr Unternehmen für die digitale Zukunft rüsten können.

Wie sich UX-Design erfolgreich in den Scrum-Prozess integrieren lässt

Gemischtes Doppel

In der Softwareentwicklung setzt sich Scrum als agile Projektmanagementmethode zunehmend durch. Inzwischen gibt es zahlreiche Workshops, umfassende Literatur und eine große Anzahl erfahrener Scrum-Experten. Im Scrum-Kontext wurde es bisher jedoch vernachlässigt, andere Elemente und fachliche Disziplinen, die essenziell für ein starkes Produkt sind, zu integrieren – wie beispielsweise das User-Experience-Design.

von Heidi Oltersdorff

Eine gute User Experience (UX) führt dazu, dass Anwender ein Produkt bzw. eine Software gerne nutzen – und ist damit ein wesentlicher Erfolgsfaktor. Dennoch treten in der Praxis User-Experience-Design und Scrum-Prozess (Abb. 1) bisher meist nur nebeneinander an und haben wenige Berührungspunkte: Das User-Experience-Design – vom User Research (Abb. 2) über die Definition von Personas bis hin zur Erarbeitung von Skizzen und Wireframes – wird oft im Voraus erstellt und für die Umsetzung dem Entwicklungsteam übergeben. Hinzu kommt, dass meist externe, spezialisierte Agenturen – die oft nach der klassischen Wasserfallmethode arbeiten – das User-Experience-Design gestalten. Das liegt zumeist daran, dass bei der Gründung insbesondere der seit Längerem bestehenden IT-Unternehmen das Thema UX noch nicht so sehr im Vordergrund stand wie heute.

Mit der zunehmenden Bedeutung und Verbreitung des Internets bis auf die Bildschirme unserer Smartphones, haben sich die Ansprüche der User an eine Software jedoch verändert. Sie legen deutlich mehr Wert auf das Design und die Nutzerführung einer Software oder einer Internetanwendung. Um den stetig steigenden Ansprüchen der User gerecht zu werden und nicht zuletzt, um Marktanteile zu halten oder gar auszubauen, muss das UX-Design den User nicht nur bei der Anwendung unterstützen, sondern ihm auch ein gutes Gefühl bei der Nutzung der Software vermitteln. Je weniger Softwareunternehmen also auf ein optimales User-Experience-Design verzichten können, umso wichtiger ist es, diese Disziplin mit dem Scrum-Prozess zu verzähnen, der von ihnen meist für die Entwicklung eingesetzt wird. Denn nur wenn das User-Experience-Design und die Entwicklung im Rahmen eines agilen Prozesses gemeinsam antreten, steht am Ende ein Sieg: ein erfolgreiches Produkt, das nicht nur funktioniert, sondern das die Nutzer auch gerne anwenden.

Die drei wichtigsten Aspekte für eine erfolgreiche Integration von UX-Design in den Scrum-Prozess lassen sich wie folgt aufschlüsseln:

Erfolgsfaktor 1: Räumliche Nähe – für mehr Austausch und gegenseitiges Verständnis

Eine gelungene agile Zusammenarbeit zwischen UX-Designern und Softwareentwicklern wird durch viele Faktoren bestimmt, der Hauptfaktor aber ist das Team selbst. Häufig ist jedoch das Verhältnis zwischen Designern und Entwicklern eher angespannt. Das liegt nicht selten daran, dass beiden Gruppen nur wenig über die Arbeitsprozesse der jeweils anderen Gruppe wissen. So ist es für Designer oft nicht leicht nachzuvollziehen, wie die Entwickler mit den von ihnen zur Verfügung gestellten Designs weiterarbeiten. Die Entwickler wiederum können sich häufig kein Bild von den Prozessen machen, die Designer durchlaufen, um ein valides Konzept zu gestalten. Für beide Seiten ist es mitunter sehr intransparent, was die jeweils andere Gruppe in ihrer täglichen Arbeit leistet.

Aufklärungsarbeit seitens der Projektleitung und der Teammitglieder selbst kann hier zunächst ein grundlegendes Verständnis schaffen. Dabei sollte erklärt werden, was die Aufgaben der einzelnen Rollen sind und inwiefern sich durch die Integration der Designer auch Prozesse in der Softwareentwicklung anpassen müssen. Es ist wichtig, dass alle Teammitglieder verstehen, dass nur durch ein gutes Zusammenspiel beider Prozesse – User-Experience-Design und Softwareentwicklung – ein gelungenes Produkt entstehen kann.

Um das Verständnis für die Arbeit der jeweils anderen zu verbessern, ist räumliche Nähe ein wesentlicher Aspekt: Wenn Designer und Entwickler zusammen im gleichen Raum arbeiten, bekommen sie mit, an was der jeweils andere arbeitet. Sie können sich Fragen stellen und/oder ihre Arbeitsschritte wie z. B. Designs oder Softwarearchitekturbilder sichtbar im Raum anbringen. Durch das offensichtliche Visualisieren der aktuellen Arbeiten können einfacher und schneller Gespräche und Diskussionen entstehen. Diese Kommunikation – die weitaus häufiger stattfindet, wenn ein Team in einem Raum sitzt – fördert den kontinuierlichen Austausch, auch im Hinblick auf die technische Realisierung von Designideen. Kurze Kommunikationswege führen dazu, dass Fragen häufiger gestellt

und schneller geklärt werden, und wirken sich damit positiv auf das Produktergebnis aus.

Wenn es zeitlich möglich ist und das entsprechende Fachwissen vorliegt, können die Designer die Entwickler auch bei der Programmierung unterstützen; etwa im Bereich der Frontend-Entwicklung. Der Vorteil: Selbst entworfene Designs lassen sich auch gleich technisch umsetzen und somit auf ihre Machbarkeit prüfen. Umgekehrt können sich auch die Entwickler im Rahmen des Scrum-Prozesses an der Erstellung des User-Experience-Designs beteiligen, etwa indem sie Mock-ups oder Prototypen erstellen.

Erfolgsfaktor 2: UX-Design schon in Sprint 0 berücksichtigen – für relevante Personas und gute User Stories

Im Scrum-Prozess wird die Zeit vor dem ersten Sprint für vorbereitende Maßnahmen genutzt, wie z.B. die Aufstellung der technischen Architektur. Dieser Zeitraum wird auch als „Sprint 0“ bezeichnet. In diese Phase lassen sich die ersten konzeptionellen und kreativen Prozessschritte des UX-Designs gut integrieren. Ziel von Sprint 0 ist in diesem Fall, gemeinsam ein grundlegendes Verständnis für den User und eine einheitliche Produktvision zu erarbeiten.

Hierfür empfiehlt sich ein Workshop, in dem alle Projektbeteiligten – Entwickler, User-Experience-Designer, Product Owner und idealerweise auch die Stakeholder – ein Big Picture der UX-Idee entwickeln. Ebenso sollten bereits in Sprint 0 die ersten Schritte des UX-Designs umgesetzt werden: User Research, Persona-Erstellung und das Erarbeiten von Anforderungen in Form von User Stories. Auf dieser Basis können die Beteiligten gemeinsam ein für alle verständliches Product Backlog erarbeiten. Bereits im Sprint 0 sollten Daily-Scrum-Meetings stattfinden, in denen sich die Teammitglieder gegenseitig über den Fortschritt und aktuellen Stand der Vorbereitungsphase informieren.

Die User Stories, die im Sprint 0 zusammen erarbeitet werden, sortiert der Product Owner am Ende des Sprints nach ihrer Priorität. Am höchsten werden die User Stories priorisiert, die das so genannte Minimal Viable Product (MVP) abbilden – also das Produkt oder Feature, welches mit dem geringsten Aufwand den größtmöglichen Nutzen für den User bringt. Diese User Stories sollen dann in den ersten Sprints umgesetzt werden.

Um abschätzen zu können, wie viele Sprints benötigt werden, um das MVP umzusetzen, sollten die Entwickler in Sprint 0 eine Aufwandsschätzung der priorisierten User Stories vornehmen. Aufgabe der User-Experience-Designer in dieser Phase ist es, den Wert und Nutzen einer User Story zu beschreiben und die Entwickler an den Aufwand für die Umsetzung des Designs zu erinnern.

Für die Entwickler kann es schwierig sein, eine Aufwandsschätzung abzugeben, wenn noch kein definiertes User-Experience-Design vorhanden ist. Um ein gemeinsames Verständnis – und damit eine genauere Schätzung – der User Stories zu erreichen, empfiehlt es sich, während Sprint 0 einfache (Papier-)Prototypen zu erstellen. Die

Ideen aller Projektbeteiligten lassen sich auf diese Weise schnell visualisieren und so besser in der Gruppe diskutieren. Die tragfähigsten Ideen können dann weiterent-

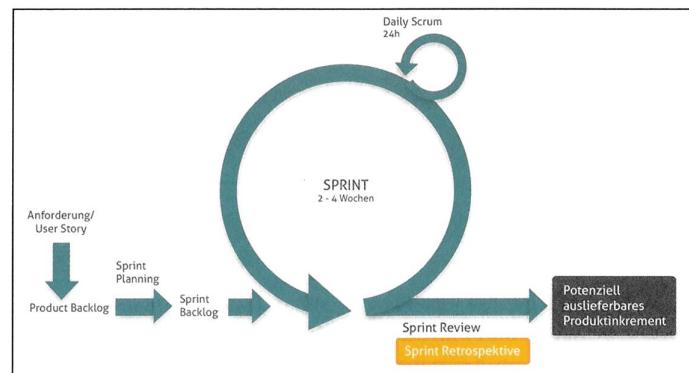


Abb. 1: Scrum ist in der Softwareentwicklung inzwischen weit verbreitet – aber häufig werden andere Disziplinen noch nicht in den Prozess integriert

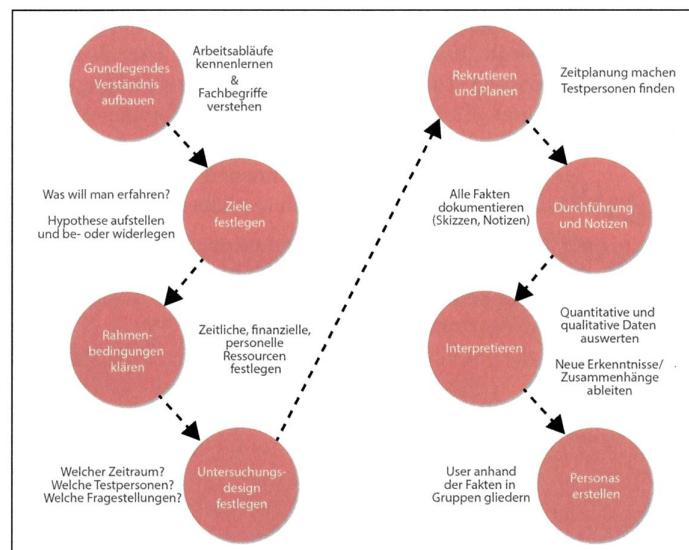
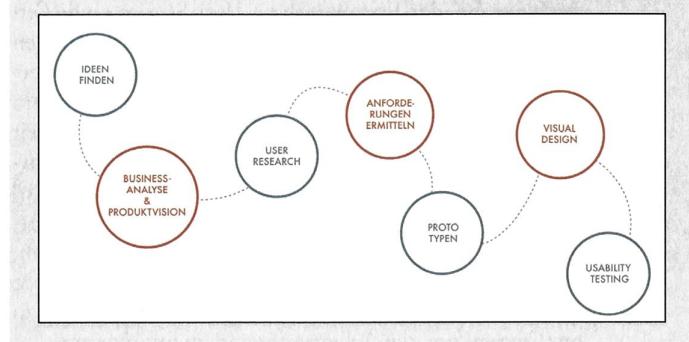


Abb. 2: Eine umfassende User Research bedeutet zwar Aufwand, kann aber wesentlich zum Produkterfolg beitragen

User-Experience-Design

Prozessbeispiel „User-Experience-Design“: Gutes UX-Design reicht von Businessanalyse und User Research über die Definition der Anforderungen bis hin zu Erarbeitung von Prototypen und regelmäßigen Usabilitytests.



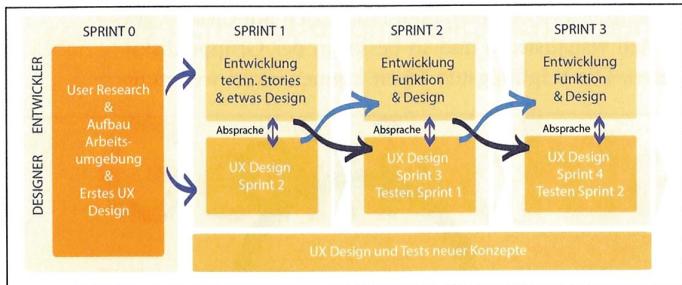


Abb. 4: Beispiel für einen agilen Projektprozess im Water-Scrum-Modell: Nur wenn User-Experience-Design und Entwicklung zusammenspielen, steht am Ende ein erfolgreiches Produkt

wickelt und idealerweise anhand von Prototypen noch in Sprint 0 auf ihre Benutzerfreundlichkeit getestet werden, z.B. von Kollegen. Das Feedback der Tester lässt sich dann in einer schnellen Iteration berücksichtigen.

Der parallelen Entwicklung geschuldet, wird es im Laufe des Projekts nur noch den User-Experience-Designern möglich sein, den kompletten Prozess immer wieder durchzuführen, d.h. kontinuierliche User Research, Anpassung – und falls erforderlich Neuerstellung – von Personas, Erstellung und Test von Prototypen etc. Die User-Experience-Designer sollten im Laufe des Prozesses darauf achten, ihre gewonnenen Erkenntnisse und Resultate regelmäßig mit dem restlichen Team zu teilen.

Erfolgsfaktor 3: UX-Designer an allen Scrum-Meetings beteiligen – für das „Big Picture“

Voraussetzung für eine ganzheitliche Integration der User-Experience-Designer in den Scrum-Prozess ist deren Teilnahme an allen Scrum-typischen Meetings. Dazu zählen Daily Scrum, Plannings, Product Backlog Refinement, Retrospektiven und Reviews.

Im Rahmen dieses „institutionalisierten“ Austauschs zwischen Entwicklern und User-Experience-Designern – idealerweise zusätzlich unterstützt durch räumliche Nähe – können beide Gruppen schnell herausfinden, ob sich das erstellte User-Experience-Design mit angemessenem Aufwand umsetzen lässt. Weiterhin haben die Entwickler so auch die Möglichkeit, ihre Ideen und Ansichten in den UX-Design-Prozess einfließen zu lassen.

Ebenfalls empfiehlt sich eine enge Zusammenarbeit zwischen UX-Designern und Product Owners, speziell bei der Ausgestaltung von User Stories. Sie können ihre Erfahrung im Umgang mit den Nutzern und deren Bedürfnisse bei der Erstellung von User Stories einfließen lassen und User-Experience-Designs zu den geplanten User Stories anfertigen. Dabei sollte darauf geachtet werden, sehr detailliertes User-Experience-Design erst dann zu erstellen, wenn es benötigt wird – und nicht schon weit im Voraus. Das Gleiche gilt auch für die finale Ausformulierung der User Stories, da es sehr wahrscheinlich ist, dass sich initiale User Stories aufgrund neuer Erkenntnisse im Laufe des Projekts ändern. Somit ist es ratsam, die User Stories auch hinsichtlich ihrer User Experience erst ein bis zwei Sprints vor der geplanten Umsetzung zu finalisieren, um den Arbeitsaufwand zu minimieren.

Als fester Bestandteil eines Scrum-Teams stehen die User-Experience-Designer jederzeit für Rückfragen zur Verfügung. Das trägt dazu bei, Mehrfacharbeit zu verhindern, etwa wenn ein Detail zum Verhalten des User Interface vom Entwickler anders verstanden und ohne Rückfrage implementiert wurde. Indem sie die Akzeptanzkriterien einer User Story prüfen, unterstützen die UX-Designer auch den Product Owner, der die User Stories maßgeblich mitprägt.

Usertests sind ein weiteres Aufgabenfeld, dessen sich UX-Designer während der Entwicklung annehmen können. Hierfür empfiehlt es sich, mit ausgewählten Anwendern in regelmäßigen Abständen – etwa alle zwei Sprints – Usabilitytests durchzuführen, idealerweise zunächst mit Prototypen. So hat die Zielgruppe die Möglichkeit, neu implementierte Produktinkremente zu evaluieren. Eine frühe Rücksprache mit den Anwendern hat den Vorteil, dass etwaige Änderungswünsche der User umgehend berücksichtigt werden können. Insbesondere wenn die Anpassungen bereits an Prototypen erfolgen und nicht erst an der bereits entwickelten Software, bedeutet das enorme Zeit- und Kostenersparnis. Anpassungen lassen sich direkt in einer darauffolgenden Iteration vornehmen, oder die Änderungswünsche werden in neuen User Stories verankert. Um den transparenten Austausch mit dem Scrum-Team aufrecht zu erhalten, sollten die UX-Designer die Ergebnisse der Usertests im Daily Scrum präsentieren.

Fazit

User-Experience-Designer können in einem Scrum-Team umfassende Aufgaben übernehmen und wesentlich zum Erfolg des Produkts beitragen. Voraussetzung dafür ist, dass die UX-Designer als Doppelpartner der Entwickler im Scrum-Prozess mitspielen (Abb. 4). Zum einen ist es ihre Aufgabe, nach vorne zu schauen und mittels User Research, Usertests und Prototyperstellung eine auf den Anwender ausgerichtete Entwicklung des Produkts voranzutreiben. Zum anderen sollten sie immer für Nachfragen der Entwickler bezüglich aktuell bearbeiteter User Stories verfügbar sein, die bereits implementierten User Stories validieren und gegebenenfalls an der Zielgruppe testen.

Dabei sollten sich alle Beteiligten bewusst sein, dass es in der agilen Softwareentwicklung im Grunde keine endgültige Lösung gibt. Mit sich verändernden Bedürfnissen der User muss sich schließlich auch das Produkt verändern – indem es stetig angepasst und weiterentwickelt wird. Daher sind mit der Freigabe eines Softwareprodukts die Entwicklung und das User-Experience-Design nicht abgeschlossen. Im Prinzip beginnt in diesem Moment schon ein neues Match.



Heidi Oltersdorff, B.Sc. Informatik, ist seit mehr als zwei Jahren für die diva-e Digital Value Enterprise GmbH als Consultant im Projektmanagement tätig, mit Schwerpunkt im Bereich E-Commerce-Lösungen. In der Rolle des Product Owners unterstützt sie die agile Umsetzung von Softwareentwicklungen mit konsequenter Fokus auf optimale User Experience.

www.diva-e.com

Betrachtet man den Schaffensprozess von Software, so kann man auch historisch den Wechsel von ingenieurmäßigem Vorgehen und kreativen Tätigkeiten erkennen und zuordnen. In den Anfängen der Softwareentwicklung ging es zunächst darum, lauffähige Programme zu erstellen, die das Fachproblem auf eine technische Art und Weise lösten. Benutzerorientierung oder ein ansprechendes Design waren damals noch kein Thema. Die Art der Programmierung war dagegen bereits durchaus komplex. Sie wurde nicht nur durch gute Architektur, sondern vielmehr durch ein hohes Maß an Kreativität des Entwicklers bewältigt. Die entstandenen Programme waren in ihrer Struktur oft fehleranfällig und schwer zu warten, und der Inbegriff der „Spaghetti“-Programmierung war damit geboren.

Ein Blick auf den heute üblichen Entwicklungsprozess liefert ein exakt spiegelverkehrtes Bild zu damals. Die technische Umsetzung basiert auf etablierten Technologien, mit dem Einsatz von Entwurfsmustern versucht man, den Herstellungsprozess weitestgehend zu standardisieren. Zwar ist auch hierbei immer noch ein Stück kreative Arbeit gefragt, aber der Anteil an exakt planbaren Tätigkeiten überwiegt. Demgegenüber stehen diejenigen Prozesse der Softwareentwicklung, die sich mit der Lösung der Aufgabenstellung aus direkter Endbenutzerperspektive beschäftigen. Hierzu zählen die Gestaltung des Benutzeroberflächen und die grundsätzliche Erarbeitung eines Lösungskonzepts. Mit viel kreativer Energie versucht man heute, Probleme des Anwenders zu antizipieren und ihm eine Lösung in Form von Software bereitzustellen. Die Programme sollen dabei neben einem reibungslosen Funktionieren auch ein positives Benutzererlebnis vermitteln. Hoch kreativ ist auch der Bereich der Ideen- und Neuproduktentwicklung. Es sollen Innovationen kreiert werden, um das Interesse des Benutzers an einem neuen Produkt zu wecken. Die größten thematischen Innovationsfelder liegen hier seit einigen Jahren im Bereich des Mobile Computing und seit einiger Zeit im weiten Umfeld des Internet of Things.

Man weiß zwar, dass viele gute und vor allem auch erfolgreiche Ideen aus dem Zufall heraus entstanden sind, dennoch sind die Akteure i. d. R. nicht zufrieden damit, auf die „zündende“ Eingebung zu warten. Das Ziel ist es vielmehr, auch diesen kreativen Prozess zu steuern und zu fördern. Eine Methode ist Design Thinking, die auch zunehmend im Bereich der IT angewendet wird. In diesem Artikel möchten wir Ihnen den Ansatz und das Vorgehen von Design Thinking vorstellen. Kommen Sie mit auf eine kreative Spielwiese.

Was ist Design Thinking?

Auf der Webseite des Hasso-Plattner-Institute (School of Design Thinking) findet man folgende Definition, was unter dem iterativem Design-Thinking-Prozess verstanden wird: „Der sechsstufige iterative Design-Thinking-Prozess verknüpft die Methodik des Ingenieurwesens mit den experimentellen Aspekten aus der Designlehre, schaut mit sozialwissenschaftlicher Brille auf die Nut-

zerInnen und hat seine Ohren dabei immer offen für Neues. Teammitglieder aus unterschiedlichsten Bereichen ziehen hier am selben Strang. Wertschätzende, innovationsfördernde Kommunikation hilft dabei, über Fachbegriffe und Hierarchiegrenzen hinaus, einen gemeinsamen Sprach- und Denkraum zu entwickeln. Nur so können die komplexen Herausforderungen gemeinsam bewältigt werden. Das Ergebnis: überzeugende Innovationen für unterschiedlichste Lebensbereiche [1]. Einen kompakten Überblick über das Entstehen von Design Thinking liefert der Kasten „Eine kurze Geschichte des Design Thinkings“.

Design Thinking im Überblick

Design Thinking steht für Ideenfindung und innovative Problemlösungen. Der Fokus liegt darauf, neue Potenziale und unbekannte Wege zu entdecken. Das Motto lautet: „Wirklich erfolgreiche und außergewöhnliche Lösungen werden nur dann entwickelt, wenn die bekannten Annahmen und Lösungen in Frage gestellt werden und gedanklich Neuland betreten wird“. Innovationen sollen dabei im Team erarbeitet werden. Dieses Team sollte möglichst heterogen in der Zusammensetzung sein, und das Umfeld soll kreatives Arbeiten fördern. Wirtschaft, Forschung und Politik erkennen das Potenzial, das in interdisziplinären Teams steckt. Im Vergleich zu anderen Innovationsmethoden und -prozessen wird dabei ein besonderer Wert auf die Praxis und die nutzergerechte Gestaltung gelegt. Design Thinking hat bisher vor allem seine Akzeptanz in größeren Unternehmen gefunden, kleinere und mittlere Unternehmen holen jedoch auf. Design Thinking basiert auf drei Bausteinen [1]:

- **People:** Das Team wird multidisziplinär formiert. Somit wird es möglich, über die Fachgrenzen hinaus zu gehen, agiler und schneller zu reagieren, die Vorteile der kollektiven Intelligenz zu nutzen und durch

Eine kurze Geschichte des Design Thinkings

Die Gründer, Entwickler und Vertreter des Design Thinkings sind Terry Winograd, Larry Leifer und David Kelley. Das Konzept wurde stark durch den Milliardär Hasso Plattner unterstützt. Im Jahr 1991 fand die erste Tagung zum Thema „Design Thinking Research Symposia“ statt, seit dem Jahr 2005 werden die Prinzipien des Design Thinkings am Hasso-Plattner-Institute of Design gelehrt. 2007 wurde die School of Design Thinking am Hasso-Plattner-Institute of Design in Potsdam gegründet. Seit einigen Jahren bietet die Hochschule für Medien und Kommunikation in München Design Thinking in einem Masterstudiengang an. Ebenso wird Design Thinking im Rahmen von anderen Studiengängen, z. B. Supply Chain Management, Kommunikation, Medien etc. unterrichtet.

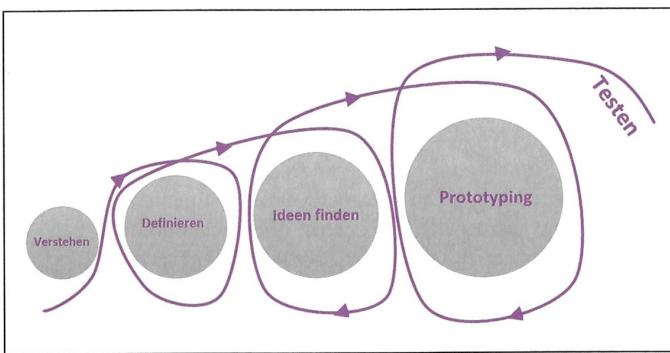


Abb. 1: Der Design-Thinking-Prozess [2]

nachhaltige Arbeitsprozesse auf erstaunliche Resultate zu kommen. Der Trend geht dabei eindeutig zur Wir-Kultur und zu einem gemeinsamen Erschaffen. Die Bereitschaft zum kontinuierlichen Lernen, ganzheitlichen Denken und auch Risikobereitschaft sollten vorhanden sein. Andere Arbeitsaufgaben müssen zurückgefahren werden, d. h., dem kreativen Prozess muss ein definierter Zeitrahmen zur Verfügung stehen. Kreativität gelingt nicht unter Zeitdruck!

- **Place:** Die richtige Umgebung ist wichtig! Am besten eignen sich variabel umstellbare Räume, die spontan an die Bedürfnisse des jeweiligen Projekts anpassbar sind. Tische und Stellwände sollten idealerweise auf Rollen bewegbar sein. Wände und andere Oberflächen können frei zum Visualisieren von Gedanken genutzt werden. Vertikale Flächen sollen zum Bekleben

und Bemalen ausgelegt sein. Post-its, Farbstifte, Klebestreifen und weiteres Material sollte bereitstehen.

- **Process:** Der nachfolgend dargestellte Prozess dient als Leitfaden und ist kein Dogma. Wichtig: Im Vordergrund steht stets der Nutzer mit seinen Bedürfnissen.

Es handelt sich um einen iterativen Prozess (Abb. 1), d. h., nach dem ersten Durchlauf ist man noch nicht am Ziel, sondern es findet eine Wiederholung der Schritte statt.

Die Teilnehmer durchlaufen mehrere aufeinander aufbauende Phasen. Oft kristallisieren sich erst im Laufe eines Prozesses die wesentlichen Fragestellungen und Erkenntnisse heraus; gelegentlich ist auch ein Zurück-springen in eine bereits durchlaufene Phase notwendig. Mit diesem zyklischen Vorgehen wird die Qualität der Ergebnisse gesteigert und ein effizientes und ergebnisorientiertes Arbeiten sichergestellt. Das Problemverständnis, die Definition der Zielgruppe und die Kenntnis der Systemgrenzen sind dabei entscheidend. Am Ende der ersten Phase (Problem verstehen) soll die so genannte Design Challenge feststehen. Darunter wird die Aufgabenstellung verstanden. Man hat sich also mit der Aufgabe/dem Problem intensiv vertraut gemacht, wobei das Verständnis ausgeprägt und spezifisch sein muss. Einschränkungen mit Blick auf mögliche Lösungen sind zu unterlassen. Mögliche Fragen sind:

- Wo liegt der Fokus?
- Welche Regeln sind in der Branche üblich?
- Welche Gestaltungsmöglichkeiten gibt es?
- Was sind die wesentlichen Einflussfaktoren?

Prototyping

Prototypen stellen frühe Muster des zu entwickelnden Produkts dar. Sie dienen dazu, ein entsprechendes Feedback von den Auftraggebern einzuholen. Das kann beispielsweise im Rahmen einer interaktiven Präsentation geschehen. Dabei wird der Prototyp vorgeführt und ggf. den potenziellen Anwendern zum „Anfassen“ und „Ausprobieren“ in die Hand gegeben. Bei einem solchen Meeting wird sehr schnell deutlich, an welchen Stellen Korrekturbedarf erforderlich ist bzw. die Ansprüche bereits treffsicher umgesetzt wurden. Für den Softwareentwickler mag die Situation bezüglich des geringen technischen Verständnisses der Anwender befremdlich sein, und oftmals ziehen vermeintlich kleine Wünsche eine Flut von Änderungen nach sich. Aber als Entwickler kann man sich ebenfalls sicher sein, dass die andere Sichtweise nicht anders ist. Der Informatiker redet aus Sicht des Benutzers ebenfalls nur „Fachchinesisch“ und kann sich offensichtlich wenig unter der praktischen Situation und Fragestellung vorstellen. Da wir als Softwareentwickler als Dienstleister unterwegs sind, dürfte es keine Frage sein, welcher Personengruppe die Anpassung obliegt. In diesem Sinne kann die Bereitstellung von Prototypen ein geeignetes Hilfsmittel sein, um mit der Abstimmung der Anforderungen voranzukommen.

Passgenaue Lösungen können nur dann entwickelt werden, wenn das Team es schafft, sich in die potenziellen Nutzer hineinzuversetzen und somit so viel wie möglich über diese zu erfahren. Eine wichtige Methode dabei sind Personas. Mit deren Hilfe können typische Nutzer mit ihrem Verhalten, Lebensumständen, Problemen, Interessen und Bedürfnissen bestmöglich abgebildet werden.

Schritt zwei (Definieren) ist bereits konkreter. Nach dem Verstehen des Problems muss man zu konkreten Zielen gelangen. Das Innovationsziel entspringt dabei nicht der reinen Fantasie, sondern berücksichtigt neben den Kundenaspekten auch die Einschränkungen aus technischer Perspektive sowie wirtschaftliche Gesichtspunkte (Abb. 2). Ziele können also nur formuliert werden, wenn die technische Umsetzung im Bereich des Möglichen liegt. Beispielsweise ist ein Ziel, ein Elektroautomobil mit einer Reichweite von 1 000 km auf den Markt zu bringen; unakzeptabel ist, wenn man genau weiß, dass die Batterietechnologie noch nicht soweit ist. Auch betriebswirtschaftlich muss sich das Ziel grundsätzlich darstellen lassen.

Bei der Beschäftigung mit dem Thema wird viel Material zusammengetragen, das für die weitere Verarbeitung aufzubereiten ist. Dazu kann zum Beispiel jeder Teilnehmer seine Skizzen, Fotos, Notizen usw. an einer

Wand präsentieren. Diese Informationen werden analysiert, bewertet und in das Gesamtkonzept eingeordnet.

Weiter geht es im nächsten Schritt mit der Ideenfindung. Es geht darum, innovative und vielversprechende Ideen zu entwickeln. Die Ergebnisse werden sortiert, visualisiert und vorgestellt. Anschließend werden sie hinsichtlich Umsetzbarkeit und Rentabilität bewertet. Wichtig: Ideen dürfen nicht zu früh aussortiert werden. Die letzte Phase beschäftigt sich mit dem Erstellen von Prototypen (Kasten: „Prototyping“). Diese ermöglichen es, die Lösungsvorschläge greifbar zu machen und sie dem Kunden besser vorzustellen. Außerdem sind sie ein wesentlicher Erfolgsgarant. Wenig überzeugende Lösungsvorschläge werden auf diese Weise frühzeitig aussortiert.

Prozess vs. Denkweise vs. Disziplin

Design Thinking kann als Prozess, Denkweise oder Disziplin verstanden werden [3]:

- Design Thinking als Prozess ist eine konsequente Schritt-für-Schritt-Vorgehensweise mit definierter Eingabe (Problemstellung, Fragestellung etc.) und definierter Ausgabe (Prototyp, Implementierung etc.).
- Design Thinking als Denkweise lässt sich nicht eindeutig definieren. Es gibt mehrere Bausteine, auf denen Design Thinking basiert: „Set of Mindsets“, „Way of Thinking“, „Thinking as a Designer“, „Design Attitude“ etc.
- Design Thinking als Disziplin ist ein ganzheitlicher Begriff, der Methoden und Denkweisen umfasst und somit als eigenständige Disziplin verstanden wird.

Der perfekte Design Thinker

Welche Qualifikationen sollte der „perfekte“ Design Thinker mitbringen? Sind diese bereits vorhanden oder können sie noch erlernt und entwickelt werden? Unbestritten ist, dass manche Menschen schon geborene Design Thinker sind. Andere müssen in diesem Punkt noch an sich arbeiten. Nachfolgend wollen wir dem Idealtyp des „perfekten“ Design Thinkers näherkommen [4].

Kreativität ist wahrscheinlich die wichtigste Fähigkeit des Design Thinkers. Kreativ heißt, es anders – neu und nicht so wie bereits bekannt – zu machen. Die Kreativität stellt die Basis dar, um zu Ideen im Innovationsprozess zu gelangen. Nicht weniger wichtig ist es, dass der Design Thinker die Situation bzw. die Perspektive von anderen Menschen einnehmen kann. Auf diese Weise findet man Lösungen, die sowohl die offensichtlichen, als auch die verborgenen Bedürfnisse der Kunden ansprechen. Anders ausgedrückt: Um innovative und kundenorientierte Lösungen zu generieren, braucht es Empathie. Natürlich soll der Design Thinker Freude am Experimentieren haben. Wechselseitige Abhängigkeiten zwischen Problem und Lösung sollte er erkennen und falsche Annahmen und Schwierigkeiten identifizieren. Der Design Thinker muss in der Lage sein, das Prob-



Abb. 2: Die Rolle der Kundenorientierung im Design-Thinking-Prozess [2]

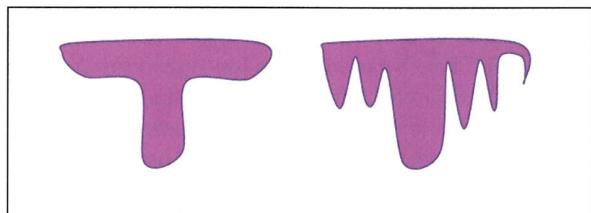


Abb. 3: T-shaped-Konzept vs. drippy-T-Konzept [3]

lem ggf. neu zu definieren. Kreative Arbeit passt nicht zu Pessimismus, daher soll der Design Thinker optimistisch bleiben und stets daran glauben, eine Lösung zu finden. Die stetige Verbesserung der aktuellen Alternative ist sein Antrieb. Design Thinker arbeiten oft in Teams mit anderen Menschen, die jeweils aus unterschiedlichen Disziplinen kommen. Daher müssen sie in der Lage sein, sich in einem interdisziplinären Umfeld zu behaupten. In der Gesamtschau kann man die Kompetenzen des Design Thinkers anhand des T-shaped-Konzepts beschreiben (Abb. 3, links). Dieses Konzept wurde durch die Firma McKinsey & Company bekannt. Die vertikale Achse steht für ein sehr gut ausgeprägtes Wissen über eine spezielle Disziplin. Die horizontale Achse steht für ein breites Wissen in einer weiteren Disziplin. Durch Guldbrandsen und Dijk wurde das T-shaped-Konzept zum so genannten „drippy T“ modifiziert (Abb. 3, rechts). Diese Modifikation entspricht den Erfahrungen aus der Praxis: Es genügt nicht mehr, nur in einer Disziplin über Detailwissen zu verfügen, sondern neben Kenntnissen in der Breite (Überblick) benötigt man in mehreren Bereichen umfassende Einblicke.

Innovation als Motor des Design Thinkings

Innovation ist eine Grundlage und Motor des Design Thinkings. Das Ziel ist es, über kurz oder lang eine Innovation hervorzubringen. Grundsätzlich kommt sie durch die Marktnachfrage („demand-pull“) oder durch das Marktangebot („technology-push“) zustande. Nachfolgend geben wir einen Überblick über die

Design Thinking kann dabei helfen, den Nutzer mit seinem Problem besser zu verstehen und eine Lösung zu entwickeln.

Objekte der Innovation und klassifizieren nach dem Innovationsgrad [3]. Innovation steht für „Neuerung“ oder „Erneuerung“. Bezuglich des Innovationsobjekts unterscheidet man zwischen Produktinnovation, Verfahrensinnovation und Sozialinnovation. Bei der Produktinnovation liegt der Fokus auf der Neugestaltung von Produkten, bei der Verfahrensinnovation auf der Neuaustrichtung der Prozesse, und Sozialinnovationen beschäftigen sich mit Veränderungen im Humanbereich. In der Praxis kommen die Innovationsarten meist in Kombination vor. Hinsichtlich des Innovationsgrads kann zwischen der radikalen und inkrementellen Innovation unterschieden werden. Radikal ist die Innovation, wenn das technologische Wissen sich signifikant vom bestehenden Wissen unterscheidet. Bei der inkrementellen Innovation wird die Funktionalität des bestehenden Produkts schrittweise verbessert.

Verdeutlichen wir dies an einem Beispiel aus dem Bereich Software. Die meisten Anwendungen kommen auf den Markt, wenn die initial geforderten Anforderungen implementiert und erfolgreich getestet wurden. Üblicherweise ist dann Version 1.0 fertig und kann dem Kunden/Anwender zur Nutzung bereitgestellt werden. Die Notwendigkeit, enthaltende Fehler zu beseitigen, weitere Funktionalität nachzurüsten oder auch kleinere Verbesserungen zu erzielen, führen dazu, dass nach einer bestimmten Zeit eine neue Produktversion erscheint. Je nach Umfang der Anpassungen wird man das auch in der Bezeichnung deutlich machen. Größere Updates würde man üblicherweise mit einem Versionssprung auf 2.0, kleinere Verbesserungen mit einer Anpassung auf 1.1 dokumentieren. Beide Updates sind im Verständnis der Innovationstheorie inkrementell, denn das grundätzliche Produkt bzw. die Art und Weise der Lösung haben sich zwischen den Versionen nicht radikal ver-

ändert. Von einer radikalen Innovation würde man sprechen, wenn für die gleiche Problemstellung eine vollständig andere Lösung entwickelt wird. Das bisherige Produkt ist mit seinem Lebenszyklus dann schlagartig am Ende angelangt, und die Bedürfnisse der Kunden werden von der Produktinnovation abgedeckt. Dabei handelt es sich um einen vollständig anderen Ansatz oder eine ganz andere Herangehensweise.

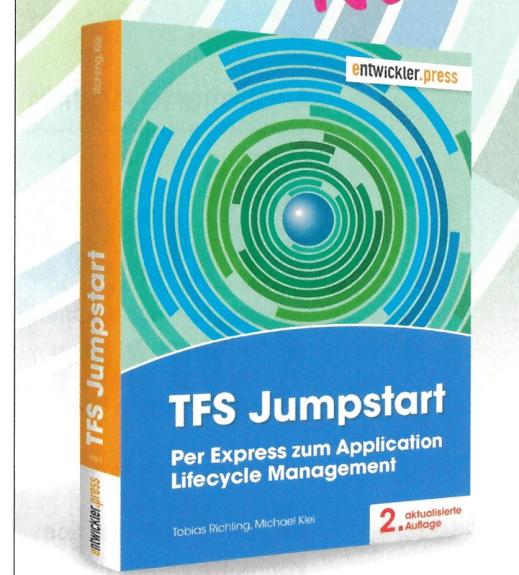
Konvergentes vs. divergentes Denken

Design Thinking baut auf ein abstraktes Durchdenken des Problems auf. Dabei werden die beiden in der Überschrift genannten Denkartens in unterschiedlichen Phasen – bei der Exploration des Problems- und des Lösungsraums – miteinander kombiniert [4]. Konvergentes Denken ist die konventionelle Art des Problemlösens durch logisches, streng rationales und planmäßiges Vorgehen. Die Lösung entsteht durch das Zusammenführen von vielen einzelnen Faktoren. Divergentes Denken ist dagegen eine offene, unsystematische und spielerische Herangehensweise zur Lösung des Problems. Denkblockaden werden dabei bewusst deaktiviert, um eine Vielzahl an Alternativen zu generieren. Die alternativen Ansätze sind nachfolgend in der Gruppe zu bewerten. Beim Explorieren des Problemraums werden in der divergenten Phase des Denkens so viele Informationen wie möglich über das Problem gewonnen. Diese werden in der konvergenten Phase analysiert und selektiert. Beim Explorieren des Lösungsraums werden in der divergenten Phase möglichst viele Ideen von Lösungen generiert, visualisiert, ausgetauscht und weiterentwickelt. In der konvergenten Phase werden sie auf Umsetzbarkeit geprüft. Außerdem versucht man, die Ideen zu kombinieren und in Beziehung zueinander zu setzen.

For-Profit- und Non-Profit-Organisationen (181 Antworten)	
Verbesserung der Arbeitskultur	71 %
Effizienzsteigerung der Innovationsprozesse	69 %
Verbesserung der Integration der Nutzer	48 %
Kostensenkung	18 %
Nur For-Profit-Organisationen (111 Antworten)	
Steigerung der Verkaufszahlen	29 %
Erhöhung der Profitabilität	18 %

Tabelle 1: Effekte von Design Thinking [1]

NEU!



Ein Blick in die Praxis

Heutzutage steht Design Thinking nicht nur für eine Kreativitätstechnik. Mittlerweile hat es sich zu einem Motor und Treiber des Unternehmenswandels entwickelt. Design Thinking wird in folgenden Bereichen eingesetzt [1]:

- Gestaltung von neuen Produkten und Services
- Bereitstellung von internem Coaching und Training
- Verbesserung interner Weiterbildung
- Erweiterung von Zusammenarbeit und Wissenstransfer
- Optimierung von internen Organisationsprozessen
- Verbesserung des Kundenverständnisses
- Erstellung zielgruppenorientierter Marketingkampagnen

Die Forscher vom HPI (Hasso-Plattner-Institut) haben zahlreiche Studien zum Design Thinking durchgeführt. Deren Erfolge sind in Tabelle 1 zusammengetragen. Die größten Erfolge sind im Bereich der Effizienzsteigerung der Innovationsprozesse festzustellen. Der finanzielle Mehrwert von Design Thinking konnte nur schwierig unmittelbar gemessen werden. Die Studien weisen jedoch darauf hin, dass sich die Unternehmensprozesse und die Kundenerfahrung nachhaltig verbessert haben. Es hat sich herausgestellt, dass Design Thinking sehr oft für die Verbesserung interner Prozesse eingesetzt wird; ebenso um Wissenstransfer und Kollaboration zu erleichtern und das Bild von der eigenen Kundschaft zu schärfen. Als Resultat entstehen neue Geschäftsmodelle, kreative Produkte, nutzerfreundlichere digitale Anwendungen oder auch innovative Softwaresysteme.

Fazit

Hoch innovative Branchen wie der IT-Sektor sind stetig auf der Suche nach Innovationen. Im Bereich der Softwareentwicklung sind Innovationen gerade im Bereich der Lösungskonzeption und der Verbesserung der Benutzererfahrung gefragt. Design Thinking kann dabei helfen, den Nutzer mit seinem Problem besser zu verstehen und eine neuartige Lösung zu entwickeln. Dabei tritt die Technik zunächst in den Hintergrund. Wichtiger sind ein umfassendes Problemverständnis und das Zulassen ganz anderer Herangehensweisen. Trotz aller Planung ist und bleibt Softwareentwicklung ein kreativ-schöpferischer Akt.



Dr. Veikko Krypczyk studierte und promovierte in Betriebswirtschaftslehre mit dem Schwerpunkt Wirtschaftsinformatik. Er ist Entwickler und Fachautor.



Olena Bochkor studierte Betriebswirtschaftslehre u. a. mit dem Schwerpunkt Wirtschaftsinformatik.

Weitere Informationen zu diesen und anderen Themen der IT finden Sie unter <http://larinet.com>.

Links & Literatur

- [1] HPI School of Design Thinking: <http://hpi.de/school-of-design-thinking/design-thinking/>
- [2] Faktenblatt Design Thinking: <https://static5.rkw-kompetenzzentrum.de/fileadmin/media/publications/2014/Innovation/Faktenblatt/20140924-Design-Thinking.pdf>
- [3] Seher, F.: „Designing Innovation. Design Thinking im Spannungsfeld zwischen Innovation und Großunternehmen“, 2011
- [4] Kretzschmar, O.: „Innovationsmethodik Design Thinking“, Stuttgart 2015

Bereits seit 2013 gewährt Microsoft auf verschiedenen Wegen kostenlosen Zugang zum Team Foundation Server (TFS). Von den Entwicklungen in seiner Umwelt hat dabei auch der TFS profitiert, z.B. durch eine vollständige Integration von Git. Der Trend zu mehr Offenheit bei Microsoft macht also auch vor deren ALM-Plattform nicht halt. Es ist heute einfacher denn je, sich mit dem TFS zu befassen und die Mächtigkeit dieser ALM-Plattform zu nutzen. Dieses Buch soll eine einfache Starthilfe auf dem Weg zum Application Lifecycle Management sein.

Mehr Infos: www.entwickler-press.de/TFS

Tobias Richling, Michael Klei

TFS Jumpstart

Per Express zum Application Lifecycle Management
(2. aktualisierte Auflage)

320 Seiten, Mai 2017

PRINT ISBN: 978-3-86802-169-1
Preis: 39,90 € / 41,10 € (A)

PDF ISBN: 978-3-86802-356-5
Preis: 27,99 €

EPUB ISBN: 978-3-86802-698-6
Preis: 27,99 €

entwickler.press

twitter.com/entwicklerpress

facebook.com/entwicklerpress

Backlogs priorisieren mit Costs of Delay und Monte-Carlo-Simulationen

Der Product Owner im Casino

Die Priorisierung des Backlogs ist eine zentrale Aufgabe für Product Owner. Gleichzeitig ist sie eine der schwierigsten Aufgaben. Viele Methoden konzentrieren sich auf strukturelle und inhaltliche Aspekte. Dieser Artikel möchte eine Methode zeigen, wie ökonomische Aspekte in Kombination mit Simulationen als Grundlage verwendet werden können.

von Gerrit Beine

Die agile Community hat etliche Methoden zur Priorisierung von Product Backlogs hervorgebracht. Einige davon können schon fast als Kanon der Product-Owner-Ausbildung betrachtet werden, z.B. MuSCoW. Andere Methoden sind eigentlich eine eigene Disziplin und beziehen schon Aspekte aus der Erstellung von Geschäftsmodellen ein, z.B. Story Mapping. All diesen Methoden ist gemeinsam, dass sie einen starken Fokus auf die Inhalte der Backlog Items legen und vor allem ein gemeinsames Verständnis bei allen Beteiligten schaffen wollen. Denn Fakt ist: Ohne dieses Verständnis lassen sich Projekte nicht erfolgreich durchführen.

Neben dem Verständnis gibt es aber eine weitere Dimension, die für den Erfolg eines Projekts entscheidend ist: die Wirtschaftlichkeit. Zwar wird gerne gesagt, dass Backlog Items entsprechend ihres Kosten-Nutzen-Verhältnisses priorisiert werden sollen, es gibt aber nur wenige konkrete Vorschläge, wie sich dieser Nutzen quantifizieren lässt. An dieser Stelle setzt die Methodik der Costs of Delay (Verzögerungskosten) an. Sie ist eine hervorragende Ergänzung zu Story Mapping, Impact Mapping oder anderen Methoden und fügt den ökonomischen Blickwinkel hinzu.

Die Grundidee der Costs of Delay

Die Costs of Delay wurden von Don Reinertsen [1], [2] beschrieben und sind mittlerweile ein anerkanntes ökonomisches Werkzeug. Die Idee besteht darin, dass es

neben den Kosten, die durch eine unternehmerische Aktivität entstehen, auch Kosten gibt, die entstehen, wenn diese Aktivität zu lange dauert oder ihre Ergebnisse zu spät verfügbar sind. Diese Kosten sind im einfachsten Fall entgangene Einnahmen durch Verzögerung, in komplizierten Fällen Strafzahlungen durch zu späte Lieferung oder komplexe soziale Folgen.

Oft kommt es vor, dass Costs of Delay mit Opportunitätskosten gleichgesetzt werden. Dies ist aber falsch. Opportunitätskosten sind einfach ausgedrückt entgangene Gewinne, die unternehmerische Aktivitäten hätten erzielen können. Costs of Delay entstehen, weil eine unternehmerische Aktivität gar nicht, zu spät oder nicht effizient durchgeführt wird. Reinertsen sagt, dass 85 Prozent aller Produktmanager die Costs of Delay nicht kennen. Außerdem seien Menschen im Allgemeinen sehr schlecht in der Abschätzung der Costs of Delay. In Untersuchungen hat er ermittelt, dass die Costs of Delay im Mittel um einen Faktor 50 unterschätzt werden. Er legt Produktmanagern nahe, sich intensiv mit dieser Idee zu beschäftigen und Costs of Delay tatsächlich zu quantifizieren. Natürlich sind die Costs of Delay für Backlog Items Prognosen über zukünftige Entwicklungen. Damit sind sie, je nachdem wie viel Wissen über die Zukunft verfügbar ist, mehr oder weniger exakt. Eine bewusste Beschäftigung mit den Costs of Delay hilft Product Owners aber in jedem Fall, eine bessere Einschätzung abzugeben als das Ignorieren dieser Kosten und ein rein inhaltliches Priorisieren von Backlogs. Basierend auf den Ideen von Reinertsen gibt es vier Basismodelle der Costs of Delay:

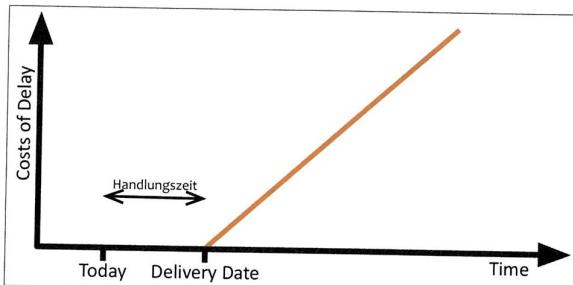


Abb. 1: Costs of Delay im Standardfall: Die Kosten steigen ab einem definierten Zeitpunkt linear; die Verzögerung wird also immer teurer

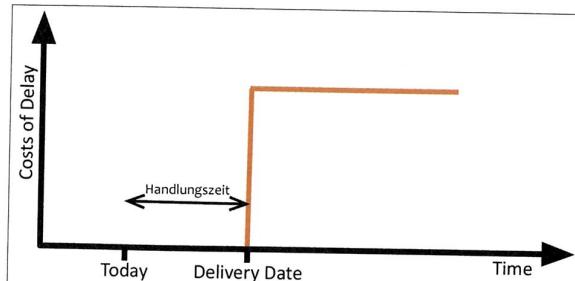


Abb. 2: Costs of Delay im Fall eines festen Zeitpunkts: Wird dieser Zeitpunkt erreicht, steigen die Kosten auf einen definierten Wert

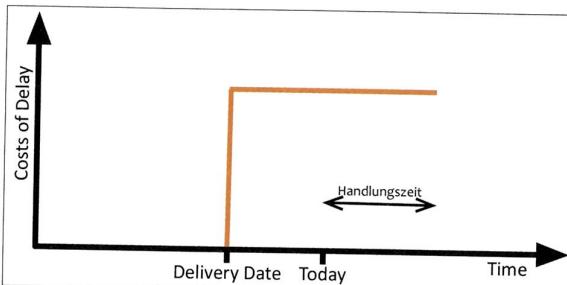


Abb. 3: Costs of Delay in einer Expedite-Situation: Die Kosten sind schon vor einer Weile entstanden, allerdings war das nicht bekannt

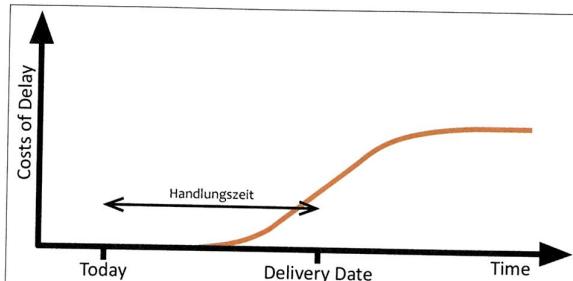


Abb. 4: Costs of Delay von Intangibles: Es ist im Voraus nicht genau bekannt, wann die Kosten steigen werden und wie hoch

- Linear mit der Verzögerung steigende Kosten (Standard)
- Kosten, die ab einem definierten Zeitpunkt entstehen (Fixed Date)
- Plötzlich entstehende Kosten (Expedite)
- Kosten, die wahrscheinlich entstehen, aber nicht exakt zu prognostizieren sind (Intangibles)

Im Standardfall sind die Costs of Delay zunächst gering und steigen sukzessive mit der Zeit an (**Abb. 1**). Die Verzögerung wird also im Laufe der Zeit immer teurer. Durch die Linearität lässt sich aber gut prognostizieren, bis wann eine Entscheidung getroffen oder ein Ergebnis fertig gestellt sein muss, um ein optimales Verhältnis zwischen Costs of Delay und der entgegenstehenden Investition zu erhalten.

Ähnlich einfach lassen sich Costs of Delay handhaben, die dem Fixed-Date-Modell folgen (**Abb. 2**). Die Kosten steigen zwar ab diesem Zeitpunkt rapide an, da der Zeitpunkt aber bekannt ist, besteht die Möglichkeit, rechtzeitig zu handeln. Hilfreich ist das Fixed-Date-Modell insbesondere dann, wenn eine Entscheidung nicht zu früh getroffen werden soll. Denn das wäre Verschwendug.

Besonders spannend sind die beiden Fälle Expedite und Intangibles. Dabei könnte man Expedite als einen Sonderfall des Fixed-Date-Modells betrachten, in dem der Zeitpunkt zu spät bekannt wurde oder bereits in der Vergangenheit liegt (**Abb. 3**). Ein Beispiel dafür ist ein schwerwiegender Fehler in einer produktiven Software. Hier fallen ab dem Moment des Eintretens Verzögerungskosten an, die praktisch unbegrenzt wachsen, je länger es dauert, den Fehler zu beheben. Neben solchen

Fehlern können aber auch andere nicht vorhersehbare Ereignisse, wie der Eintritt eines neuen Marktteilnehmers, zu Verzögerungskosten führen, die dem Expedite-Modell folgen.

Während sich in den drei beschriebenen Varianten die Costs of Delay in der Regel gut bis sehr gut quantifizieren lassen, ist das bei Intangibles oft nicht der Fall (**Abb. 4**). Es ist im Voraus nicht genau bekannt, wann die Kosten steigen werden und wie hoch; eine exakte Quantifizierung kann nur im Nachhinein erfolgen; dann kann allerdings auch aus dem Intangible ein Expedite werden. Bei den Intangibles handelt es sich um Themen wie technische Schulden, Forschungsaufgaben oder auch Investitionen in Bereichen wie Human Resources. Für solche Beispiele können die Costs of Delay häufig nur indirekt und a posteriori quantifiziert werden. Natürlich ist es aber auch in diesen Fällen sinnvoll, die Costs of Delay möglichst gut zu prognostizieren. Dies kann durch Erfahrungen und Expertise in den betreffenden Gebieten auch in ausreichender Qualität gelingen.

Die Herausforderung des Product Owners

Ein Product Owner kennt die Costs-of-Delay-Modelle, die für sein Projekt relevant sind. Einige Organisationen nutzen Costs of Delay bereits zur Bewertung von unternehmerischen Entscheidungen und ergänzen damit die üblichen Methoden wie Return on Investment (ROI), Net Present Value (NPV) oder Internal Rate of Return (IRR). Innerhalb eines Projekts oder für eine Produktentwicklung sind Costs of Delay jedoch nicht immer einfach umzusetzen. Denn die Costs of Delay einzelnen Features zu quantifizieren, kann eine große Herausforderung werden. Wie beschrieben, fällt es

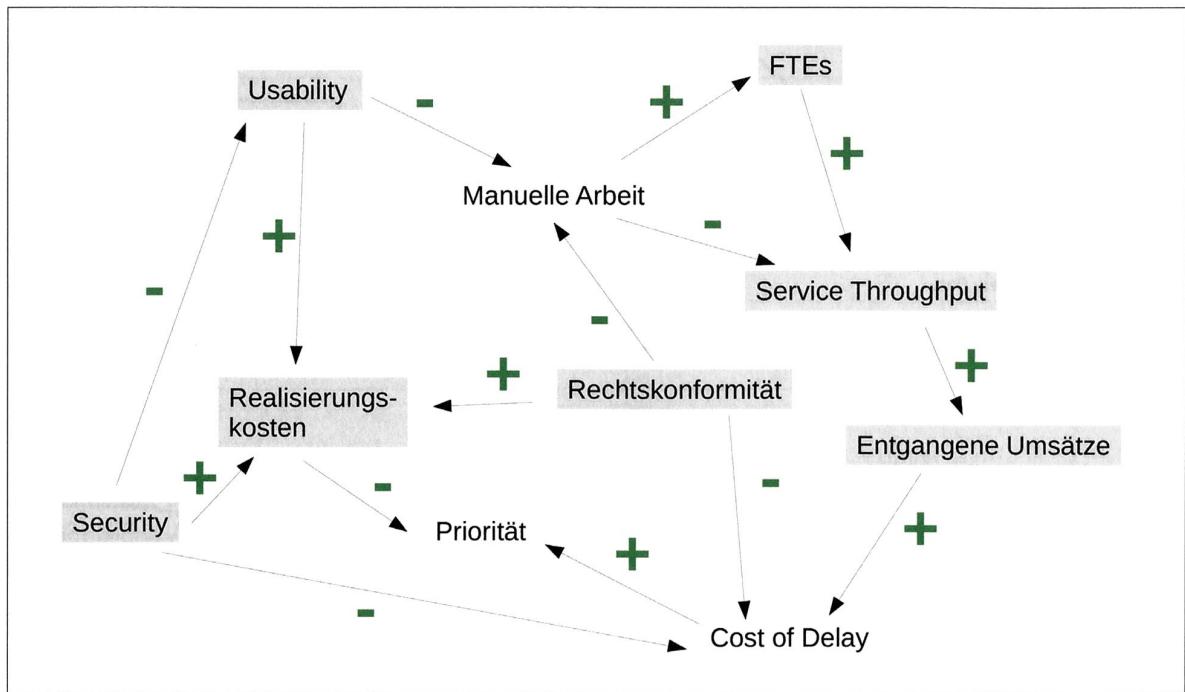


Abb. 5: Wirkdiagramm für ein Costs-of-Delay-Modell

uns Menschen schwer, Costs of Delay gut zu schätzen. Eine Schätzung ist zwar immer noch besser als gar keine Betrachtung, reicht aber bei Weitem nicht aus, um gute Entscheidungen zu treffen. Vielen Organisationen fällt es schon schwer, einzelne Features von Software überhaupt mit einem ökonomischen Wert abseits des Aufwands für ihre Erstellung zu quantifizieren. Product Owner benötigen also eine Herangehensweise, die es ihnen erlaubt, eine solche Bewertung zu ermitteln.

Ist es gelungen, ein Modell zur Ermittlung der Costs of Delay einzelner Features zu erstellen, steht die nächste Herausforderung an: Costs of Delay sind kein fester Parameter. Sie verändern sich im Laufe der Zeit und hängen damit von der Reihenfolge der Features ab, die sie wiederum bestimmen sollen. Damit wird die Priorisierung des Product Backlogs für den Product Owner zu einem Optimierungsproblem.

Je nachdem, von wie vielen Aspekten die Costs of Delay bestimmt werden, ist die Berechnung dieses Optierungsproblems aufwendig. Da die Priorisierung eines Product Backlogs im Verlauf eines Projekts auch volatil ist, muss die entsprechende Berechnung regelmäßig wiederholt werden. Manchmal muss auch noch das zugrunde liegende Modell angepasst werden, weil neue Erkenntnisse gewonnen wurden oder sich Rahmenbedingungen verändert haben und natürlich, weil bereits Teile des Projekts realisiert wurden. Der Fokus der Priorisierung für ein Projekt oder eine Produktentwicklung sollte immer auf der Frage liegen, was als Nächstes getan werden muss. Diese Aspekte sprechen dafür, eine Monte-Carlo-Simulation zu nutzen, um eine Priorisierung für das Product Backlog zu ermitteln. Monte-Carlo-Simulationen erzeugen viele ver-

schiedene Priorisierungen des Product Backlogs und ermitteln jeweils die kumulierten Costs of Delay. Mit relativ wenig Aufwand erhält ein Product Owner ein priorisiertes Backlog, das auf Basis des zum Zeitpunkt der Priorisierung verfügbaren Wissens mit großer Wahrscheinlichkeit optimal ist.

Wirkmodelle als Basis der Monte-Carlo-Simulation

Um eine solche Monte-Carlo-Simulation durchführen zu können, benötigt man zunächst ein Wirkmodell, mit dem sich die unterschiedlichen Treiber für die Costs of Delay und damit die Priorität von Backlog Items beschreiben lassen. Ein Beispiel für ein solches Wirkdiagramm ist in Abbildung 5 dargestellt. Das Diagramm zeigt die Beziehungen und Wirkungen unterschiedlicher Aspekte der Features in einem Projekt. Werden die Aspekte pro Feature bewertet, kann das Modell Grundlage für eine Monte-Carlo-Simulation sein. Verschiedene Treiber wirken verstärkend (+) oder abschwächend (-) auf einander und letztendlich auf die Costs of Delay. Diese wiederum wirken verstärkend auf die Priorität eines Backlog Items. Je höher die Costs of Delay sind, desto höher ist die Priorität des Backlog Items.

Welche Aspekte eine Rolle für ein solches Wirkdiagramm spielen können, ist dabei von Projekt zu Projekt unterschiedlich. Fundamental ist in jedem Fall das Geschäftsmodell oder der Business Case, die hinter einem Produkt oder Projekt stecken. Diese liefern bereits viele Aspekte, die in ein solches Wirkdiagramm einfließen können.

Das Wirkdiagramm kann anders als rein monetäre Mechanismen wie IRR oder NPV auch intangible As-

pekte enthalten. Damit wird es möglich, die Wirkung von Themen wie Usability, Rechtskonformität, IT Security oder auch Umweltschutz auf die Costs of Delay zu modellieren. Methoden zur Quantifizierung solcher Intangibles hat Douglas Hubbard ausführlich in seinem Buch „How to Measure Anything“ [3] beschrieben.

Ein solches Wirkdiagramm und das zugrunde liegende Modell können groß und kompliziert werden. Die Arbeit mit dem Wirkdiagramm ist dennoch leicht, weil sich die einzelnen Themen zunächst unabhängig voneinander betrachten lassen. Ein Usabilityexperte kann die Wirkung einzelner Backlog Items auf die User Experience bewerten, während eine Juristin das Gleiche für die Rechtskonformität machen kann. Im Modell wird das Wissen aller zusammengeführt. Das Wirkdiagramm ist dadurch als Erklärung der Zusammenhänge der Priorisierung im Dialog mit Stakeholdern hilfreich. Erfahrungen zeigen, dass viele Stakeholder sich wohler fühlen, wenn sie nur Aspekte ihrer Domäne beurteilen müssen und nicht über die Wichtigkeit eines Backlog Items als Ganzes entscheiden sollen. Damit kommt die Dekonstruktion der Treiber für die Costs of Delay den Menschen entgegen, hilft Schätzungen zu verbessern und ermöglicht es, schwierige Themen, die keinen direkten monetären Nutzen liefern, korrekt zu priorisieren.

Auf ins Casino: Die Monte-Carlo-Simulation

Mit den gewonnenen Daten und dem Modell lässt sich nun eine Monte-Carlo-Simulation füttern. Diese erzeugt eine große Menge zufälliger Reihenfolgen der bewerteten Backlog Items und berechnet für jedes Backlog Item die Costs of Delay anhand der Position im Backlog, der voraussichtlichen Umsetzungsgeschwindigkeit des Teams und natürlich der anderen beschriebenen Wirkungen. Die kumulierten Costs of Delay sind das Maß für die Güte der jeweiligen Priorisierung. Dabei gilt: Eine Priorisierung, bei der geringere Costs of Delay entstehen als bei den vorher berechneten, ist eine bessere Priorisierung.

Üblicherweise werden in einer solchen Monte-Carlo-Simulation nicht alle möglichen Priorisierungen berechnet. Die Simulation wird entweder nach Erreichen einer definierten Anzahl von Durchläufen oder nach Ablauf eines Zeitfensters beendet. Die bis dahin ermittelte beste Priorisierung wird dann als Entscheidungsgrundlage verwendet. Natürlich kann es sein, dass diese Priorisierung nicht die bestmögliche Priorisierung ist. Das ist sogar wahrscheinlich. Sie ist aber, unter der Voraussetzung, dass das Wirkmodell durchdacht ist und eine genügend große Anzahl von Simulationen stattgefunden hat, ausreichend gut, um zu entscheiden, was als Nächstes getan wird.

In regelmäßigen Abständen, wenn z. B. das nächste Backlog Grooming, Replenishment oder Release Planning ansteht, wird die Simulation erneut durchgeführt und die Priorisierung der Backlog Items erneut festgelegt. Dabei kann es sein, dass Backlog Items herausfallen, neue hinzukommen, Bewertungen oder sogar das

Mit Monte-Carlo-Simulationen erhält ein Product Owner mit relativ wenig Aufwand ein priorisiertes Backlog.

Wirkmodell verändert wurden. Die Monte-Carlo-Simulation liefert zuverlässig die wahrscheinlich beste Reihenfolge für das Product Backlog.

Fazit

Product Owner benötigen Werkzeuge, um gute und nützliche Produkte mit ihren Teams zu bauen. Neben vielen bekannten Werkzeugen, die vor allem auf inhaltliches Verständnis und Kollaboration abzielen, werden auch solche, die ökonomische Aspekte handhabbar machen, immer wichtiger. Dabei müssen insbesondere die nicht materiellen und indirekt wirkenden Aspekte berücksichtigt werden. Die wenigsten Product Owner können die dafür notwendigen Bewertungen allein stemmen. Entwicklungsteams und Stakeholder müssen sie dabei unterstützen. Das gelingt am effektivsten, wenn alle ihr Domänenwissen sinnvoll einbringen können und dieses gezielt zusammengeführt wird. Aufgaben wie die Priorisierung des Backlogs können Product Owner dann getrost Computern überlassen. Das hat neben dem Effekt, dass es Zeit für wichtige Aufgaben schafft, noch die Vorteile, dass es unpolitisch ist und allen Beteiligten entgegenkommt, weil sich niemand mehr gegen etwas entscheiden muss. Außerdem macht das Entwickeln einer solchen Simulation jede Menge Spaß.



Gerrit Beine ist Managing Consultant bei der adesso AG mit den Schwerpunkten Agilität und Softwarearchitektur. Er ist regelmäßig Sprecher auf Konferenzen und beschäftigt sich gerne mit außergewöhnlichen Fragestellungen zur Softwareentwicklung.

Links & Literatur

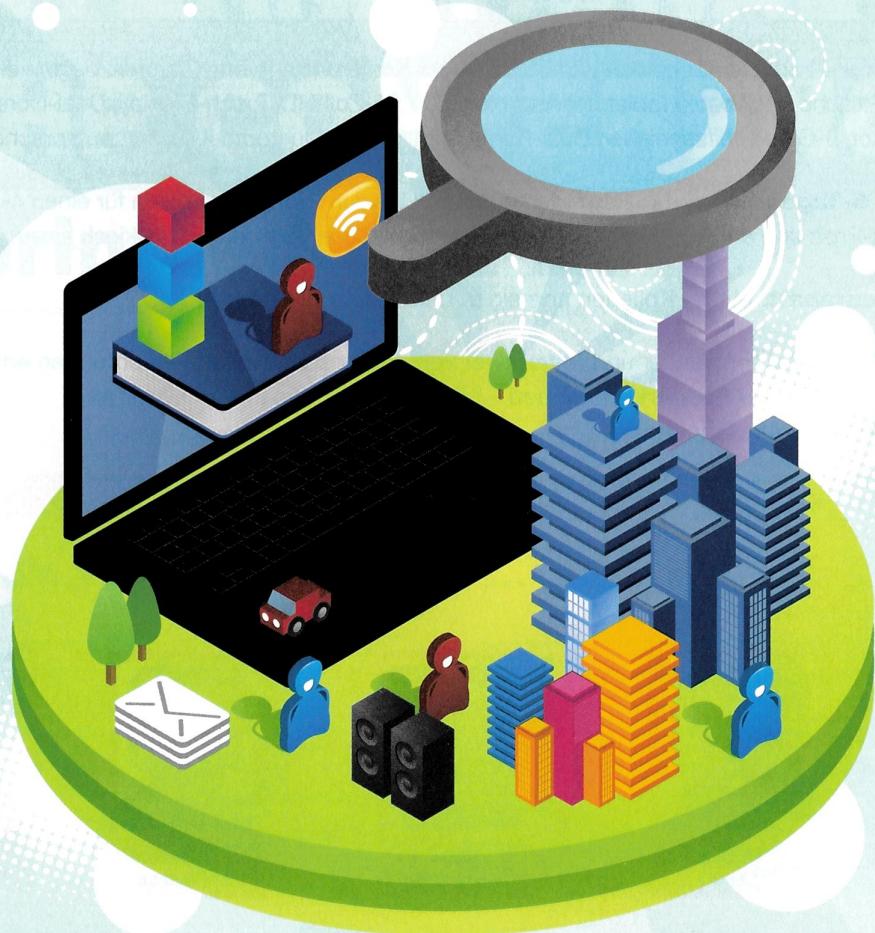
- [1] Reinertsen, Donald G.: „Managing the Design Factory“, Free Press, 1997
- [2] Reinertsen, Donald G.: „The Principles of Product Development Flow“, Celeritas Publishing, 2014
- [3] Hubbard, Douglas W.: „How to Measure Anything“, Wiley, 2014

Minimalismus für Softwareunternehmen

Weniger ist mehr

Oftmals verrennt man sich in Kleinigkeiten, obwohl weniger doch manchmal mehr wäre. Die Frage sollte also lauten: Ist Minimalismus für softwareentwickelnde Menschen eine gute Idee? Ich glaube: Ja!

von Dennis Wilson



© iStockphoto.com/ryccio

Welche Schuhe trage ich heute? Soll ich lieber einen Frappuccino Latte Macchiato oder einen Mocha Latte Palermo bestellen? Oh, eine E-Mail. Habe ich heute Morgen wirklich die richtigen Schuhe angezogen? Welche Programmiersprache lerne ich als Nächstes? Welches Framework? Oh, eine E-Mail. Gehe ich heute Abend zum Sport oder mache ich doch lieber nur einen Filmabend? Ach nein, ich muss ja das Auto zum Service bringen. Oh, eine E-Mail. Puh, heute schon wieder nichts geschafft. Schreibe ich halt noch schnell E-Mails.

So oder so ähnlich sehen vermutlich die Tage vieler Menschen in der IT-Branche aus. In einer durch und durch komplexen Welt sieht man sich täglich, bewusst oder unbewusst, mit vielen Entscheidungen konfrontiert. Den einen fällt das mehr, den anderen weniger auf. Regelmäßig findet man Productivity- und Downshifting-Literatur auf den Toplisten der Buchhändler, und auch in Zeitschriften sind diese Themen hoch im Kurs. Die Welt will minimal, agile und lean werden. Das ist sogar vollkommen ernst gemeint und eventuell sogar die logische Konsequenz aller Dinge. Dieser Artikel nimmt sich deshalb des Themas Minimalismus in der Softwareentwicklung an.

Minimalismus in der Softwareentwicklung

Was hat das nun konkret mit Minimalismus zu tun? Wenn die Definitionen von Minimalismus auch noch so weit gefasst sind, geht es im Kern immer um das Gleiche: Mit so wenig wie möglich gleichbleibende oder sogar noch bessere Ergebnisse erzielen. Sämtliche Definitionen teilen im Kern die Aussage, dass

- a) es zu begrüßen ist, Dinge zu entfernen, wenn Sie nicht unmittelbar oder signifikant zum Ergebnis beitragen, und
- b) sich stattdessen auf wenige, wesentliche, zielerfüllende Elemente zu beschränken.

Wie genau das aussieht, ist dann letzten Endes immer kontextabhängig. In der Designbranche gilt beispielsweise Dieter Rams' These [2] „Gutes Design ist so wenig Design wie möglich“ als gesetzt und anerkannt. Das ist ein Grundsatz, der lange Zeit als der Wettbewerbsvorteil von Apple gehandelt wurde, das stark durch Dieter Rams inspiriert ist. Man fokussiert sich dabei auf die wesentlichen Elemente, die den eigentlichen Mehrwert ausmachen. Dinge, die vom Mehrwert oder der Kernfunktion ablenken oder unnötigen Aufwand bedeuten, werden entfernt.

Nach diesem kleinen Exkurs ist es jetzt selbstverständlich spannend zu erkunden, wie minimalistische Prinzipien im Entwicklungsalltag eingebracht werden können. Dazu stelle ich im Folgenden zwei wesentliche Ansätze vor:

1. Mehr Systeme und weniger Entscheidungen

Menschen wie Einstein, Jobs und Zuckerberg haben es bereits vorgelebt: sie trugen jeden Tag dasselbe Outfit.

**Es geht immer um das Gleiche:
mit so wenig wie möglich gleich-
bleibende oder bessere Ergebnisse
erzielen.**

Für Außenstehende wirkt das zunächst etwas befremdlich; schließlich kommunizieren wir und identifizieren und differenzieren uns über Kleidung. Wem es finanziell gut geht, zeigt das gerne über seine Kleidung.

Die gewählte Tristheit hat jedoch einen cleveren Hintergrund: Zuckerberg beispielsweise möchte dadurch seine Energie für wichtigere Entscheidungen aufbewahren [3]. Damit spielt er auf eine psychologische Theorie, das so genannte „Decision Fatigue“ [4] an. Der Theorie nach sinkt somit die Entscheidungskraft von Menschen, je mehr Entscheidungen Sie pro Tag treffen. Regenerationsphasen können die Entscheidungsfähigkeit wiederherstellen, und genauso können „Entscheidungsressourcen“ durch Systeme eingespart werden. Im Klartext heißt das: Besitzt man wenige Dinge, oder hat Systeme, die Entscheidungen vereinfachen, kann man mehr Konzentration/Energie/Fokus für Dinge aufwenden, die diese Aufmerksamkeit tatsächlich benötigen.

Automation

Wer dennoch an seiner Garderobe hängt, kann bereits mit systematischen Arbeitsblöcken viel erreichen. Denn häufig (hintereinander) das Arbeitsmuster wechseln hat hohe Initialkosten und führt zu geistiger Ermüdung. Sprich: Wenn ich von Code zu Telefonat, zu Meeting, zum Parkplatz und wieder zum Code springe, geht das alles zu Lasten der Arbeitsleistung. Automation ist eines der Zauberworte. Geht man davon aus, dass die Summe unseres täglichen Handelns die Definition unserer Identität bildet (ein Programmierer programmiert, ein Bäcker backt), dann gilt: Du bist, was du nicht bereits automatisiert hast. Will man Beispielsweise ein Bug Hunter sein, oder doch lieber den Vorteil automatischer Softwaretests nutzen? Demnach haben Einstein, Jobs und Zuckerberg vor allem ihren Kleiderschrank automatisiert. Doch auch hier ist Vorsicht geboten: Je mehr Automaten man sich anschafft, desto schwieriger wird es an einem Punkt, sich um all diese Automaten zu kümmern.

2. Was du besitzt, besitzt am Ende dich

Was zunächst wie eine kommunistische Parole klingt, ist vor allem als eine Metapher zu verstehen. Was ich damit meine: Je mehr Teile man einem System hinzufügt, umso mehr Teilen muss man langfristig Aufmerksamkeit schenken. Schon ein Teil alleine möchte gerne

gepflegt werden. Der Pflegeaufwand steigt jedoch mit jedem Ding an. Was nämlich oftmals außer Acht gelassen wird, ist, dass Dinge neben einem Initialaufwand auch immer Folgeaufwände oder Risiken mitbringen. Nennen wir das Ganze der Einfachheit halber Bedürfnisse.

Initialaufwände können finanzielle Aufwände sein. Auch einmalig anfallende Zusatzarbeit gehört dazu. Ein Auto ist schnell gekauft, doch muss es mindestens einmal transportiert werden. Anschließend versichert man es oder stellt es irgendwo gut unter (Initialaufwand). Wird es dann regelmäßig genutzt, braucht es Wartung und Benzin. Im Winter möchte es neue Reifen, Kühlflüssigkeit, Enteiser; und überhaupt möchte es eine regelmäßige Innen- und Außenpflege genießen. Andernfalls geht es kaputt und braucht spätestens dann einen Werkstattbesuch. Alles Folgeaufwand.

Vor allem der Folgeaufwand steigt oftmals proportional mit der Anzahl der Dinge, um die es geht. Während das bei Initialaufwand manchmal weniger wichtig ist, macht es bei den Folgeaufwänden schon einen Unterschied. Es mag immer noch wesentlich weniger aufwendig sein, dreißig Autos auf einmal zu kaufen als dann dreißig Autos auf einmal zu pflegen.

Das ist quasi eine Gesetzmäßigkeit, die sich in allen Handlungen wiederfinden lässt, die Dinge einem Inventar oder Projekt hinzufügen; auch in der immateriellen Branche der Softwareentwicklung. Hier ist vor allem die Verlockung, viele Dinge anzuhäufen, relativ groß, da der Initialaufwand oftmals verlockend gering ist: Software und Betriebssysteme bekommt man im Internet nämlich, ohne Geld dafür zu bezahlen.

Versteckte Folgeaufwände

Klassiker aus den Augen verlorener Folgeaufwände in der Softwareentwicklung sind beispielsweise die folgenden:

- Versionssprünge und Sicherheitsupdates von Libraries und Betriebssystemen
- Spiegeln genutzter 3rd Party Libraries, um jederzeit Deployment-ready zu bleiben
- Anfälligkeit eigenen Glue-/Frameworkcodes, der ggf. regelmäßig auf externe Libraries angepasst werden muss
- Daten-Backups, Failover, Error-Logging, Monitoring und Resilience selbst gehosteter Dienste
- Kompatibilitätsmanagement eigener Services untereinander
- Schulung und Wissenstransfer innerhalb des Teams
- Koordinierte Vertretungsplanung oder Erreichbarkeit bei Mitarbeiterausfällen

Wem also beispielsweise Betrieb und Erhalt einer Anwendung oder von Services über den Kopf wachsen, der kann unter anderem

1. dafür sorgen, dass (API-)Services weniger kleinteilig werden. Dafür vielleicht immer noch einen sinnig geschnittenen SOA-Ansatz verfolgen.

2. versuchen, sich auf wesentliche Kompetenzen zu konzentrieren und das Operating einzelner Dienste (Hosting, Mails, Monitoring) externen Anbietern überlassen.

Durch bewusste Entscheidung für schmale Lösungen und Strukturen entfallen zusätzliche Administration, Pflege oder Koordinierung. Auf diese Art frei gewordene Ressourcen (Zeit/Geld) können somit auf Dinge gelenkt werden, für die man als Entwickler/Arbeitgeber/Unternehmen wirklich von seinen Kunden bezahlt wird. Zugleich schafft man dennoch mehr seiner eigentlichen Arbeit. Manchmal ist es in der Tat erst gar nicht so offensichtlich, welche der Entscheidungen den größeren Vorteil bringen. Dazu lohnt es sich durchaus, Beratungen zu konsultieren und einen extern geleitete Workshops abzuhalten.

tl;dr

Nochmal rückblickend: Was hat Ephemeralization in der IT-Branche mit Minimalismus zu tun? Je weniger Dinge man in einer Organisation selbst besitzt, desto mehr kann man sich auf das eigentliche Unternehmensziel konzentrieren. Trenne dich von Dingen, die nicht zu deinen Kompetenzen gehören. Wenn möglich, finde sinnvolle Gruppierungen von Dingen, die einen ähnlichen Folgeaufwand besitzen. Muss man wirklich einen eigenen Mailserver betreiben, nur, weil er scheinbar kostenlos ist? Muss das Ticketsystem wirklich selbst gehostet werden?

Oftmals lohnt es sich, Geld für Dienste zu bezahlen, weil die versteckten Folgeaufwände beim Selbstmachen übersehen werden. Auch hier mag es unter Umständen einen gewissen Schwellenwert geben, ab dem es wesentlich komplexer wird, externe Dienste zu organisieren als interne Wertschöpfung zu betreiben. Doch darüber werde ich ein anderes Mal schreiben.



Hauptberuflich begleitet und schult **Dennis John Wilson** als selbstständiger IT-Berater Entwicklerteams. Nebenberuflich schreibt er für unterschiedliche Magazine und spricht gelegentlich auf Konferenzen. Trotz mehr als fünfzehn Jahren Berufserfahrung liebt er Softwareentwicklung noch immer wie beim ersten Hello World.



www.denniswilson.de

Links & Literatur

- [1] <https://phpconference.com/session/doing-everything-with-nothing-architecture-ephemeralization-and-the-law-of-accelerating-returns/>
- [2] https://de.wikipedia.org/wiki/Dieter_Rams#Zehn_Thesen_f%C3%BCr_gutes_Design
- [3] <http://www.independent.co.uk/news/people/why-mark-zuckerberg-wears-the-same-clothes-to-work-everyday-a6834161.html>
- [4] https://en.wikipedia.org/wiki/Decision_fatigue

Pair Programming als Ausweg aus der Qualitätsfalle?

Never code alone

Teamwork in Entwicklerteams ist heute der einzige Weg, den hohen technischen Anforderungen an Webapplikationen gerecht zu werden und nachhaltige Softwareentwicklung zu betreiben. Der einfache Blick über die Schulter und der abschließende Blick auf den entstandenen Code beim obligatorischen Codereview reichen da allerdings nicht aus. Richtig eingesetzt indes, kann Pair Programming die Softwarequalität erhöhen und Arbeitsbedingungen in Unternehmen verbessern.

von Roland Golla

Durch Pair Programming werden Unternehmen in einem Markt mit starkem Fachkräftemangel zu attraktiven Arbeitgebern. Denn bei den vielen Möglichkeiten, aus denen Programmierer heute auswählen können, ist es wichtig, mehr als MacBook, Kicker und Club Mate zu bieten.

Vielmehr stellt Pair Programming, ob nun zu zweit oder im Team umgesetzt, klare technische und räumliche Mindestanforderungen. Zum einen braucht man einen separaten Raum mit einem großen Monitor mit guter Auflösung. Auf keinen Fall einen Beamer, zumal hier stundenlang an Quellcode gelesen werden muss – das kann man nicht mit einem Tapetenmuster im Hintergrund. Und zum anderen wird noch eine Funktastatur mit Funkmaus benötigt. Nur dann kann jeder in die Programmierung integriert werden und auch mal übernehmen.

Die Regeln von Pair Programming

Auch wenn Programmierer häufig in Teams verteilt an Aufgaben arbeiten, haben sie doch sehr selten gemeinsam bzw. gleichzeitig an Quellcode gearbeitet. Das kann gerade zu Anfang eine eher unangenehme Situation sein, die eine hohe Sozialkompetenz erfordert. Darauf empfiehlt sich hierbei der Einsatz eines erfahrenen Moderators, denn viele Programmierer haben noch nie mit direktem Feedback gearbeitet und sich im Prozess selbst entsprechend selten mit den Ideen und Gedanken anderer auseinandersetzen müssen. In der PHP-Entwicklung gibt es zudem eine sehr strenge Hierarchie vom Senior bis zum Junior. Doch am Ende geht es darum, gemeinsam an Lösungen und effektiver als Team zu arbeiten, indem man gemeinsames Wissen auf eine Problemstellung anwendet. Hier ein paar Regeln bzw. Gebote, die das Leben beim Pair Programming leichter machen:

- **Plant vor dem Schreiben!** Leider wird die Lösung für eine Aufgabe viel zu selten geplant. Test-driven Development (TDD) ist eine Möglichkeit, Software zu planen und wie gewünscht zu erstellen. Natürlich ist es eine Methode der agilen Entwicklung, die das Ziel verfolgt, möglichst wenig Code für die Lösung zu erstellen. Es geht aber auch ohne TDD. Welche Methoden brauche ich? Welche Methoden werden als *public* der Applikation zur Verfügung gestellt? Wie gehe ich mit Abhängigkeiten um? Diese Fragen sollten im Vorfeld im Team geklärt werden.
- **Redet über Codezeilen!** Es bringt nichts, irgendwo hinzuziegen und zu erwarten, dass der andere euch unmittelbar versteht. Redet stattdessen über Zeilenummern! Das macht alles wesentlich einfacher.
- **Führt einen Dialog miteinander!** Wenn jemand etwas nicht versteht, hat das nur selten mit der Akustik zu tun. Es bringt also nichts, dann noch einmal denselben Satz lauter zu wiederholen. Versucht stattdessen, einen Dialog zu führen. Es ist nicht die Lösung, die Tastatur zu übernehmen und einfach alles runterzuschreiben. Zielführender ist es, kurz aufzustehen und direkt am Monitor zu erklären, was ihr meint. Zwar kann man in kleineren Gruppen auch mal schriftlich kommunizieren, doch das sollte warten, bis man sich eingespielt hat und dieselbe Sprache im Code spricht.
- **Teilt euer Wissen miteinander!** Es kommt immer wieder vor, dass ein Programmierer etwas weiß, was sein Partner nicht weiß. Dieses Wissen gilt es zu teilen, ohne den Gegenüber zappeln zu lassen oder gar bloßzustellen. Arbeitet lösungsorientiert und nach vorne! Nicht gegeneinander. Bringt eure jeweiligen Skills ein und ergänzt euch im Team, damit die Softwarequalität besser wird. Das ist ein wichtiger Punkt.
- **Verliert euch nicht in Details!** Pair Programming mit zwei oder mehr Programmierern ist ein erheblicher

Kostenfaktor. Daher müssen vereinbarte Ziele auf jeden Fall erreicht werden. Klar geht alles immer noch ein wenig besser, aber das darf auf keinen Fall zur Maxime werden. Schließlich gibt es später noch das Codereview. Zunächst einmal geht es darum, fertig zu werden und nach vorne zu kommen.

Listing 1: „preparePrint“

```
public function preparePrint($id) {
    $returnedtag = Mage::getModel('glsbox/shipment')>getCollection()>addFieldToFilter(
        'id', $id)>getFirstItem()>getGlsMessage();
    if($returnedtag === false || $returnedtag == "") {
        return false;
    } else {
        $tags = $this>parseIncomingTag($returnedtag);
        if(is_Array($tags)) {
            $service = Mage::getModel('glsbox/shipment')>getCollection()>addFieldToFilter(
                'id', $id)>getFirstItem()>getService();
            if ($service == "business" || $service == "cash") {
                $glsService = Mage::getModel('glsbox/label_gls_business');
            }
            elseif ($service == "express") {
                $glsService = Mage::getModel('glsbox/label_gls_express');
            }
            if($glsService != null) {
                $glsService>importValues($tags);
                return $glsService>getData();
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}
```

Listing 2: Nach dem Refactoring

```
public function preparePrint($id) {
    if(!is_int($id)) {
        throw new \InvalidArgumentException('no integer');
    }

    /** @var GlS_Unibox_Model_Shipment $firstItem */
    $firstItem = Mage::getModel('glsbox/shipment')>getCollection()
        >addFieldToFilter('id', $id)>getFirstItem();
    $returnedTag = $firstItem>getGlsMessage();

    if($returnedTag === false || $returnedTag == "") {
        return false;
    }

    $tags = $this>parseIncomingTag($returnedTag);

    if(!is_array($tags)) {
        throw new \Exception($id . ' has no msg in gls box model.');
    }

    $service = $firstItem>getService();

    if ($service == "business" || $service == "cash") {
        $glsService = Mage::getModel('glsbox/label_gls_business');
    }
    elseif ($service == "express") {
        $glsService = Mage::getModel('glsbox/label_gls_express');
    }

    if($glsService != null) {
        $glsService>importValues($tags);
        return $glsService>getData();
    } else {
        return false;
    }
}
```

- **Wechselt euch alle zehn Minuten ab!** Jeder muss im Wechsel an die Tastatur. Denn nur wenn jeder mitmacht und seine Rolle übernimmt, wächst das Team zusammen, und es werden Ziele erreicht. Es darf also nicht unangenehm sein, Code zu schreiben, sondern ihr müsst es können und wollen. Der Schreiber sollte am besten auf Zuruf der anderen arbeiten. Denken und schreiben muss man in einer solchen Situation erst lernen.

- **Schreibt mehr Tests!** Es gibt zu wenig Tests in PHP-Projekten, und das ist ein großes Problem. Wenn man aber erst einmal ein paar geschrieben hat und sie locker von der Hand gehen, sieht man die Vorteile und will nicht mehr ohne arbeiten. Dass sie nicht geschrieben werden, liegt oft an fehlenden Skills in diesem Bereich. Macht das daher im Team und entwickelt euch auf diese Weise weiter! Ihr müsst dazu nur den Ausredenkatalog zuklappen und einfach anfangen.

Code Refactoring im ganzen Team

Der Berg der technischen Schuld ist in PHP-Projekten riesig und wächst täglich. Das hat leider mehrere Ursachen. So wird der schwarze Peter immer schön im Kreis herumgereicht und am Ende dem Kunden zugespielt, weil dieser dafür nicht bezahlen will. Aber das ist ein anderes Thema.

Wir dagegen sind jetzt so weit, dass wir im Team mit einer Funktastatur vor dem bestehenden Projekt sitzen und loslegen wollen (Listing 1).

Hier ist eine Methode purer Schmerz: tiefe Verschachtelungen, Early Exits, Mixed Returns, Duplicate Code und auch sonst alles, was das Herz begehrte. Nicht.

Das ist ungefähr so wie in meinem Garten: beides „historisch gewachsen“, wie man auch in der Entwicklung so schön sagt. Ein unerfahrener Gärtner würde in Versuchung geraten, sich sofort an die Arbeit zu stürzen und wahllos Pflanzen herauszureißen. Full stop! Das ist hier

nicht die Lösung. Richtig wäre es, einen Container zu bestellen, Arbeitsgerät und Kleidung zu besorgen. Einen Kasten Wasser daneben zu stellen und ein paar Stunden richtig zu arbeiten. Anpacken und Dinge bewegen.

Beim Refactoring im Team ist das Problem ähnlich gelagert: Es geht nicht richtig nach vorne, und die Arbeit erstickt im Detail. Besser ist: Test schreiben, Komplexität entfernen, den Code weiter nach links bringen. Dann hat man einen wesentlichen Fortschritt, und die Arbeit wird anerkannt.

Das Beispiel in Listing 2 wird von Unit-Tests unterstützt – so kann sich in einer Session ein Team eine Stunde darauf konzentrieren. Danach kommt ungefähr ein solches Ergebnis raus. Ja, es ist lesbarer und logischer, aber nach einer Stunde mit dem ganzen Team doch ein unbefriedigendes Ergebnis. Neben den starken Kommunikationsproblemen kommen dabei meist auch echte Wissenslücken zum Vorschein. Das kann schnell unangenehm werden. Alles in allem wird man mit dieser Taktik keinen Schritt weiterkommen und hat sogar viel Zeit verschwendet. Die Einführung von Pair Programming scheitert, wenn man mit Codereviews anfängt.

Clean-Code ist eine Teamaufgabe

Wir haben in einem kleinen ein- bis zweiköpfigen Special-Force-Team folgende Dinge getan: Front- und Backend-Tests geschrieben, einen Codestandard eingeführt, jeglichen auskommentierten und toten Code entfernt. Das ist schon viel besser. Aber ehrlich gesagt hat es die Softwarequalität im Sinne von Refactoring noch nicht wesentlich erhöht.

Zwei wichtige Schritte bei der Codeoptimierung sind es, die Komplexität zu verringern und die Methoden zu verbessern. So wird der Code besser testbar. Des Weiteren muss die Zahl der *if*-Statements drastisch reduziert werden, damit der Code lesbarer wird und effektiver ist. Hier muss man sich mit Clean-Code auseinandersetzt haben – und zwar mit den Regeln und Design Patterns. Das trifft auf Teams einfach nicht zu. Auch das ist eine Aufgabe für eine kleine Zwei-Mann-Einheit, also zwei Personen in Vollzeit. Eine schwierige wirtschaftliche Entscheidung.

Neue Features bringen den Erfolg

Da die Codequalität beim Pair Programming an neuen Features stark steigt, ist das eine sehr gute Möglichkeit, Pair Programming effektiv und wirtschaftlich einzuführen. Bei der gemeinsamen Arbeit findet ein umfangreicher Wissensaustausch statt, der alle weiterbringt und weiterbildet. Und statt Kritik an alten Fehlern kommen viele Ideen für neuen Code zusammen. Zudem können neue Codestandards und Coderegeln vereinbart und Tests zeitnah erstellt werden.

Insgesamt nimmt bei Features mit Pair Programming die Anzahl der Bugs um mehr als 50 Prozent ab. Hinzu kommt die hohe Lesbarkeit für alle Teilnehmer: Man fängt an, eine gemeinsame Sprache zu sprechen, und jeder Entwickler kennt sich später viel besser mit dem neuen Code aus – auch das steigert die Effektivität.

Über Ziele und Nebenwirkungen

Pair Programming kann unterschiedliche Ziele verfolgen. Auf der einen Seite geht es immer um die Verbesserung der Softwarequalität, und auf der anderen Seite soll der Know-how-Transfer unter den Entwicklern gefördert und intensiviert werden.

Daneben gibt es aber noch eine ganze Reihe angenehmer Nebeneffekte, die im Gesamtbild einen großen Teil des Mehrwerts beim Pair Programming ausmachen. So ist etwa immer wieder eine hohe bzw. erhöhte Motivation zu beobachten, wenn die Tastatur alle zehn Minuten zum nächsten Programmierer geht und der übernächste beschreibt, was zu programmieren ist. Außerdem gibt es in PHPEntwicklerteams strenge Hierarchien und häufig auch Inselwissen. Letzteres infolge unterschiedlicher Zuständigkeiten für verschiedene Bereiche in Applikationen wie Payment, Affiliate oder Elasticsearch. Pair Programming hingegen bricht das Gefüge auf und führt Teams zusammen, indem sich Entwickler gegenseitig bei Aufgaben unterstützen, mit denen sie sonst auf sich alleine gestellt wären. Das vermindert den Druck.

Ebenfalls ganz wichtig: Pair Programming ermöglicht nicht nur eine bessere Vertretung im Urlaubs oder Krankheitsfall, sondern erleichtert auch die Integration neuer Entwickler in bestehende Teams.

Pair Programming: Ein positives Fazit

Es sprechen viele wirtschaftliche und personelle Gründe für die Einführung von Pair Programming. Doch was zuvor jahrelang kaputtgemacht wurde, lässt sich natürlich nicht im Handumdrehen wieder in Ordnung bringen. Sprich: Wege werden nicht kürzer, nur weil man sie gemeinsam geht. Aber es macht deutlich mehr Spaß.

Heute werden immer mehr Technologien auf einer Seite eingesetzt, sodass neben den Rollen im Team jeder Entwickler auch seine persönlichen technischen Schwerpunkte hat. Gemeinsam kann hier das ganze Potenzial genutzt werden, um den Weg für Innovationen zu ebnen. Allerdings geht auch das nicht von jetzt auf gleich. Vielmehr erfordern Innovationen neben Standards auch immer Strukturen und ausreichend Zeit. Erste Schritte sind zwar sehr wichtig – jedoch ist der Weg zu den einzelnen Etappenzielen nie einfach und kurz. Dafür müssen alle Beteiligten an einem Strang ziehen und hinter der Sache stehen.

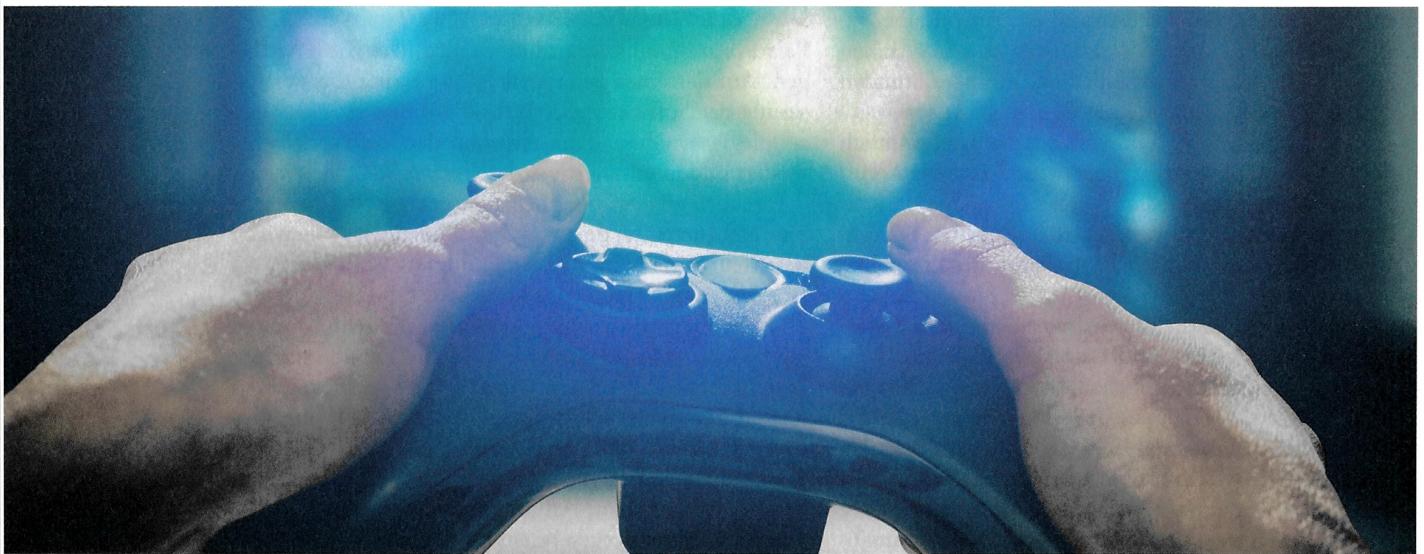
Professionelle Softwareentwicklung verfügt schon seit Langem über etablierte Methoden. Bei den Webworkers der Trial-and-Error-Generation sind diese oft aber noch nicht angekommen. Daher ist es zwingend nötig, die verschiedenen Generationen unter einen Hut zu bringen, und Pair Programming kann hier eine immens wichtige Säule darstellen. Allerdings reicht es nicht aus, ein paar Programmierer vor einen großen Monitor zu setzen und einzuschließen.



Roland Golla ist Gründer der Initiative „Never Code Alone“. Diese setzt sich für bessere Softwarequalität in Webprojekten und kostenlose Schulungen für Webworker ein.



www.nevercodealone.de



REDFPIXEL.PL/Shutterstock.com

Sauberer Code? Gezwungenermaßen

Force Feedback Programming

Mit Force Feedback Programming, einer Erweiterung für Visual Studio, soll unsauberer Code sichtbar gemacht werden. Die Erweiterung geht aber noch einen Schritt weiter: Sie zwingt Entwickler, sauberen Code zu schreiben. Force Feedback Programming nutzt die Microsoft .NET Compiler Platform für die Analyse des Codes.

von Robin Sedlaczek

Wir alle wissen, dass Feedback wichtig ist. Ohne Feedback würden wir Menschen nicht wissen, ob wir Dinge richtig machen oder nicht. Fehler und Probleme würden ohne Feedback (meistens) nicht sichtbar werden. Feedback hilft uns, die Dinge, die wir täglich tun, erzeugen oder auf irgendeine Art und Weise herstellen, zu verbessern. Viel wichtiger: Feedback hilft uns, uns selbst stetig zu verbessern. Und je früher Feedback gegeben wird, umso effizienter ist es, umso mehr Vorteile ergeben sich für uns als Menschen.

Hintergrund

Stellen wir uns vor, wir möchten eine Fremdsprache lernen. Wir investieren eine Menge Stunden, Tage, Wochen, um uns selbstständig Vokabeln und Grammatik beizubringen. Würden wir nicht frühzeitig bzw. sofort damit beginnen, die Sprache durch praktische Anwendung zu erlernen und zu trainieren, würden wir den Großteil der erlernten Vokabeln und grammatischen Regeln sehr schnell vergessen – zumindest die meisten von uns. Durch das Sprechen mit anderen Menschen erhalten wir sofortiges Feedback: Wir können sehen, ob sie uns verstehen, oder eben auch nicht. Nicht zuletzt am

Direktes Feedback ist wichtig, um Probleme und Fehler frühzeitig zu erkennen, um effizient zu arbeiten und die eigene Arbeit und sich selbst ständig verbessern zu können.

Gesichtsausdruck des Gegenübers, wenn ein entgleisendes Gesicht oder plötzliches Stirnrunzeln unsere Aussagen quittiert. Und genau das ist direktes Feedback! Wir bekommen es sofort, reagieren sofort darauf und lernen damit unmittelbar – ohne Zeitverlust. (Direktes) Feedback ist also eine gute Sache!

Historisches

1982 postulierte Ben Shneiderman [1] das Paradigma der direkten Manipulation für das Design von Benutzerschnittstellen [2]. Zuvor beruhte die Interaktion zwischen Mensch und Maschine weitestgehend auf einer Metapher, z. B. auf einem Kommandozeileninterface. Dieses ist relativ indirekt und abstrakt: Nach der Eingabe eines Kommandos (dessen Syntax dem Benutzer bekannt sein muss) muss dieses mit der Eingabetaste bestätigt werden. Es wird dann ausgeführt, und erst etwas später gibt es Feedback. Im schlimmsten Fall wurde das Kommando aufgrund falscher Syntax nicht verstanden und somit nicht ausgeführt.

1978 wurde TeX veröffentlicht [3]. TeX ist ein typografisches System, mit dem reichhaltige und hoch qualitative Textdokumente erzeugt werden können. Dazu bietet TeX ein umfangreiches Set an Befehlen an, mit denen Text arrangiert und formatiert werden kann. Diese Befehle müssen textuell eingegeben und in das Dokument integriert werden. Um das finale Ergebnis zu sehen, muss ein Prozessor bemüht werden, der die Befehle/Kommandos interpretiert und anschließend die Ausgabe generiert. Feedback kommt also relativ spät im Erstellungsprozess.

An dieser Stelle kommt die direkte Manipulation ins Spiel. Textverarbeitung, Tabellenkalkulation und die Desktopmetapher waren Kernpunkte des Vorschlags von Ben Shneiderman. Bei der direkten Manipulation werden Dokumente in ihrer finalen Form angezeigt. Der Benutzer arbeitet direkt auf den dargestellten Objekten, manipuliert deren Eigenschaften und sieht sofort die Auswirkungen der Manipulationen. Somit gibt es sofortiges Feedback auf Benutzeraktionen. Stellen wir uns moderne Textverarbeitungsprogramme vor. Text kann auf ganz einfache Art und Weise markiert, Schriftart, Schriftgröße, Schriftfarbe etc. können auf Knopfdruck geändert werden. Bilder können im Text direkt platziert, verschoben, rotiert und skaliert werden. Dazu genügen ein paar Klicks mit der Maus oder Gesten mit dem Finger auf mobilen Geräten. Direkte Manipulation

lässt uns also sehr schnell unsere Arbeit anpassen und verbessern. Ganz nebenbei lernen wir zudem, wie das System funktioniert. Fast auf spielerischem Wege. Und wieder: Direktes Feedback ist eine gute Sache.

Code

Betrachten wir das Schreiben von Code, insbesondere das Schreiben von sauberem Code. Was haben wir hier bezüglich direkter Manipulation und direktem Feedback? Natürlich bieten uns moderne IDEs eine Vielzahl smarter Tools, wie z. B. Syntax-Highlighting, automatische Formatierung, Auto vervollständigung und IntelliSense. Dies aber auch nur mehr oder weniger, je nach Grad der Typisierung einer Sprache. Unit-Tests validieren den Code gegen erwartetes und unerwartetes Verhalten. Diese Tests müssen aber zuerst selbstständig geschrieben und dann immer und immer wieder ausgeführt werden. Feedback kommt hier auch erst wieder relativ spät, also nicht direkt beim Codieren.

Tools wie NCrunch [4] geben uns direktes Feedback zur Testabdeckung. NDepend [5] gibt Feedback zu einer Menge anderer Codemetriken. Dennoch ist dieses Feedback mehr oder weniger indirekt, denn NDepend bringt zusätzliches Tooling ins Spiel, dessen Ergebnisse manuell evaluiert und analysiert werden müssen. Ob und wie wir als Entwickler darauf reagieren, bleibt uns überlassen.

Zurück zu sauberem Code, Clean Code: Was gibt es hier? Meckert unsere IDE mit uns, wenn wir z. B. zu viele Zeilen Code in einer Methode haben? Weiß die IDE, ab wann eine Methode zu lang ist? Denn dies ist ja im Wesentlichen abhängig von der Art des Projekts, dem Team und/oder den Vorgaben durch den oder die Projektbetreiber. Sicherlich ist die Länge einer Methode nicht alleinige Metrik, um festzustellen, ob Code sauber oder unsauber ist. Nach Prof. Dag Sjøberg ist sie aber ein guter Indikator dafür [6].

Die Idee

An dieser Stelle kommt Force Feedback Programming ins Spiel. Force Feedback Programming ist eine Idee von Ralf Westphal, erstmals vorgestellt unter [7]. Die Idee hinter Force Feedback Programming ist es, dem Entwickler direktes Feedback über die Codequalität zu geben, während der Code getippt wird. Beginnend mit der Lines-of-Code-(LOC-)Metrik, werden Methoden basierend auf der Anzahl der Codezeilen farblich in der

```

0 references | 0 changes | 0 authors, 0 changes
public Guest SupremeGuest
{
    get
    {
        if (_supremeGuest == null)
        {
            var creditCard = new CreditCard
            {
                Id = 1,
                Number = "423400",
                ValidUntil = DateTime.Today.AddYears(1),
                VerificationNumber = 111
            };

            _supremeGuest = new PrivateGuest(1, "Mr.", "Supreme", creditCard);
        }

        return _supremeGuest;
    }

    set
    {
        var hasCreditCard = (value as BusinessGuest)?.CreditCards.Count == 0;

        if (!hasCreditCard)
            throw new InvalidOperationException("Supreme business guest has no credit card.");

        _supremeGuest = value;
    }
}

```

Abb. 1: Force Feedback Programming in Aktion

IDE markiert. Je mehr Zeilen Code eine Methode hat, umso kritischer fällt die Farbgebung aus. Das ist aber noch nicht alles, Force Feedback bedeutet, dass der Entwickler die Qualität des Codes spüren soll – nicht nur visuell,

sondern auch taktil. Um dies zu erreichen, werden automatisch unerwünschte Zeichen im Code an der aktuellen Cursorposition generiert, während der Entwickler versucht, unsauberen Code zu erweitern. Auf diese Art und Weise soll es dem Entwickler erschwert werden, schmutzigen Code noch schmutziger zu machen. Die LOC-Metrik ist hier ein erster Anfang. Viele weitere Metriken können bezüglich Clean Code ausgewertet und Feedback dazu gegeben werden.

Mit dieser Idee soll direktes Feedback zur Codequalität direkt zum Entwickler, in die IDE, gebracht werden. Initial zu Trainingszwecken gedacht, findet sich Force Feedback Programming bereits in produktiven und realen Projektumgebungen wieder, und ärgert dort die Entwickler ...

Listing 1

```
{
    "methodTooLongLimits": [
        {
            "lines": 6,
            "color": "#ffffe5",
            "transparency": 0.25
        },
        {
            "lines": 16,
            "color": "#fec4f",
            "transparency": 0.50,
            "noiseDistance": 8
        },
        {
            "lines": 26,
            "color": "#cc4c02",
            "transparency": 0.80,
            "noiseDistance": 6
        }
    ]
}
```

Listing 2

```

var syntaxRoot = await currentDocument.GetSyntaxRootAsync();

var tooLongCodeBlocks = syntaxRoot
    .DescendantNodes(node => true)
    .Where(node => node.IsSyntaxBlock()
        && ( node.Parent.IsMethod()
            || node.Parent.IsConstructor()
            || node.Parent.IsSetter()
            || node.Parent.IsGetter()))
    .Select(block => block as BlockSyntax);

```

Das Projekt

Gemeinsam mit dem Autor dieses Artikels wird Force Feedback Programming als Erweiterung für Visual Studio entwickelt und kann direkt über die Visual Studio Gallery heruntergeladen werden [8]. Das Projekt wird in einem Open-Source-Umfeld auf GitHub gepflegt [9]. Mitmachen ist ausdrücklich erwünscht. Entwicklungsprozess, Coding Conventions und Contributor Guidelines sind im GitHub Repository gut dokumentiert. Der Continuous-Integration-Prozess ist mittels AppVeyor umgesetzt. Kontakt zum Team kann über Issues auf GitHub oder direkt per Gitter-Chat [10] aufgenommen werden. Eine kurze Einführung in die Funktionsweise der Visual-Studio-Erweiterung gibt es unter [11].

Die Visual-Studio-Erweiterung

Doch wie schaut das Ganze nun aus? Abbildung 1 zeigt die Force-Feedback-Erweiterung für Visual Studio in Aktion. Nicht nur Methoden, auch die Implementierung von Gettern und Settern von Properties werden je nach Anzahl der Codezeilen farblich markiert. Je mehr Zeilen eine Implementierung umfasst, umso kritischer die farbliche Darstellung der Methode. Die Erweiterung passt die Darstellung während des Tippens an. So wird dem Entwickler sofort ersichtlich, wann eine Methode beginnt, zu lang zu werden. So viel zum visuellen Feedback.

Je mehr Zeilen Code eine Methode hat, umso kritischer fällt die Farbgebung aus.

Überschreitet die Länge einer Methode einen bestimmten Grenzwert, gibt es zusätzlich zum visuellen Feedback taktiles Feedback. Dazu werden automatisch störende Zeichen generiert, während der Entwickler tippt. Das soll die Eingabe von Code in zu langen Methoden erschweren. Und das tut es auch. Selbst das Löschen von Zeichen generiert unter Umständen wieder diese störenden Zeichen. Schnell kommt man davon ab, eine unsaubere Methode zu erweitern.

Farben und Grenzwerte für Zeilenanzahl, für das visuelle und das taktile Feedback können einfach über eine JSON-Konfigurationsdatei eingestellt werden. So mit kann das „Feedback“ auf die eigenen Bedürfnisse angepasst werden. Der Codeausschnitt in Listing 1 zeigt, wie eine solche JSON-Konfigurationsdatei aufgebaut ist.

Anzeige

PHPmagazin³

Jetzt abonnieren und **3 TOP-VORTEILE** sichern!



Alle Printausgaben
frei Haus erhalten



Im entwickler.kiosk **immer**
und überall online lesen –
am Desktop und mobil



Mit vergünstigtem Upgrade
auf **das gesamte Angebot**
im entwickler.kiosk
zugreifen

PHP-Magazin-Abonnement abschließen auf www.entwickler.de

Grenzwerte können frei und beliebig oft definiert werden. In Listing 1 gibt es davon drei an der Zahl. Festgelegt werden kann, ab wie vielen Codezeilen eine Methode mit welcher Farbe eingefärbt wird. Dies wird mit der Eigenschaft *lines* angegeben. Die Eigenschaft *color* definiert dabei die Farbe, die zur Markierung der Methode angewendet wird, wenn die Anzahl der Codezeilen den Wert in *lines* überschreitet. Mit *transparency* kann zudem eine Transparenz für die Farbe festgelegt werden. Das taktile Feedback wird über die Eigenschaft *noiseDistance* gesteuert. Dieser Wert legt fest, wie viele Zeichen getippt werden können, bevor ein störendes Zeichen erzeugt wird. Im obigen Beispiel: Ist eine Methode länger als sechzehn Zeilen, wird jedes neunte Zeichen automatisch generiert. Ab sechsundzwanzig Zeilen ist es schon jedes siebte Zeichen, das störend erzeugt wird. Mit einer entsprechenden Staffelung kann der Druck auf den Entwickler also gesteuert werden.

Die technische Umsetzung

Das Visual-Studio-Plug-in erweitert den Codeeditor für die farbliche Darstellung. Dafür werden so genannte Adorner verwendet. Das sind im Wesentlichen grafische Erweiterungen, die in den Texteditor gehängt werden können. Unter der Haube instrumentalisiert das Force Feedback Programming die Microsoft .NET Compiler Platform. Herangezogen wird die semantische Analyse

Listing 3

```
foreach (var codeBlock in codeBlocks)
{
    var linesOfCode = codeBlock
        .WithoutLeadingTrivia()
        .WithoutTrailingTrivia()
        .GetText()
        .Lines
        .Count;

    var correspondingLimitConfiguration = null as
    LongMethodLimitConfiguration;

    foreach (var limitConfiguration in ConfigurationManager.
        Configuration.MethodTooLongLimits.OrderBy(limit => limit.Lines))
    {
        if (linesOfCode < limitConfiguration.Lines)
            break;

        correspondingLimitConfiguration = limitConfiguration;
    }

    if (correspondingLimitConfiguration != null)
    {
        var occurrence = new LongCodeBlockOccurrence(codeBlock,
                                                       correspondingLimitConfiguration);
        _longCodeBlockOccurrences.Add(occurrence);
    }
}
```

des C#-Compilers, um Methoden zu finden, die zu lang sind. Dazu werden zuerst alle Codeblöcke in der aktuell angezeigten Codedatei gesammelt (Listing 2).

Anschließend wird für jeden gefundenen Codeblock berechnet, ob die Anzahl der Codezeilen einen in der JSON-Konfigurationsdatei definierten Grenzwert überschreitet (Listing 3).

Entsprechend gefundene Verstöße werden gesammelt, um für diese visuelle Darstellungen Adorner zu generieren. Diese Adorner werden dann an der gefundenen Codestelle in den Editor gehängt. Über die Textpositionen werden dazu entsprechende grafische Koordinaten berechnet.

Fazit

Direktes Feedback ist wichtig, um Probleme und Fehler frühzeitig zu erkennen, um effizient zu arbeiten und die eigene Arbeit und sich selbst ständig verbessern zu können. Mit Force Feedback Programming kommt direktes Feedback bezüglich Clean Code nun auch in Visual Studio. Die Erweiterung weist durch Visualisierung auf unsauberen Code hin und verhindert sogar, dass der Entwickler unsauberen Code schreibt.



Robin Sedlaczek hat mehr als fünfzehn Jahre Erfahrung als professioneller Softwareentwickler. Seit sieben Jahren tätig bei der Fairmas GmbH in Berlin, hat er das auf Finanzplanung tools spezialisierte Unternehmen mit aufgebaut. Als CTO ist er dort für den gesamten Entwicklungsbereich zuständig, beschäftigt sich mit der Produktentwicklung, Softwarearchitektur und -spezifikation, Technologieevaluierung, Prozess- und Teammanagement mit Ausrichtung auf die Microsoft .NET Platform. In seiner Freizeit organisiert er die .NET User Group Berlin-Brandenburg und vermittelt sein Wissen auf Konferenzen, in User Groups, Onlinekursen, Fachartikeln und in seinem Blog.

<http://robinsedlaczek.com/> @RobinSedlaczek
 robin.sedlaczek@live.de

Links & Literatur

- [1] https://en.wikipedia.org/wiki/Ben_Shneiderman
- [2] <http://web.archive.org/web/2012020811520/http://www.elearning-reviews.org/topics/human-computer-interaction/design-principles/1983-shneiderman-direct-manipulation/>
- [3] <https://en.wikipedia.org/wiki/TeX>
- [4] <http://www.ncrunch.net/>
- [5] <http://www.ndepend.com/>
- [6] <http://blog.ralfw.de/2014/06/klein-ist-okonomisch.html>
- [7] <http://ralfw.de/2016/06/force-feedback-programming-clean-code-leicht-gemacht/>
- [8] <https://marketplace.visualstudio.com/items?itemName=RobinSedlaczek.ForceFeedback>
- [9] <https://github.com/robinsedlaczek/ForceFeedbackProgramming>
- [10] <https://gitter.im/robinsedlaczek/ForceFeedbackProgramming>
- [11] <https://vimeo.com/171889390>

Qualitätsstandards als Communityaufgabe

Die Entwickler machen lassen

Was Qualitätsmaßnahmen anbelangt, herrscht bei Entwicklern meist eher Augenrollen als Begeisterung. Zu oft sind die Maßnahmen unpraktisch und stehen mehr im Weg als dass sie helfen. Ganz anders kann es aussehen, wenn die Mitarbeiter ihre Standards und Vor-gehensweisen in einer Software Craftsmanship Community selbst entwickeln.

von Andreas Christian Fischer

Wie entsteht qualitativ hochwertiger Code? Die Frage treibt notwendigerweise jedes Softwarehaus um. Schon immer. Allerdings ist die Frage aktuell vielleicht noch etwas intensiver im Fokus als in der Vergangenheit. Denn in einem zunehmend komplexen Umfeld hängen Qualität, Erweiterbarkeit, Wartbarkeit und damit auch die Wirtschaftlichkeit von softwarebasierten Produkten einerseits in hohem Maße von den technischen Fähigkeiten der einzelnen Entwickler, andererseits aber auch von der Bereitschaft aller, in einem Team Verantwortung zu übernehmen und gemeinsam an einem Ziel zu arbeiten, ab. Diese Gedanken finden sich in zahlreichen Entwicklungen und Initiativen der letzten Jahre wieder, angefangen von Extreme Programming über agile Softwareentwicklung bis hin zu Software Craftsmanship und der Clean-Code-Bewegung. Interessant dabei ist, dass all diese Entwicklungen nicht top-down von oben verordnet, sondern bottom-up von den Softwareentwicklern selbst vorangetrieben werden. Daraus sind Initiativen wie das Agile Manifest von 2001 oder das Software Craftsmanship Manifest von 2009 entstanden. Die dahinter stehende Grundhaltung ist, dass jeder Entwickler gute Software schreiben will. Man muss ihn nur lassen. Aber was passiert, wenn man ihn tatsächlich lässt? Die DATEV hat sich darauf eingelassen. Welche Dynamik daraus entstanden ist, verdeutlicht die Entstehungsgeschichte der Software Craftsmanship Community @DATEV und ihr Wirken hinsichtlich der Entwicklung eines übergreifenden Standards für Softwarequalität im Unternehmen.

Bei der DATEV in Nürnberg arbeiten knapp 2 000 Softwareentwickler stetig daran, die bestehende Soft-

warepalette zu verbessern und neue Produkte für ihre Mitglieder – Steuerberater, Rechtsanwälte und Wirtschaftsprüfer – sowie deren meist mittelständische Mandanten zu gestalten. Das Problem: Wie ist bei einer kontinuierlich wachsenden Entwicklergemeinschaft gewährleistet, dass alle mit ihren unterschiedlichen Stärken und Kenntnissen der einzelnen Programmiersprachen einheitliche Standards befolgen und die Vielzahl an Lösungen wirtschaftlich beherrschbar bleibt?

Nachdem an verschiedenen Stellen im Unternehmen Qualitätsinitiativen ins Leben gerufen wurden – mal mit mehr, mal mit weniger Erfolg – sollte 2012 eine einheitliche, für alle Entwickler gleichermaßen gültige Initiative gestartet werden. Die Querschnittsverantwortung für diese Qualitätsinitiative wurde mir übertragen. Nach über 20 Jahren Entwicklungstätigkeit in verschiedenen Bereichen des Unternehmens war mir das Thema Clean Code als ein wesentlicher Qualitätssicherungsfaktor sehr ans Herz gewachsen. Gemeint ist damit, dass in erster Linie Quellcodes, aber auch Dokumente, Konzepte, Regeln und Verfahren so geschrieben sind, dass sie in kurzer Zeit und mit wenig Aufwand intuitiv richtig verstanden werden. Es geht also darum, im Code selbst möglichst hohe Transparenz herzustellen, beispielsweise, indem Variablen sprechend benannt werden und eine stringente Struktur verwendet wird.

Von der Idee zur Community

Die Software Craftsmanship Community geht zurück auf die Initiative einiger weniger Personen bei der DATEV. Wie so häufig, war der Ursprung des späteren Erfolgs ein Misserfolg: Eine der ersten Initiativen in meiner neuen Verantwortung bestand in der Ausrichtung

eines Seminars für Clean Code und Softwarequalität. Allerdings geriet es zum Flop. Der extern geladene Seminarleiter konnte die Erwartungen der Teilnehmer hinsichtlich Themenauswahl und methodischen Ansätzen nicht erfüllen. Die Initiative drohte zu scheitern, bevor es überhaupt richtig losging.

Durch Zufall erfuhr Martin Heider, freiberuflich tätiger agiler Coach aus Nürnberg, von dem gescheiterten Seminar. Ein Entwicklerteam hatte ihn engagiert, um die Einführung agiler Entwicklungsmethoden als Berater zu begleiten. Dabei kam die Sprache auch auf dieses Seminar. Aus seiner Sicht drohte ein Schaden, der über ein gescheitertes Seminar weit hinaus reichte. Wenn, so seine Überlegung, Themen wie ein einheitliches Qualitätsverständnis und Clean Code durch ein solches Seminar zu Un-Themen würden, hätte das auch erhebliche negative Folgen für die weitere Entwicklung agiler Methoden bei DATEV. Er ergriff also die Initiative und suchte das Gespräch mit mir, um einen neuen Anlauf in die Wege zu leiten.

Heider war damals schon gut mit den Ideen der Software Craftsmanship vertraut. Ihren Ursprung haben sie im Software Craftsmanship Manifest von 2009, in dem handwerkliche Aspekte als ein wesentlicher Teil der Softwareentwicklung herausgestellt werden. Es geht dabei um die Übertragung von Werten, die man gemeinhin mit dem guten alten Handwerk in Verbindung bringen würde. Also beispielsweise die handwerkliche Präzision und den Anspruch, etwas von Dauer zu schaffen, auf die moderne Softwareentwicklung anzuwenden. Um dies leisten zu können, müssen die Entwickler ihre Werkzeuge beherrschen. Daraus leiten sich die vier Grundprinzipien der Bewegung ab: üben, lernen, lehren und sich kümmern. Jeder kann und soll diese vier Prinzipien in seiner täglichen Arbeit beherzigen, um sowohl die eigenen Fertigkeiten auszubauen als auch andere an ihnen teilhaben zu lassen. Codequality wird somit zu einem Anliegen der gesamten Entwicklergemeinschaft.

Aus dem Austausch mit Martin Heider ist der Gedanke entstanden, in der DATEV eine eigene Community für Software Craftsmanship zu gründen. Das erste Communitytreffen fand Anfang März 2013 statt. 37 Teilnehmer waren gekommen und diskutierten lebhaft über ihre Vorstellungen einer solchen Community und über die gewünschten Inhalte für einen neuen Seminaransatz. Aus den Diskussionen über die Seminarinhalte wurde im Jahr 2013 gemeinsam mit der Weiterbildungsabteilung ein neuer Kurs entwickelt, der bis heute Bestand hat. Unter dem Titel „Der verantwortungsvolle Softwareentwickler“ werden in drei Tagen die aktuellen Entwicklungen im Handwerk eines Softwareentwicklers vorgestellt und es wird ein Bewusstsein für die weichen Faktoren wie Verantwortung und Teamarbeit geschaffen. Über 400 Entwickler haben das Angebot mittlerweile wahrgenommen.

So konnte Heider nur kurze Zeit später auf dem zweiten Communitytreffen eine Reihe von Ansätzen und Methoden der Software Craftsmanship vorstellen. Auf

Wenn Themen wie Clean Code und Codequality zum Un-Thema werden, hat das erhebliche negative Folgen.

diese Weise fanden erstmals Begriffe wie Coding Dojo, Code Kata oder Coderetreat Eingang in den Wortschatz und vor allem in die Gedankenwelt der DATEV.

Die Einheit von Lehren und Lernen: Coding Dojos und Coderetreats

Zusammen mit Daniel Bögelein, der sich ebenfalls für das Konzept der Software Craftsmanship begeistert hatte, verfolgten wir die Ideen weiter. Getreu dem Motto „Change braucht Verrückte“ luden wir zum ersten Coding Dojo der DATEV ein. Ohne eine nähere Vorstellung davon zu haben, wie so etwas abläuft oder wie man das moderiert. Doch das Format begeisterte alle Teilnehmer sofort. Bei einem Coding Dojo finden sich mehrere Entwickler zusammen, um ein Code Kata, also eine vorgegebene Programmieraufgabe von überschaubarer Dimension, gemeinsam durchzuführen und so voneinander zu lernen. Zwei Entwickler arbeiten im Team an einem Rechner. Die restlichen Teilnehmer verfolgen das Geschehen über einen Beamer. In einem festgelegten Rhythmus rotiert ein Entwickler aus dem Programmierpaar heraus und ein anderer Teilnehmer nimmt seinen Platz ein. Die Aufgabe wird dabei ständig wiederholt, sodass sich die Lösung kontinuierlich weiterentwickelt. Dieses erste Dojo war so erfolgreich, dass sich schnell Nachahmer fanden und bald eine eigene Coding-Dojo-Moderatorencommunity ins Leben gerufen wurde.

In ihrem ersten Jahr ging die Software Craftsmanship Community den nächsten Schritt: Sie lud zum ersten Coderetreat der DATEV. Jutta Rößner, damals Hauptabteilungsleiterin und mittlerweile Mitglied der Geschäftsleitung, übernahm die Schirmherrschaft für die Veranstaltung. Gemäß den Vorgaben eines Coderetreats fand auch dieser erste DATEV-Event seiner Art an einem Samstag zwischen 8:30 und 17 Uhr statt, mit anschließender Gelegenheit für einen entspannten Ausklang. Bewusst wählte man als Veranstaltungsort eine kleinere Zweigstelle, fernab der perfektionistischen Organisationsmaschinerie der Hauptstandorte. 32 Teilnehmer konnten an diesem ersten Coderetreat teilnehmen und waren begeistert.

Bei Coderetreats geht es nicht darum, neue Aufgaben zu lösen, sondern für eine gleichbleibende Problemstellung immer wieder andere Lösungswege zu suchen. Die Aufgabe ist in der Regel „Conways Game of Life“. Jeweils zwei Programmierer haben 45 Minuten Zeit, gemeinsam ihren Weg zu finden. Am Ende jeder Session



Abb. 1: Auch beim weltweiten Lernevent Global Day of Coderetreat war die DATEV-Community dabei

wird der Code gelöscht. Es zählt nur der Weg, nicht das Ziel. Nach 45 Minuten soll nichts weiter übrigbleiben als ein Erkenntnisgewinn. Ohne weiteren Ballast kann man sich sodann darauf konzentrieren, das eigene Entwicklungs- und Problemlösungsvorgehen weiter zu optimieren, neue Techniken auszuprobieren und sich von anderen inspirieren zu lassen. Bevor die nächste Runde losgeht, wird in der gesamten Gruppe über die Erfahrungen der letzten 45 Minuten diskutiert. Was lief gut? Was war schlecht? Welchen Hindernissen und Problemen sind die Gruppen auf ihrem Lösungsweg begegnet? Mit den so gewonnenen Erkenntnissen wird dann die Session mit veränderten Bedingungen wiederholt. So kann es beispielsweise sein, dass keine Maus verwendet werden darf, um Tastatur-Shortcuts zu üben. Ein Verbot bestimmter Codekonstrukte kann zu neuen Denkansätzen anregen. Ein Schweigegebot erfordert ein intensiveres Nachdenken darüber, wie der Code selbst zum Sprechen gebracht wird, damit der Partner nahtlos weiterprogrammieren kann. Häufig wird auch ein Partnertausch angeordnet, damit Denkstrukturen aufgebrochen werden.

Die zunächst noch wahrnehmbaren Stimmen, die Coderetreats als „esoterischen Quatsch“ bezeichneten, wurden von der Begeisterung der Teilnehmer dieses ersten und auch der folgenden Retreats schnell marginalisiert. Viele begannen im Anschluss, sich aktiv an der Software Craftsmanship Community zu beteiligen. Es sprach sich herum, dass dieses Format eine enorme Bereicherung ist, und auch die nächsten Veranstaltungen waren regelmäßig überbucht. Um dennoch Chancengleichheit für alle zu wahren, wurden die Teilnehmer unter den Anmeldungen ausgelost. So folgten 2014 zwei Coderetreats, 2015 waren es vier, 2016 dann fünf. Und bei allen zeigten sich Geschäftsleitungsmitglieder vor Ort und bekundeten so ihre Wertschätzung für die Initiative und für das freiwillige Engagement der Teilnehmer. 2016 nahm DATEV auch erstmals als Ausstragungsort am Global Day of Coderetreat teil, einem weltweiten Lernevent, zu dem auch externe Gäste in den

DATEV IT-Campus geladen waren (Abb. 1). Die Impulse waren für alle Teilnehmer sehr inspirierend.

Codequalität im Guten wie im Schlechten

Im Jahr 2012 hatte sich die Agile Community gegründet, die sich der Verbreitung agiler Methoden wie Scrum verschrieben hatte. Die Zusammenarbeit der beiden Communitys war von Anfang an intensiv, und 2014 riefen beide gemeinsam zu einem hausinternen Entwickler-Award auf. Einzelpersonen und Teams durften sich mit ihren Projekten bewerben, bewertet wurden verschiedene Kategorien wie agiles Projektmanagement, agile Entwicklungstechniken oder Codequalität. Neben der Ehrung dieser erfolgreichen Projekte ging es vor allem darum, Aufmerksamkeit für die Anliegen der beiden Communitys und auch für die Communitys selbst zu schaffen.

Doch bald folgte die Überlegung, dass die Ehrung von erfolgreichen Projekten sicher sinnvoll und wichtig sei, dass aber der Blick auf gescheiterte oder schlecht ausgeführte Projekte viel lehrreicher sein könnte. So kam es 2015 zum Code Skunk Award. Jeder Teilnehmer durfte einen eigenen Programmiercode einreichen und die seiner Meinung nach missratenen Aspekte vorstellen. Die Zuschauer sollten im Gegenzug auf die ihrer Ansicht nach positiven Aspekte eingehen.

Beide Awards waren sehr erfolgreich und haben viel Aufmerksamkeit auf die Communitys und ihre Anliegen gelenkt. Allerdings wurden sie 2016 eingestellt, da der Wettbewerbsgedanke zu sehr im Vordergrund stand und damit der Communitygeist infrage gestellt wurde. Der Code Skunk Award lebt heute jedoch als Bad Code Slam weiter.

Verbindlichkeit in die Community geben

Mittlerweile waren die Aktivitäten der Community so umfangreich und vielfältig, dass die bislang lose Organisationsstruktur des Netzwerks an ihre Grenzen stieß. Neben der Organisation von größeren Events wie Coderetreats, Coding Dojos und Awards geht es auch viel um gegenseitige Beratung, Weiterbildung und Austausch unter Kollegen. 2016 fand daher ein Strategietreffen der Community statt. Ein Ergebnis war die Entscheidung, ein festes Organisationsteam zu installieren. Auch von Seiten der Führung wurde das Engagement der Community honoriert und für mich eine neue Jobbeschreibung entwickelt, die auf die Leitung dieses Organisationsteams abzielt.

Ein zweiter Punkt aus dem Strategietreffen betraf die Frage, ob und wie sich die Community verbindlich in die Erarbeitung von unternehmensweiten Qualitätsstandards für die Softwareentwicklung einbringen kann. In der ersten Jahreshälfte 2016 war SonarQube eingeführt worden, eine Plattform für die statische Analyse der technischen Codequalität. Getreu dem Grundgedanken der Community ist die Nutzung von SonarQube freiwillig: Es ist ein Werkzeug für Mitarbeiter, das jeder im Rahmen seines Projekts verwenden kann oder

auch nicht. Führungskräfte haben keinen Zugriff auf die Plattform, um jeglichem Zweifel bezüglich der Freiwilligkeit oder gar Vermutungen hinsichtlich versteckter Kontrollabsichten zu begegnen.

Die Plattform arbeitet mit einer Datenbank und Scannern, die für jede der gängigen Softwaresprachen wie C#, C++, Java, XML oder COBOL ein eigenes Regelwerk enthält. Wenn ein Mitarbeiter SonarQube im Rahmen seines Projekts verwendet, wird im Hintergrund der Code analysiert und mit dem Regelwerk abgeglichen. Bei Verstößen erhält der Nutzer eine Meldung mit Hinweisen, wie er diese beheben kann. Allerdings geht es bei der technischen Codequalität nicht zwangsläufig um richtig oder falsch, sondern häufig um ein einheitliches und nachvollziehbares Vorgehen im Sinne von Clean-Code-Prinzipien. Dementsprechend ist es notwendig, das Regelwerk an die Gegebenheiten im Unternehmen anzupassen. Voraussetzung dafür ist, dass ein Konsens über das gewünschte Vorgehen existiert oder hergestellt wird. Die Verantwortung dafür war mir übertragen worden: Neben dem Einrichten des Servers war ich für das Regelwerk zuständig. Nur war mir aus verschiedenen Gründen nicht wohl dabei: Ich habe einige der Programmiersprachen noch nie verwendet, außerdem konnte ich mir nicht vorstellen, dass ein zentral vorgegebenes Regelwerk wirklich in der Entwicklergemeinschaft Anerkennung finden würde. Daher setzte ich einen basisdemokratischen Prozess über die Software Craftsmanship Community in Gang, um das Regelwerk auf die DATEV anzupassen. Das Ziel war, einen Austausch über die Regeln anzustoßen und damit auch ein gemeinsames Verständnis von guter Codequalität zu schaffen.

Zum Kick-off-Treffen kamen rund 40 Kolleginnen und Kollegen, die sich nach einer kurzen Einführung rege und intensiv an der Diskussion beteiligten. Ziel der Kick-off-Veranstaltung war es zunächst, die Rahmenbedingungen für den Prozess abzustimmen. Angesichts des zu erwartenden Aufwands, pro Sprache etwa 200 bis 400 Regeln hinsichtlich ihrer Relevanz und Anwendbarkeit auf die DATEV-Welt zu prüfen, musste ein möglichst effizientes Verfahren etabliert werden. Außerdem sollte größtmögliche Transparenz und Offenheit geschaffen werden, um die Akzeptanz für SonarQube und das Regelwerk bei den Entwicklern zu steigern.

So einigten wir uns darauf, zunächst alle Regeln zu aktivieren. Sobald ein Entwickler über einen Verstoß gegen eine Regel stolpert, mit der er nicht einverstanden ist, kann er diese Regel zur Diskussion stellen. Wenn ein entsprechender Hinweis eingeht, wird ein zeitnäher Termin für eine Regelkonferenz angesetzt, um darüber zu entscheiden. Parallel wird eine Onlinediskussion zur jeweiligen Regel angelegt, um bereits im Vorfeld Gründe und Argumente zu sammeln, die auf der Regelkonferenz in die Entscheidung mit einfließen. Auf der Konferenz wird jeder Antrag inklusive Begründung vorgestellt. Es folgt sofort eine Abstimmung ohne weiteren Gedanken-

austausch. Bei Stimmabgabe ohne Gegenstimme ist der Antrag entschieden, ansonsten folgt eine fünfminütige Diskussion mit erneuter Abstimmung. Bei dieser gilt dann das einfache Mehrheitsprinzip. Sowohl die Onlinediskussionen als auch die Regelkonferenzen sind für alle Entwickler offen. Jeder ist zum Mitmachen eingeladen. Auf einer Konferenz werden maximal 20 Regeln bearbeitet. Danach gilt für diese eine zweimonatige Quarantäne, bis sie erneut zur Diskussion gestellt werden können.

Innerhalb des ersten Monats wurden über 80 Regeln von der Community auf diesem Weg bearbeitet. An den Diskussionen und Regelkonferenzen haben sich etwa 90 Mitarbeiter beteiligt, die Resonanz auf die Aktion ist sehr positiv. Die Nutzungszahlen von SonarQube zeugen davon, dass die Regeln innerhalb der Entwicklergemeinde der DATEV eine hohe Akzeptanz finden.

Hohe Akzeptanz durch Mitmachprinzip

Die Entwicklung der Software Craftsmanship Community der DATEV ist ein anschauliches Beispiel dafür, welche positive Dynamik entstehen kann, wenn Mitarbeiter einfach mal machen dürfen. Der SonarQube-Prozess steht zwar erst am Anfang, aber die ersten Schritte zeugen von einer hohen Identifikation mit den dort getroffenen Entscheidungen. Gerade weil jeder die Gelegenheit hat, sich in den Prozess einzubringen, sind die Akzeptanz und die Bereitschaft zur Mitarbeit hoch.

Die DATEV hat so positive Erfahrungen damit gemacht, Communities zuzulassen und ihnen Verantwortung zu übertragen, dass mittlerweile allein in der Softwareentwicklung über 20 solcher Netzwerke bestehen. Sie organisieren gegenseitige Beratung, Austausch und auch mal den Blick über den Tellerrand. In den Communities wird der Transfer von Wissen auch außerhalb von organisierten Schulungen betrieben. Jenseits etablierter Bereichsgrenzen entstehen so auf unbürokratische Weise Netzwerke, die auch in der täglichen Arbeit viele Prozesse vereinfachen. Dabei geschieht alles, was in den Communitys passiert, auf freiwilliger Basis: Wer mitmacht, der macht es aus Überzeugung und weil er es für richtig hält. Sicher, es dauert länger, bis sich die Ideen durchsetzen. Aber dafür ist die Akzeptanz in der Belegschaft für den angestrebten Wandel enorm hoch. Die Mitarbeiter stehen voll dahinter, eben weil sie selbst an den Entscheidungen beteiligt und von ihnen überzeugt sind. Und der Spaßfaktor kommt dabei auch nicht zu kurz.



Andreas Christian Fischer ist seit dreißig Jahren Softwareentwickler mit Leidenschaft. Schon früh erkannte er die Bedeutung guten les- und damit wartbaren Sourcecodes. Er ist Gründer der DATEV-internen Software Craftsmanship Community, die möglichst viele Entwickler im Unternehmen von der Notwendigkeit technischer Exzellenz und professioneller Einstellung zum Job überzeugen möchte.

Aktiver Wissenstransfer im Team ist notwendig

Crossfunktionale Teams

Wie gehen wir schlau mit dem vorhandenen Wissen im Team um? Stichworte gibt es viele dazu: Crossfunktionalität, Interdisziplinarität, Generalisten, Spezialisten oder ein T-Shaped-Skill-Set. Diese Stichworte lassen sich auf folgende Fragen zurückführen: Welche Fähigkeiten benötigen wir, und bekommen wir das Wissen sinnvoll verteilt?

von Julia Schmidt



„Agile Entwicklungsteams sind interdisziplinär. Sie haben als Team alle Fähigkeiten, die notwendig sind, um ein Produktinkrement zu erstellen“ – im Scrum-Guide schreiben Ken Schwaber und Jeff Sutherland [1] über interdisziplinäre Teams. Wenn ein Team, ein Projekt-inkrement oder gar ein gesamtes Projekt erfolgreich umsetzen soll, muss das für die Umsetzung erforderliche Wissen auch im Team vorhanden sein. Ein Team zeichnet sich vor allem dadurch aus, dass alle Beteiligten auf das gleiche Ziel hinarbeiten: das gemeinsame Teamziel. Alle anderen, möglicherweise konkurrierende Ziele, werden diesem untergeordnet. Alle kennen die Stärken und Schwächen der Beteiligten – sowohl was die Fähigkeiten als auch was das Verhalten anbetrifft – und können damit zielführend umgehen. Ein Team, das nicht über das erforderliche Wissen zur Umsetzung eines Produktinkrements verfügt, das in Form einer User Story beschrieben ist, wird früher oder später scheitern. Dabei meint Scheitern nicht nur das endgültige Scheitern im Sinne von „nicht fertig werden“, sondern auch den Verlust von Geschwindigkeit und Innovation. Welche Kompetenzen im Team benötigt werden, hängt auch von der „Definition of Ready“ (DoR) ab. Damit definiert das Team, welche Aufgaben bereits erfüllt sein müssen, bevor die Aufgabe im Team bearbeitet werden kann.

Dabei sind Anforderungen an ein Produktinkrement nicht mit Vorgaben gleichzusetzen. In der Gesamtstruktur kann ein Produkt- oder Projektteam nur dann funktionieren, wenn die vorhandenen Skills im Team an die Feinheit der Anforderungen angepasst werden. Ein Projektteam ohne Designkenntnisse wird es schwer haben, Anforderungen ohne klar definierte Designvorlage umzusetzen.

Gleichzeitig wird ein Team ohne CSS-Kenntnisse nicht dazu in der Lage sein, eine solche Designvorlage auch umzusetzen. Das Team ist in der Selbstorganisation eingeschränkt – es ist auf Zulieferungen angewiesen. Solche Zulieferungen spiegeln sich auch in der Definition of Done (DoD) wider. Ist ein Inkrement fertig, wenn es auf der Testumgebung bereitsteht oder dann, wenn es tatsächlich live verfügbar ist?

Die agile Idee setzt auf kleine, abgeschlossene Produktinkemente, die in sich fertig und damit „releaseable“ sind. Dieses Vorgehen ermöglicht dem Team das sofortige Deployment einzelner User Storys. Das Team wird aus sich heraus, also selbstorganisiert, liefern.

Häufig wird der Deployment-Prozess aus dem eigentlichen Projektteam ausgeklammert. Die Selbstorganisation des Projektteams endet dann vor dem eigentlichen Release. Das Deployment findet erst am Ende eines Sprints oder sogar in individuell festgelegten Abständen durch Nichtmitglieder des Projektteams statt. Dieses Vorgehen hat direkte Auswirkungen auf die Lead-Time, also auf die Durchlaufzeit eines Inkrements von der Idee bis zum Release. Je mehr Stationen ein Inkrement durchlaufen muss, desto länger wird auch der tatsächliche Bearbeitungszeitraum. Wenn das eigentliche Veröffentlichen nicht durch das Projektteam erfolgt, mag

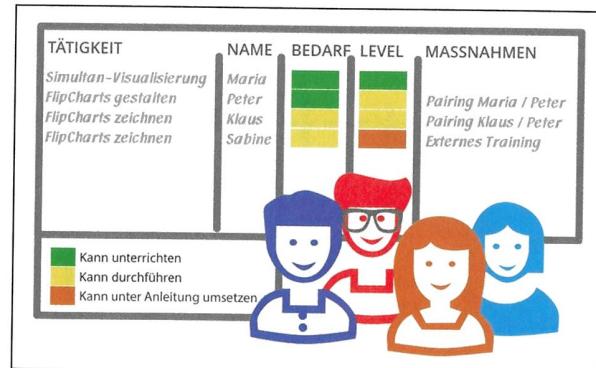


Abb. 1: Die Team-Skill-Matrix wird für das Projekt erfasst und öffentlich aufgehängt

das Team an Geschwindigkeit gewinnen, diese geht aber beim Warten auf das Release direkt wieder verloren.

Ist ein Team auf Zulieferungen eines Einzelnen oder einer anderen Abteilung angewiesen, arbeiten die Zulieferer an konkurrierenden Zielen. Dies wird die Geschwindigkeit des Projektteams ebenfalls beeinflussen. Auch hier ist mit Wartezeit zu rechnen. Ein Team, das von der Anforderung bis zur Auslieferung selbstgesteuert an Inkrementen arbeiten kann, wird schneller ausliefern. Damit ergibt sich eine kurze Durchlaufzeit für Produkte oder Produktverbesserungen. Das ist in sehr beweglichen Märkten ein erstrebenswerter Vorteil.

„Es sind alle im Team, die wir brauchen!“

Ein agiles Team hat alle Disziplinen im Team, die es braucht, um das Projekt oder Produkt zu realisieren. Zu Beginn mögen dabei nur Spezialkräfte im Team sein, die genau ihre Disziplin ausfüllen können. Aus Auslastungsgründen ist es allerdings mehr als sinnvoll, dass Tätigkeiten im Team möglichst von allen übernommen werden können. Wenn das nicht der Fall ist, verschiebt sich das Warten auf eine Disziplin in die agile Teamarbeit.

Ein Projektteam sollte deshalb vor allem eines sein: crossfunktional. In der Crossfunktionalität gehen wir von einem T-Shaped-Skill-Set aus. Das bedeutet, jedes Teammitglied hat seinen oder ihren Spezialbereich, gleichzeitig aber möglichst ein breites Basiswissen, um auch andere Aufgaben übernehmen zu können.

Aufgaben bleiben weniger lang liegen, wenn mehrere Kandidaten für die Bearbeitung bereitstehen. Es ergibt also Sinn, Teammitglieder in neue Techniken, Vorgehensweisen oder Aufgaben einzuarbeiten. Das kostet zunächst Aufwand, der sich allerdings bezahlt macht. Denn es ergibt sich der Vorteil einer zügigen und kompetenten Bearbeitung von Anforderungen. Die Aufgaben innerhalb eines agilen Entwicklungszyklus sind priorisiert. Um den größtmöglichen Erfolg des Projekts sicherzustellen, ist die Abarbeitung der Aufgaben je nach Priorität unerlässlich. Das gelingt nur dann, wenn möglichst viele Teammitglieder in der Lage sind, alle Aufgaben zu bewältigen. Viele Teams scheuen den aktiven Wissenstransfer.

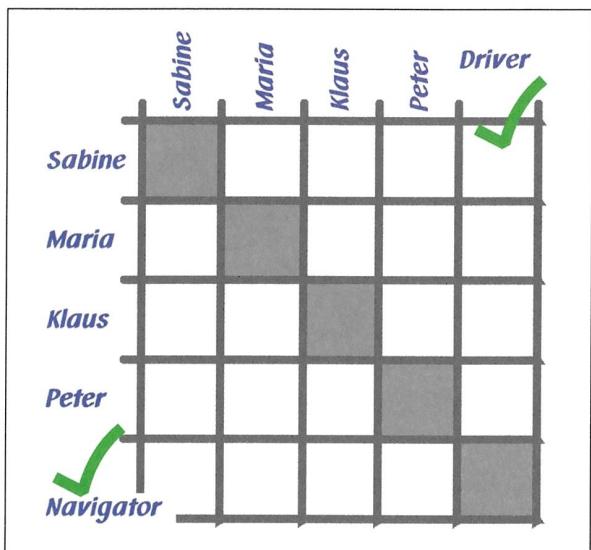


Abb. 2: Die Pairing-Matrix gibt Auskunft, welche Teammitglieder in welchen Rollen miteinander gearbeitet haben

Der vermeintlich einfachste Weg ist, im Team auszuloten, wer welches Spezialwissen mitbringt und die anstehenden Aufgaben dann entsprechend zu verteilen. Dieser Weg berücksichtigt allerdings nicht, dass Teammitglieder aufgrund von Urlauben oder gar Krankheiten ausfallen können. Ein solcher Ausfall gefährdet das Teamziel. Hinzu kommt die Wartezeit auf den Spezialisten. Inkremeante können eventuell nicht rechtzeitig fertiggestellt werden. Das Commitment des Teams wird nicht erfüllt, und am Ende werden Anforderungen nicht zeitgerecht umgesetzt.

Wir neigen dazu, den einfachsten Weg zu gehen. Aktiver Wissenstransfer ist eine bewusste Entscheidung, die Projektrisiken minimiert. Und für viele Teams gilt: „Unsere teaminterne Aufteilung in Disziplinen ist doch gut. Alle haben ihren klar definierten Spielbereich.“ Diese Haltung gefährdet die Wirtschaftlichkeit von Projekten. Verringern lassen sich diese Gefahren durch crossfunktionale Teams.

Start where you are

Startet ein Team neu, erscheint Crossfunktionalität meist wie eine unerreichbare Utopie. „Dafür haben wir keine Zeit“, „Warum sollte jemand die Aufgabe machen, der nicht das größte Wissen hat?“ oder „Niemand kann sich da sinnvoll einarbeiten. XYZ weiß das alles viel besser als der Rest“ sind häufig gehörte Argumente gegen aktiven Wissenstransfer im Team. Die Vorteile eines crossfunktionalen Teams werden erkannt, aber der Weg dahin erscheint zu steinig. Dabei lässt sich aktiver Wissenstransfer sehr leicht in das Daily Doing integrieren.

Rituale nutzen!

Ein „Wir reden da mal drüber“ macht noch keinen Wissenstransfer. Wissensaufbau und -austausch im Team braucht Struktur. Ein bewährtes Mittel ist die Team-Skill-Matrix. Mit der Team-Skill-Matrix setzt man sich

bewusst mit den notwendigen und vorhandenen Kompetenzen im Team auseinander.

Die zentralen Fragestellungen sind: Welche Skills brauchen wir im Team, um das Ziel zu erreichen? Welche Skills bringen wir bereits mit? Und wie sind diese konkret verteilt? Die Antworten auf diese Fragen lassen sich in einer Team-Skill-Matrix visualisieren (Abb. 1). Hierfür listet man zunächst alle identifizierten Skills und den benötigten Skill-Level auf. In die X-Achse setzt man alle Teammitglieder mit Namen und jeweils eine Spalte für Notizen und Maßnahmen. Bei der Skill-Betrachtung ist es empfehlenswert, zwischen drei Wissensständen zu unterscheiden:

1. Kann unterrichten.
2. Kann durchführen.
3. Kann unter Anleitung umsetzen.

Wurde der Wissenstransferbedarf identifiziert, folgt die Priorisierung. Nicht alle Skills und Wissenslücken sind gleich wichtig. Manche Kompetenzen werden erst zu einem bestimmten Zeitpunkt gebraucht, oder es gibt bereits mehrere Teammitglieder, die einen Bereich vorerst gut abdecken. In einem anderen steht allerdings nur eine Person zur Verfügung. Am besten man beginnt mit dem Wissenstransfer der Kompetenz, die für das Team den größten Hebel darstellt.

Um sich deutlich zu machen, wo dieser Hebel liegen kann, kann man die Team-Skill-Matrix auch um den Bedarf in einem perfekten Team ergänzen. Hierfür kann eine Spalte „Bedarf [Aufwand] in Stellen/Zeit/Stunden“ ergänzt und damit die real existierenden Kompetenzen im Team abgeglichen werden. Über diesen Vergleich sieht man sehr schnell, welche Kompetenzen als Erstes ausgebaut werden sollten [2].

Nun entwickelt man entsprechende Maßnahmen. Mögliche Maßnahmen können der gezielte Besuch von Fortbildungen sein. Aber auch Slack Time mit Themen-schwerpunkten ist als Maßnahme denkbar, um sich selbstständig in ein Thema einzulesen und sich anschließend mit einem anderen Teammitglied über das Thema auszutauschen.

Pair Programming? Ja – nach festen Regeln

Eine weitere Maßnahme für den gezielten Wissenstransfer ist Pair Programming. Erfolgreiches Pair Programming ist mehr als lediglich miteinander an einem Schreibtisch zu sitzen und zu coden. Für ein erfolgreiches Pair Programming sollte man im Vorfeld eine Pair-Programming-Matrix erstellen. Sofern man im Vorfeld eine Team-Skill-Matrix erarbeitet und damit eindeutig identifizierte Wissenstransferbereiche zur Verfügung hat, nutzt man diese Ergebnisse für die Pair-Programming-Matrix. Dabei werden alle Teammitglieder einmal in der X-Achse als „Driver“ und einmal in der Y-Achse als „Navigator“ aufgeführt. Das Ziel ist es, dass jedes Teammitglied einmal in einem bestimmten Zeitraum in jeder dieser beiden Rollen gearbeitet hat (Abb. 2).

Für ein erfolgreiches Pair Programming sollte zudem eine genaue Anzahl an Aufgaben oder Storys für einen konkreten Zeitraum festgelegt werden, die im Pairing bearbeitet werden sollen. Im Planning werden die ausgewählten User Storys entsprechend markiert.

Kommt eine solche Aufgabe nun zur Bearbeitung, legt man die beiden Bearbeiter fest. Eine Person als Driver, die andere als Navigator. Der Driver schreibt während der Umsetzung den Code und trifft Detailentscheidungen. Der Navigator gibt die Richtung vor, sagt also an, was als Nächstes zu tun wäre und erläutert das grundständische Design. Er oder sie kontrolliert den Code und sucht eventuelle Fehler. Nach einem festgelegten Zeitraum wechseln die Beteiligten ihre Rollen. Ein Wechsel kann zum Beispiel nach der Umsetzung einer Unteraufgabe, nach einem festen Zeitraum oder einer bestimmten Anzahl an Zeichen erfolgen.

Der Wechsel zwischen diesen Rollen ist zwingend erforderlich. Er fördert das Verständnis beider Seiten über die eigentliche Wissenshürde. Und beide Seiten können überprüfen, wie der Lernfortschritt vorangeht.

Pairing – auch über die Technik hinaus

Auch außerhalb von Entwicklungsteams ist Pairing ein bewährtes Mittel. So kann man für nahezu jeden Wissens- und Tätigkeitsbereich eine Matrix erstellen. Das Prinzip bleibt ähnlich: Derjenige, der Wissen vermittelt, bleibt zunächst aktiv, während der oder die Lernende als Zuschauer oder „Schatten“ begleitet. Zu gegebener Zeit tauschen die Beteiligten ihre Rollen. Diese Art der Wissensvermittlung führt zum Beispiel im Bereich Moderation zu sehr guten Ergebnissen. Wichtig ist allerdings, dass die Beteiligten nach allen Terminen ein direktes Feedbackgespräch vereinbaren, um Gesehenes zu besprechen.

Über das Team hinaus

Ein weiteres Ritual zum gezielten Wissenstransfer können interne Konferenzen oder gezielte Slack Time, also nicht verplante Zeit, sein. Insbesondere für den teamübergreifenden Wissenstransfer sind interne Konferenzen oder Community of Practices (CoP) ein gutes Mittel.

Eine CoP besteht üblicherweise aus Personen, die an ähnlichen Aufgaben arbeiten. Sie treffen sich in regel-

Pairing nicht nur als Pair Programming nutzen

Pairing funktioniert nicht nur als Pair Programming. Auch in anderen Aufgabenbereichen kann Pairing als Wissenstransferbaustein genutzt werden. Beispielsweise hier:

- Moderation von Teamritualen
- Erstellung von Basisdesigns für das Softwaresystem (Grafik plus Frontend)
- Formulierung von Anforderungen (Epics, User Storys)

mäßigen Abständen, um über Erfolge und Misserfolge, aber auch über neue Entwicklungen zu sprechen. Es hat sich außerdem bewährt, wenn die CoP für jeden Termin gezielt ein Thema und teilweise sogar einen Referenten festlegt. Wissen, das in einer CoP erlangt wurde, sollte mit einem ähnlichen Ritual wieder zurück ins eigene Entwicklungsteam gebracht werden, um den Wissensgewinn möglichst weit zu streuen.

Im Rahmen von internen Konferenzen haben Mitarbeiter Zeit, sich bewusst mit Themen auseinanderzusetzen. Die Themen können hierbei entweder vorgegeben oder frei gewählt werden. Wichtig für den Wissenstransfer ist es, ein Format zu finden, in dem in Teams gearbeitet wird.

Für eine interne Konferenz kann unter anderem ein Format ähnlich einer „Unconference“ genutzt werden. Bei einer Unconference benennen die Teilnehmer Themen, zu denen sie etwas beitragen möchten. Gleichzeitig dürfen sie aber auch Themen benennen, zu denen sie gerne etwas hören oder arbeiten möchten. Mit einem einfachen Punkte-Voting-Verfahren werden die tatsächlichen Themen ausgewählt. Alle Ergebnisse der Themen-Sessions werden dokumentiert und im Anschluss für jeden zur Verfügung gestellt.

Einfach mal machen

Wissenstransfer kostet Zeit. Zeit, die man im Projektalltag zunächst gefühlt nicht hat. Spätestens wenn eine User Story ins Stocken gerät, weil der Spezialist für die Aufgabe ausfällt, wird man merken: Langsam an einer Aufgabe zu arbeiten, ist immer noch besser, als gar nicht an ihr zu arbeiten.

Verabschiedet euch von dem Gedanken, innerhalb weniger Tage die perfekten T-Shaped-Skill-Sets aufzubauen. Die braucht man auch nicht. Man braucht ein Team, das zu jedem Zeitpunkt dazu in der Lage ist, die Anforderungen im aktuellen Entwicklungszyklus umzusetzen. Die Frage ist also nicht, wie man ein perfektes Skill-Set im Team erreicht, sondern zunächst, wie man den ersten Schritt in Richtung T-Shaped machen kann. Das Etablieren von Ritualen zum Wissenstransfer ist dieser Schritt.



Julia Schmidt ist Trainerin und Projektcoachin, die Teams bei der Einführung agilen Arbeitens unterstützt. Ihr Schwerpunkt liegt auf agilen Projektmethoden, Moderationsmethoden und Visualisierungstechniken. Julia bloggt regelmäßig über ihre Arbeit bei BERATUNG JUDITH ANDRESEN und bietet offene Workshops zu Visualisierungs- und Moderationstechniken an. Das Team der BERATUNG JUDITH ANDRESEN möchte echte Zusammenarbeit möglich machen.



www.judithandresen.com

Links & Literatur

[1] <http://www.scrumguides.org/>

[2] <https://management30.com/leadership-resource-hub/team-competency-matrix/>

Effektiver Einsatz von Codereviews

Vier Augen sehen mehr als zwei

Codereviews führen zu qualitativ hochwertigeren Systemen und zu besser vernetzten Teams. Denn das gemeinsame Durchgehen von Quelltexten bringt Fehler frühzeitig zum Vorschein und hilft beim Wissensaustausch. Die optimale Einbettung in den Entwicklungsprozess und eine Infrastruktur aus etablierten Tools sorgen zudem dafür, dass Codereviews auch langfristig zum Erfolg eines Unternehmens beitragen.

von Thorsten Maier

Kontinuierliche Codereviews sind ein effektives Mittel, um Softwarequalität langfristig zu steigern. Bei einem Codereview begutachten ein oder mehrere Entwickler den Quellcode, der von einem Kollegen geschrieben wurde. Regelmäßig und richtig durchgeführte Codereviews versprechen viele Vorteile. Denn je häufiger Quellcode betrachtet wird, desto mehr Fehler können darin gefunden werden. Und umso früher ein Fehler gefunden werden kann, desto günstiger lässt er sich beheben. Befindet sich ein Fehler erst einmal in einem Produktionssystem, kann die Behebung teuer werden. Daher ist es sinnvoll, möglichst viele Fehler bereits in einer frühen Entwicklungsphase zu finden [1].

Codereviews fördern auch die Wissensvermittlung. Ein häufiges Problem bei der Entwicklung von Software ist das Entstehen von Wissensinseln. Codereviews mildern diesen Effekt, da sich jeder Entwickler zwangsläufig mit anderen Codeteilen auseinandersetzen muss. Als positive Folge wird sich auch die Wartbarkeit der Software erhöhen. Code, der weniger Fehler enthält und über den mehr Entwickler ein tieferes Verständnis haben, wird über kurz oder lang eine höhere Qualität aufweisen und somit auch wartbarer sein. Nicht zu unterschätzen ist der letzte Vorteil von Codereviews: Beim Reviewen werden nicht nur Fehler, sondern auch viele gute Codestellen zum Vorschein kommen, die dem Reviewer als Anregung für weitere Entwicklungsarbeiten dienen können.

Trotz dieser vielen Vorteile werden Codereviews nicht in allen Entwicklungsprojekten gelebt. Viele Entwickler berichten sogar von negativen Erfahrungen mit Codereviews. Die Gründe hierfür sind vielfältig. Mit

der richtigen Vorbereitung lassen sich auftretende Anfangsschwierigkeiten allerdings in der Regel schnell überwinden. Zunächst sollten einige technische Voraussetzungen geschaffen werden, um das effiziente Durchführen von Codereviews überhaupt erst zu ermöglichen. Besonders wichtig ist es zudem, eine positive Grundhaltung bei allen Beteiligten zu schaffen, also bei den Entwicklern und dem Management. Nur wer sich ernsthaft und offen gemeinsam im Team verbessern möchte, kann die Vorteile von Codereviews auch nutzen. Nicht zuletzt scheitert die Einführung von Reviews allzu häufig am Fehlen konkreter Regeln und an einer nicht optimalen Einbindung in den Entwicklungsprozess.

Vor dem Codereview steht der Test

Ein wesentlicher Vorteil von Codereviews ist das Senken der Gesamtentwicklungskosten. Diese Kostenersparnis lässt sich allerdings nur erreichen, wenn die zur Verfügung stehende Zeit der beteiligten Personen optimal genutzt wird. In Softwareentwicklungsprojekten sind die Personalkosten in aller Regel der größte Kostenfaktor, und daher sollten wir die für Reviews eingesetzte Zeit eines jeden Entwicklers auf das notwendige Mindestmaß beschränken. Durch geschickte Automatisierung von Qualitätssicherungsmaßnahmen können wir den manuellen Reviewaufwand auf den tatsächlich notwendigen Teil beschränken. Die konkreten Maßnahmen unterscheiden sich zwar von Projekt zu Projekt, basieren aber letztlich immer auf einer Kombination der gleichen Mechanismen.

Beim Betrachten eines Codefragments lassen sich Bugs meist nur schwer erkennen. Noch schwieriger ist allerdings das Finden von unerwünschten Seiteneffekten, die sich durch geänderte Codestellen ergeben. Diese

Seiteneffekte lassen sich bei einem Review nur mit viel Erfahrung und einem tiefen Wissen über die Anwendung finden. Es ist daher wichtig, mit einer hohen Testabdeckung möglichst viele Seiteneffekte automatisiert zu finden, da sich dies nur bedingt im Rahmen eines Reviews leisten lässt.

Zu einer hohen Testabdeckung gehört zwingend auch ein automatischer Build-Prozess, der in der Lage ist, auch Tests automatisch auszuführen. Des Weiteren benötigt man einen Continuous-Integration-Server, der diesen Build-Prozess regelmäßig startet, sodass die Entwickler über fehlgeschlagene Tests informiert werden. Auch Codequalitytools wie SonarQube [2], FindBugs [3], PMD [4] oder Checkstyle [5] können beim automatischen Entdecken von Problemstellen im Code hilfreich sein, die man ansonsten manuell nur mit großem Aufwand finden würde.

Neben diesen zum Glück mittlerweile weit verbreiteten Qualitätsmaßnahmen lassen sich noch viele weitere automatisierte Prüfungen in den Build-Prozess integrieren. Dazu gehört vor allem das Überprüfen von Architekturvorgaben und Coding-Guidelines. So lassen sich mit dem statischen Codeanalysetool jQAssistant [6] Verletzungen in der Schichtenarchitektur der Anwendung oder aber auch bei den Namenskonventionen von Klassen-, Methoden- und Paketnamen feststellen. Die jQAssistant-Regel in Listing 1 prüft beispielsweise, dass Klassen, die mit der Annotation `@Controller` versehen sind, auch in einem Paket mit dem Namen `controller` liegen. Diese Regel wurde in Cypher geschrieben, der Abfragesprache der Graphdatenbank Neo4j.

Akzeptanz schaffen

Sind die technischen Voraussetzungen geschaffen, geht es daran, alle Beteiligten von der Nützlichkeit von Codereviews zu überzeugen. Es gilt dabei vor allem, zwei verschiedene Gruppen zu gewinnen: das Management, das über die Verteilung von Entwicklungsbudgets entscheidet, und die Entwickler, die die Reviews letztlich durchführen müssen. Die Durchführung von Codereviews kostet Geld, und daher müssen zunächst die Geldgeber von der Wirksamkeit überzeugt werden, da ansonsten kein Budget für diese Qualitätssicherungsmaßnahme zur Verfügung steht. Erfreulicherweise ist die Akzeptanz von langfristigen Qualitätsmaßnahmen bei den meisten Budgetverantwortlichen recht hoch. Zumindest auf einen leichtgewichtigen Versuch mit einem Teil des Entwicklungsteams sollten sich die meisten Verantwortlichen einlassen. Eine einfache Metrik kann bei einem solchen Versuch dabei helfen, die Wirksamkeit von Codereviews und somit den Einsparungseffekt aufzuzeigen. Das Erfassen einer solchen Metrik sollte dabei möglichst wenig Aufwand für die Entwickler bedeuten und dennoch die Gesamtkostensparnis sowie die Qualitätsverbesserungen sichtbar machen. Eine einfache, aber dennoch hilfreiche Metrik lässt sich beispielsweise durch das Erfassen der folgenden Daten für jedes durchgeführte Review erstellen:

- Codeautor
- Reviewer
- Zeitaufwand für die Entwicklung
- Zeitaufwand für das Review
- Anzahl der überprüften Codezeilen
- Anzahl der gefundenen Stellen mit Verbesserungspotenzial
- Anzahl der gefundenen Fehler

Aus diesen wenigen Informationen lassen sich bereits wichtige Kennzahlen erheben. So lässt sich damit das Verhältnis zwischen Entwicklungs- und Reviewaufwand ermitteln. In der Praxis sollte der Aufwand für das Review nicht mehr als 15 Prozent des Entwicklungsaufwands betragen. An einem durchschnittlichen Acht-Stunden-Arbeitstag entspricht dies vereinfacht gesagt sieben Stunden Entwicklung und einer Stunde Codereview. Ist der zeitliche Aufwand für die Reviews deutlich höher, sollten Gegenmaßnahmen ergriffen werden. So können die beschriebenen automatisierten Qualitätssicherungsmaßnahmen nach und nach erweitert werden, um damit den manuellen Aufwand zur Prüfung des Codes zu verringern. Alleine durch die mit vergleichsweise geringem Aufwand früh gefundenen Fehler lässt sich das Management erfahrungsgemäß relativ schnell vom langfristigen Nutzen von Reviews überzeugen.

Deutlich schwieriger ist es mitunter, alle Entwickler für diese Art der Qualitätssicherung zu begeistern. Den Willen, sich ständig weiterzuentwickeln und von anderen etwas zu lernen, haben leider bis heute nicht alle Entwickler verinnerlicht. Dabei sollte eigentlich jedem klar sein, dass man eine Verbesserung der Codequalität am besten gemeinsam erreichen kann. Um negative Assoziationen mit Codereviews zu vermeiden, muss man darauf achten, dass die Stimmung während eines Reviews möglichst positiv ist. Es geht nicht darum, die Arbeit eines Einzelnen zu kritisieren, sondern darum, gemeinsam ein Ziel zu erreichen. Konstruktive Kritik kann dabei natürlich nicht vollständig ausbleiben. Umso wichtiger ist es daher, immer wieder den positiven Grundton zu betonen. Beispielsweise kann man versuchen, jede kritische Anmerkung mit einer positiven Anmerkung zu verbinden: „Deine Umsetzung gefällt mir wirklich sehr gut, aber ich glaube an dieser Stelle können wir den Code noch ein wenig lesbarer machen.“ Werden Reviews erst einmal als etwas Positives gesehen, verbessert sich da-

Listing 1: jQAssistant-Regel

```
<constraint id="my-rules:ControllerInControllerPackages">
<description>Controllers must be in a controller package</description>
<cypher><![CDATA[
MATCH (c:Class)-[:ANNOTATED_BY]->(a:Annotation) MATCH (a)-[:OF_TYPE]->(at {name: 'Controller'}) MATCH (p:Package)-[:CONTAINS]->(c) WHERE p.name => 'controller' RETURN c,p LIMIT 10
]]></cypher>
</constraint>
```

durch automatisch die Akzeptanz. Sind die technischen Voraussetzungen geschaffen und alle Beteiligten von der positiven Wirkung von Codereviews überzeugt, geht es an die konkrete Umsetzung der Reviews.

Zu zweit, nicht zu viel und nicht zu schnell

Gerade in zeitlich und örtlich getrennten Entwicklungsteams lässt sich eine asynchrone Kommunikation zwischen den einzelnen Entwicklern nicht immer vermeiden. Aus verschiedenen Gründen ist es allerdings dennoch empfehlenswert, dass der Codeautor beim Codereview beteiligt wird und dem Reviewer unmittelbar für Rückfragen zur Verfügung steht. Das Review sollte daher, wenn möglich, immer zu zweit durchgeführt werden. So kann der Autor wichtige Hinweise geben, in welcher Reihenfolge die Codeänderungen begutachtet werden sollten, um die Struktur des Codes möglichst schnell zu durchdringen. Auch funktioniert die so wichtige Wissensvermittlung am besten, wenn sich zwei Entwickler gegenübersetzen und sich unmittelbar über den Code unterhalten können. Zudem lassen sich Meinungsverschiedenheiten persönlich meist am besten klären, und man erspart sich dadurch das zeitaufwändige und wiederholte Kommentieren von Codestellen in einem Codereviewtool. Besonders wenn die Stimmung im Team nicht besonders gut ist oder die positive Grundeinstellung gegenüber Codereviews verbessert werden muss, sollten Codereviews ausschließlich im Vier-Augen-Prinzip durchgeführt werden. In einem gut funktionierenden Team mit Entwicklern, das sich ohnehin offen für konstruktive Kritik zeigt, lässt sich eher auf dieses Prinzip verzichten.

Ebenfalls aufgrund der Wissensverteilung und des Team-Buildings sollten die Reviews nicht immer die gleichen Personen durchführen. Viel besser ist ein möglichst häufiger Wechsel der beiden Beteiligten eines Reviews. So sollten explizit auch etwas unerfahrenere Mitarbeiter den Code von sehr erfahrenen Mitarbeitern begutachten, da dabei ein maximaler Wissenstransfer stattfinden kann und dennoch der ein oder andere Fehler entdeckt wird. Um eine möglichst breite Streuung zu erreichen, kann die Auswahl eines Reviewers per Zufallsprinzip sinnvoll sein.

Codereviews erfordern vor allem beim Reviewer ein hohes Maß an Konzentration. Daher sollte man darauf achten, dass nicht zu viel Code in zu kurzer Zeit betrachtet wird. Eine ausführliche Studie zu diesem Thema [7] kommt zum Schluss, dass nicht mehr als 400 Zeilen Code pro Review überprüft werden sollten. Ansonsten sinkt die Anzahl der gefundenen Fehler pro Codezeile deutlich. Ein Review sollte nicht länger als eine Stunde dauern. Zudem sollte man auch darauf achten, die Codezeilen nicht zu schnell zu begutachten und die maximale Geschwindigkeit von 500 Zeilen pro Stunde nicht zu überschreiten.

Checkliste erstellen und Potenzial nutzen

Dem Autor und dem späteren Reviewer kann eine Checkliste mit den wichtigsten zu prüfenden Punkten zur Verfügung gestellt werden. Der Autor kann sich mithilfe dieser

Liste optimal auf das Review vorbereiten und bereits im Vorfeld Flüchtigkeitsfehler beseitigen. Dem Reviewer hingegen dient eine solche Checkliste als roter Faden für den Ablauf des Reviews. Beispielsweise kann eine solche Checkliste die folgenden Checkpunkte enthalten:

- Wurden sinnvolle Namen für Pakete, Klassen, Methoden und Variablen gewählt?
- Lässt sich eine klare Struktur im Code erkennen?
- Werden Datentypen korrekt verwendet?
- Sind alle notwendigen Kommentare vorhanden?
- Wurde auf eine ausreichende Testabdeckung geachtet?
- Sind klar definierte Schnittstellen vorhanden?
- Ist das Exception Handling korrekt umgesetzt?
- Wurden Texte internationalisiert?

Diese Checkliste muss ständig an sich ändernde Gegebenheiten im Projekt angepasst werden. Sie sollte zudem so konkret wie möglich sein, um Diskussionsbedarf auf ein Minimum zu reduzieren.

Bei einem Review werden nicht nur Bugs, sondern auch viele weitere Verbesserungsmöglichkeiten zum Vorschein kommen, die eher der Codequalität als einer Fehlerliste zuzuordnen sind. Diese sollten grundsätzlich, genau wie das Beheben der gefundenen Fehler, in Ruhe nach dem Review und nicht hektisch zwischen-durch umgesetzt werden. Sie bilden die Grundlage für die langfristige Verbesserung der Codebasis und somit für die erhöhte Wartbarkeit. Umso wichtiger ist es, sich hierfür genügend Zeit zu lassen, da sich die Ergebnisse der zeitaufwendigen Reviews ansonsten nur zu einem kleinen Teil nutzen lassen.

Wir sollten daher alle größeren Verbesserungsmöglichkeiten in unsere Aufgabenverwaltung aufnehmen, um somit eine unproblematische Planung der Aufgaben zu ermöglichen. Erst so kann fortlaufend eine Priorisierung zwischen neuen Features, bestehenden Bugs und den notwendigen Aufgaben zur Verbesserung der Codebasis stattfinden. Zielkonflikte bleiben durch eine solche Vorgehensweise natürlich nicht aus, aber zumindest werden sie bereits im Planungsprozess sichtbar und lassen sich somit managen.

Leichtgewichtig starten und Tools gut integrieren

Die effektive Durchführung von Codereviews lässt sich durch den Einsatz der richtigen Tools noch weiter verbessern. Grundsätzlich kann man allerdings die Empfehlung geben, dass man gerade am Anfang nicht zu viel Aufwand in die Auswahl eines passenden Reviewwerkzeugs stecken sollte. Codereviews lassen sich auch leichtgewichtig ohne größere Anpassung des bestehenden Entwicklungsprozesses und zusätzliche Tools durchführen. Mit der Versionsverwaltung und dem in jeder Entwicklungsumgebung ohnehin bereits integrierten Diff-Tool haben alle Entwickler bereits die wichtigsten Werkzeuge für das Durchführen eines Reviews installiert. Erst wenn sich die Qualitätssicherung per Codereview als natürlicher Teil der Entwicklungsarbeit

etabliert hat, ist es an der Zeit für die Auswahl eines geeigneten Tools. Dieses leichtgewichtige Vorgehen bewahrt davor, dass aus den Codereviews der ungeliebte schwergewichtige Prozess wird, der leider in zu vielen Unternehmen zu negativen Erfahrungen geführt hat.

Oberste Priorität bei der Auswahl eines Tools sollte eine gute Integration in den bestehenden Entwicklungsprozess und die bereits vorhandenen Entwicklungswerkzeuge haben. Ein Codereviewtool, das völlig losgelöst von der Aufgabenverwaltung oder der Versionsverwaltung eingesetzt werden soll, wird kaum auf große Gegenliebe bei den Entwicklern stoßen. Für die Auswahl ist es daher vor allem wichtig, den bestehenden Entwicklungsprozess zu kennen.

Werden zum Beispiel Feature-Banches für die Entwicklung eingesetzt, bietet sich der Einsatz von so genannten Pull Requests an. Mit diesen signalisiert ein Entwickler, dass die Änderungen auf einem Feature-Branch zurück in die Hauptentwicklungslinie überführt werden sollen, da die Arbeiten abgeschlossen sind. Aus Sicht des Codereviewprozesses haben Pull Requests den großen Vorteil, dass man einen definierten Zeitpunkt und eine definierte Codemenge für ein Review geschenkt bekommt. Vor der Reintegration werden alle Codeänderungen, die sich auf dem Feature-Branch gegenüber der Hauptentwicklungslinie ergeben haben, einer Prüfung durch ein Codereview unterzogen. Falls die Änderungen den Qualitätsanforderungen des Projekts entsprechen, erfolgt die Reintegration (Merge). Andernfalls wird der Code zunächst auf dem Feature-Branch stabilisiert, ohne dabei andere Entwickler auszubremsen. Dieser Pull-Request-basierte Workflow ist vor allem im Open-Source-, aber mehr und mehr auch im kommerziellen Umfeld zu finden. Als wohl bekanntester Vertreter nutzt GitHub diese Vorgehensweise. Wer lieber auf selbst gehostete Lösungen setzen möchte, findet beispielsweise mit Bitbucket Server (ehemals Stash) von Atlassian [8] eine ausgezeichnete Unterstützung für diesen Workflow (**Abb. 1**). Beide Tools sind eigentlich für die Verwaltung von Code-Repositories gedacht und bieten daher neben der guten Unterstützung für Codereviews auf Basis von Pull Requests noch viele weitere Funktionen.

Zunächst spielt der Entwickler seine Änderungen in der Versionsverwaltung Git auf einem separaten Feature-Branch ein. Anschließend markiert der Entwickler die Aufgabe in der Aufgabenverwaltung JIRA als abgeschlossen. JIRA fungiert als zentrales Tool, in dem zu jedem Zeitpunkt der Status einer Aufgabe abgerufen kann. Daher müssen alle Statusübergänge an JIRA gemeldet werden. Der Continuous-Integration-Server Bamboo erkennt automatisch die neuen Änderungen und führt den Build-Prozess auf dem Feature-Branch aus. Hierzu nutzt Bamboo seine automatische Branch Detection, die neue Branches anhand eines Namensmusters (z.B. „feature/578“) erkennen kann. Nach dem erfolgreichen Build meldet Bamboo den Statusübergang an JIRA. Der Entwickler kann den Zustand somit sowohl in Bamboo als auch in JIRA nachvollziehen. Nun erstellt der Entwickler

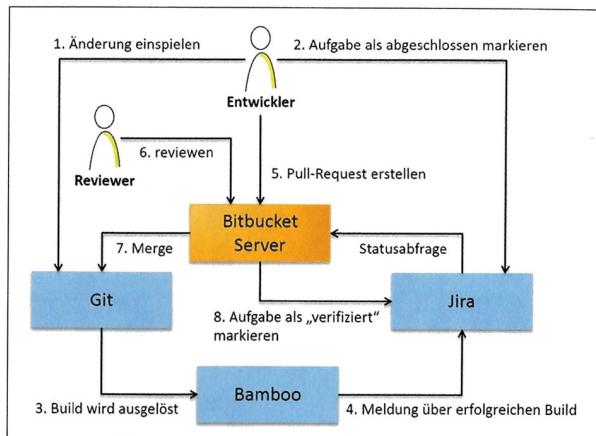


Abb. 1: Die Atlassian-Tools unterstützen die Entwickler beim Codereview mit einem definierten Workflow

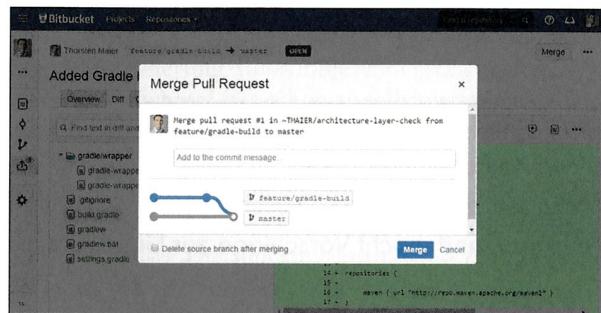


Abb. 2: Der Reviewer bekommt eine Benachrichtigung zum Pull Request in Bitbucket Server und kann dann mit dem Review beginnen

in der Repository-Verwaltung Bitbucket Server einen Pull Request und weist diesem Request einen Reviewer zu. Er gibt damit bekannt, dass seine Arbeiten abgeschlossen sind und die Änderungen zurück in den Hauptentwicklungszweig übernommen werden sollen. Der Reviewer bekommt eine Benachrichtigung über den neuen Pull Request und kann anschließend das Codereview über die Weboberfläche von Bitbucket Server ausführen (**Abb. 2**). Nach dem erfolgreichen Review werden die Codeänderungen des Feature-Branchs in den Hauptentwicklungszweig übernommen (Merge). Die Information über den erfolgreichen Merge wird an JIRA weitergereicht und die Aufgabe dort als *verifiziert* markiert.



Thorsten Maier arbeitet bei OIO Orientation in Objects. Er erschließt kontinuierlich bessere Wege, Software zu entwickeln, indem er selbst als Softwareentwickler unterwegs ist und anderen als Berater, Trainer, Autor und Speaker dabei hilft.

@ThorstenMaier

Links & Literatur

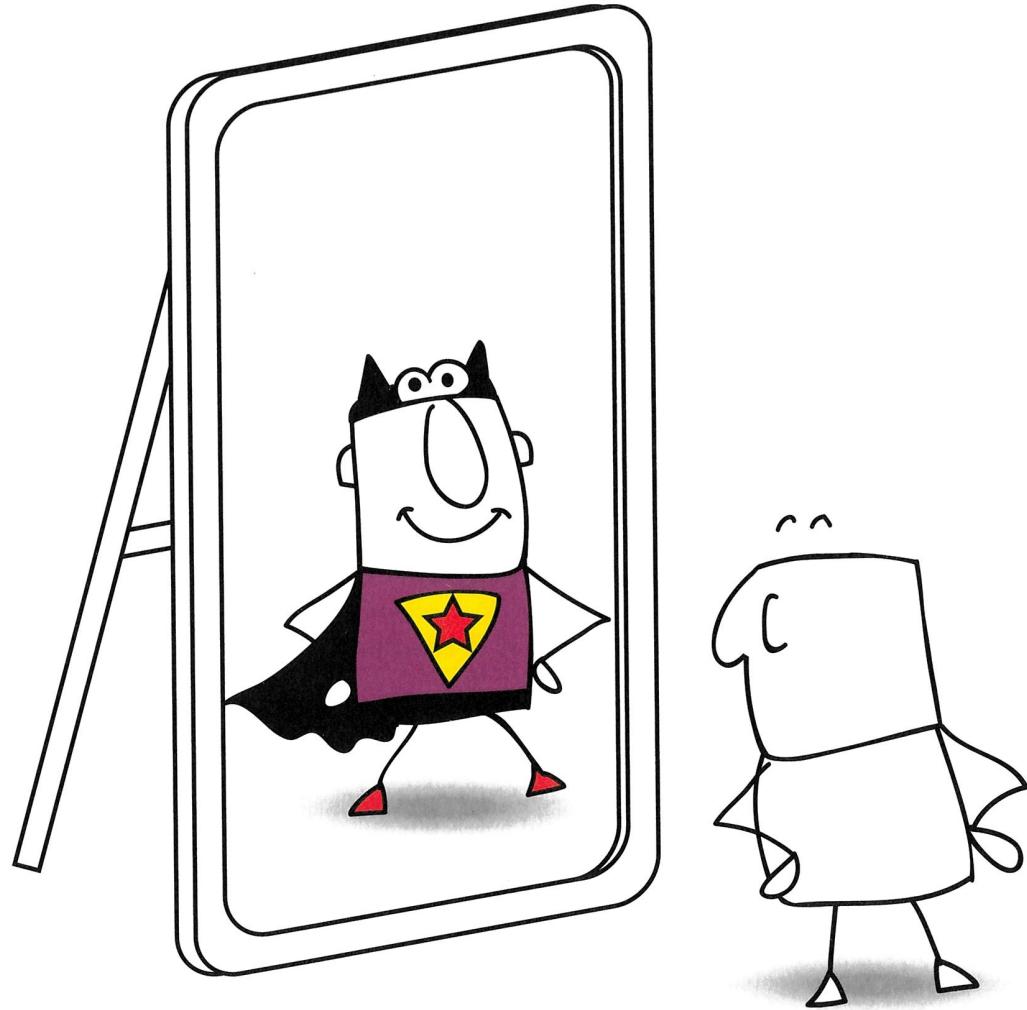
- [1] Boehm, Barry: „Software Engineering Economics“, Prentice Hall, 1981
- [2] SonarQube: <http://www.sonarqube.org/>
- [3] FindBugs: <http://findbugs.sourceforge.net/>
- [4] Checkstyle: <http://checkstyle.sourceforge.net/>
- [5] PMD: <https://pmd.github.io/>
- [6] jQAssistant: <http://jqassistant.org/>
- [7] Best Practices for Code Review: <https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/>
- [8] Bitbucket: <https://bitbucket.org/product/server>

Teamstruktur durch skalierte Agilität verbessern

Was sich am einfachsten ändert lässt: Sie selbst!

Wir sprechen von skalierten Agilität, sofern mehrere agil arbeitende Teams an einem Vorhaben beteiligt sind. Dieser Text beleuchtet einige der damit verbundenen Fragestellungen und macht Vorschläge zur Unterstützung erfolgreicher Skalierung.

von Ulf Schneider



Christophe BOISSON/Shutterstock.com

Ein Team ist mehr als eine Gruppe. Es ist gekennzeichnet durch seine Leistungsfähigkeit. Nicht ohne Grund wird die Hochphase der Teamentwicklung im Modell von Tuckmann [1] als Performancephase bezeichnet. Leistet ein Team nichts oder nur wenig, haben wir es vielleicht nur mit einer Gruppe von Menschen zu tun, die täglich dieselbe Kaffeemaschine benutzen und Smalltalk pflegen. Die Organisationsfähigkeit eines Teams zur Erreichung gemeinsam getragener Ziele, das Ablaufenlassen der nötigen Verhandlungsprozesse, das Treffen und Einhalten von Entscheidungen, das Erreichen von Resultaten, all das sind Voraussetzungen für effizientes, teamorientiertes Arbeiten. Derartiges Verhalten führt zur spezifischen Identität des Teams, die weder durch Vorgesetzte angeordnet noch durch Geld gekauft werden kann. Das Team entscheidet selbst, ob es ein Team ist [2].

Dreh- und Angelpunkt für agile Vorhaben, egal wie umfangreich, ist das Team. Gemeint ist das interdisziplinär besetzte Team mit weniger als zehn Mitgliedern. Es ist mit möglichst geringen Abhängigkeiten zu anderen Teams durch Selbstorganisation in der Lage, regelmäßige und in kurzen Abständen funktionsfähige Produkterweiterungen zu erstellen, zu testen und in einer definierten Umgebung ablaufen zu lassen. Dieser iterative Arbeitsansatz ist besonders geeignet zum Erbringen von Ergebnissen in komplexen Umfeldern, in denen Anpassung und Lernen aller Beteiligten absolut notwendig sind und in denen detaillierte Aprioriplanungen viel kosten und wenig nutzen [3].

Anpassen oder sterben

Jede Organisation muss dem Gesetz des Lebens folgen und sich im Laufe der Zeit an veränderte Bedingungen anpassen – oder sterben. Agilität unterstützt Anpassungsfähigkeit in komplexen Umfeldern. Je wichtiger Lernen und Anpassen in Ihrem Umfeld sind, umso mehr kann agiles Arbeiten Ihre Organisation unterstützen.

Agilität erweitert den Teamgedanken um die iterative Lieferung von Produkterweiterungen. Wenn sie schnell sein wollen, dürfen Teams nicht auf andere Teams warten. Entscheidungen innerhalb der Teams müssen zu Resultaten führen. Teams müssen interdisziplinär und lösungsorientiert besetzt sein, sodass die Teammitglieder mit ihrer Arbeit verbunden sind, nach Meisterschaft streben können und in der Lage sind, sich selbst und ihre Ergebnisse zu hinterfragen und zu verbessern. Für mich hat Agilität mit Sinn, Freude am Schaffen und Potenzialentwicklung zu tun. Das lässt sich nicht mit routiniertem Wohlverhalten und dem Vermeiden von Fehlern erreichen. Vielmehr geht es um das Suchen von Erfolg, die Kunst des Machbaren und die Herausforderung des Status quo [4], [5]. Ist es ein Ziel der Organisation, mit agilen Teams zu operieren, sich Randbedingungen zu setzen, die genau diese beiden Aspekte stärken: selbstorganisierte Teams und wiederkehrende Lieferung von Ergebnissen in kurzen Zeitabständen.

Jede Organisation muss dem Gesetz des Lebens folgen und sich im Laufe der Zeit an veränderte Bedingungen anpassen – oder sterben.

An Entwicklungsteams wird die Forderung der regelmäßigen Lieferung funktionierender Lösungen leicht gestellt, manchmal fast schon zu selbstverständlich. Haben Sie als Führungskraft ein entsprechendes Verhalten eigentlich auch schon für sich selbst im Kontext des Managementteams formuliert und umgesetzt? In welchen Zeitabständen räumen Sie organisatorische Hemmnisse aus dem Weg? Die Führungskräfte sind dafür verantwortlich, wie die Organisation als Ganzes funktioniert. Also auch, wie skalierte Agilität funktioniert.

Erfolg: Beginnen Sie vorne. Erarbeiten Sie für Ihren wettbewerblichen und organisatorischen Kontext Antworten auf die Fragen: Was bedeutet Erfolg für uns? Wann wissen wir, dass wir Erfolg haben? Können wir Schritte auf dem Weg zum Erfolg messbar machen? Involvieren Sie die gesamte Organisation und kommunizieren Sie die Ergebnisse. Können Sie eine Verbindung zwischen Ihrer Erfolgsdefinition und agilem Arbeiten herleiten? Wenn ja, kann das die Energie in Ihrer Organisation freisetzen, die es braucht, um mit Ihrer agilen Transformation erfolgreich zu sein.

Führen: Mir ist kein skaliert-agiles Vorhaben bekannt, das ohne aktives Zutun der Führungskräfte erfolgreich war. Wenn Sie als Führungskraft nicht überzeugt sind, wenn Sie nicht in die Richtung agilen Arbeitens führen, scheitern Sie. Wenn Sie selbst nicht teamorientiert arbeiten und regelmäßig Ergebnisse liefern, scheitern Sie. Auch ein Schaffen von Freiräumen für die Entwicklungsteams ohne zugehörige Führung genügt nicht. Es entsteht lediglich ein Vakuum, das beliebig ausgefüllt wird. Die zentrale Aufgabe von Führungskräften in agilen Organisationen ist, Randbedingungen zu schaffen, die den Selbstorganisationskräften eine Richtung geben. Sehr vereinfacht gesagt: Sprechen Sie mehr über das *Was* und weniger über das *Wie*. Das macht die Richtung der Selbstorganisation klar und gibt gleichzeitig den nötigen Freiraum dazu.



Abb. 1: Erfolgreich skalieren heißt nicht einfach nur größer machen

Wenn Sie mit einem Team erfolgreich sind, skalieren Sie Schritt für Schritt nach dem Bedarf Ihres Vorhabens um weitere Teams. Bauen Sie auf den Lerneffekten der bereits erfolgreichen Teams auf.

Transition: Verfolgen Sie diesen einfachen Ansatz [6]: Beginnen Sie mit einem einzelnen Entwicklungsteam und arbeiten Sie so lange, bis dieses Team verlässlich in jeder Iteration eine qualitätsgesicherte Produkterweiterung erzeugt, die in einer definierten Umgebung abläuft. Unterstützen Sie das gesamte Vorhaben durch ein Transitionsteam. Dieses ist für die Anpassung der umgebenden Organisation verantwortlich. Es ist mit Menschen besetzt, die in der Lage sind, die Organisation zu ändern und Probleme zu lösen, die nicht durch das Entwicklungsteam gelöst werden können. So wie das Entwicklungsteam eine priorisierte Liste mit erledigten und zu erledigenden Arbeiten pflegt, so pflegt das Transitionsteam eine Liste für Organisationsanpassungen. Beide Listen sind für alle Beteiligten offen einzusehen. Nutzen Sie die Hinweise des Entwicklungsteams. Sofern das Entwicklungsteam nicht in der Lage ist, in regelmäßigen Abständen Ergebnisse zu liefern, analysieren Sie die Gründe dafür. Liegen die Hemmnisse in der umgebenden Organisation, haben Sie genau die Arbeit identifiziert, die durch das Transitionsteam zu leisten ist.

Wenn Sie mit einem Team erfolgreich sind, skalieren Sie Schritt für Schritt nach dem Bedarf Ihres Vorhabens um weitere Teams. Bauen Sie auf den Lerneffekten der bereits erfolgreichen Teams auf. Beachten Sie die Randbedingung: Jedes Team sollte in der Lage sein, Verantwortung für die erzielten Ergebnisse zu übernehmen. Dazu sind Abhängigkeiten zwischen Teams zu minimieren.

Möglicherweise sehen Sie die Notwendigkeit, sofort mit mehr als einem Team zu beginnen. Das ist zwar möglich, aber schwieriger. Meiner Ansicht nach erspart Ihnen auch keines der einschlägigen agilen Scaling Frameworks, wie zum Beispiel SAFe [7], den oben beschriebenen Prozess der aktiven Organisationsgestaltung durch ein Transitionsteam. Ich gehe so weit zu behaupten, dass ein agiles Scaling Framework ohne Transitionsteam und ohne lernendes Management nur Dekoration ist. Sehen Sie sich die ScALED Principles [8] an, die meiner Meinung nach eine gute Hilfestellung zur Skalierung Ihres Vorhabens bieten.

Positive Abweichung: Versuchen Sie positive Lösungsmuster, die an einer Stelle bereits funktionieren, zu erkennen und in Bereiche zu übertragen, in denen für ähnliche Probleme bisher nur schlechte oder keine

Lösungen gefunden wurden [9]. Sie müssen nicht alles auf den Kopf stellen. Die Evolution fing auch nicht jedes Mal von vorne an. Bauen Sie auf dem auf, was funktioniert.

Aufgaben klar machen: Ein Team kann selbstorganisiert nur etwas erreichen, wenn das zu lösende Problem oder das zu erreichende Ziel klar ist. Sind Sie sicher, das richtige Ziel zu verfolgen? Wenn ja, ist das Ziel für *alle* Beteiligten klar? Ohne diese Ausrichtung erhält auch die Selbstorganisation keine Richtung. Sind Sie in der Lage, das Ziel in priorisierte Zwischenschritte herunterzubrechen, für jedes Team, für jede Iteration, in Form qualitätsgesicherter Produkterweiterungen?

Kurze Releases: Eine große Hilfe zum positiven Umgang mit allen zuvor genannten Punkten ist die Verkürzung der Releasezyklen auf drei oder vier Monate und die Unterteilung dieser Releases in zwei- bis dreiwöchige Iterationen.

- Legen Sie vor jedem Release ein Release Goal schriftlich fest. Das Release Goal ist nicht länger als eine halbe Seite.
- Legen Sie den Iterationsplan und die Termine für Abstimmungen, Ergebnisprüfungen und Retrospektiven fest.
- Erarbeiten Sie gemeinsam mit allen Mitgliedern ihres Vorhabens innerhalb der ersten Iteration Anforderungsskizzen für die Funktionen, die in dem Release geliefert werden sollen. Jeff Pattons User Story Mapping [10] ist ein ideales Verfahren, um den Zusammenhang des Releases herauszuarbeiten und Funktionskandidaten zu identifizieren, die Sie weiter vertiefen möchten. Halten Sie nicht funktionale Anforderungen fest und identifizieren Sie die Bereiche, in denen Architekturentscheidungen getroffen werden müssen. Unterlassen Sie die Analyse und Vorbereitung von Anforderungen, die nicht Bestandteil des Release Goals sind.
- Liefern Sie immer die wichtigsten Dinge zuerst. Verschieben Sie nicht den Releasetermin, sondern streichen Sie die unwichtigeren Funktionen, falls die Zeit nicht reicht.

Die Kürze des Releases vereinfacht vieles: Die Zeitknappheit führt zu größerer Klarheit sowohl für die

Priorisierung als auch für die Aufbereitung der Aufgaben, da zwangsläufig weniger Dinge erledigt werden können. Das kurze Release ist ein Container, der Selbstorganisation und Zusammenarbeit erzwingt. Gleichzeitig ist die Planung des kürzeren Zeitraums deutlich einfacher.

Fakten sichtbar machen: Machen Sie Fakten für alle sichtbar. Dazu gehört, den Fortschritt Ihres Vorhabens nicht mit einem Verbrauchsmaß zu messen, z.B. verbrauchtem Aufwand. Nutzen Sie ein Fortschrittsmaß, z.B. fertiggestellte Softwarefunktionen. Die Kunst besteht darin, mit Metriken zu arbeiten, die sich normierend auf individuelles und soziales Handeln auswirken. Dahinter steht erneut die Frage, wie in der Organisation Erfolg definiert ist und wie der Fortschritt auf dem Weg zum Erfolg sichtbar gemacht wird. Die OLUPCAT-Merkmale, die durch agile Metriken erfüllt sein sollten, geben eine Orientierung:

- **On demand:** Es wird erst gemessen, wenn der Bedarf für die Messung klar ist. Zum Beispiel, wenn ein Problem erkannt ist und der Weg zur Besserung gemessen werden soll. Oder, wenn man sich ein Ziel gesetzt hat und mit einer Metrik den Fortschritt oder Zielerreichungsgrad ermitteln will. Daten zu erheben, nur weil man es kann, bringt niemanden weiter.
- **Lightweight:** Agile Metriken haben einen kleinen Fußabdruck. Sie sind methodisch einfach zu ermitteln und ohne hohen Aufwand im Rahmen der Arbeitsprozesse zu erheben – idealerweise automatisiert.
- **Understandable:** Jeder in der Organisation versteht die Metrik und weiß, warum sie angewendet wird, also welches Ziel durch die Metrik unterstützt wird.
- **Periodic:** Um kurze Rückkopplungsschleifen zu haben, wird kontinuierlich in kurzen Zeiträumen gemessen, z. B. in Arbeitstagen, Iterationen und Releases. Die Auswertung der Messung steht unmittelbar nach der Messung zur Verfügung.
- **Foster Collaboration:** Agile Metriken unterstützen die Zusammenarbeit. Das bedeutet, es werden nicht Sachverhalte gemessen, die im direkten Einflussbereich einer Person liegen. Stattdessen misst man eine Ebene oberhalb, sodass Verhaltensanpassungen auf dem Wege der Zusammenarbeit stattfinden müssen. Dieses Vorgehen schützt vor lokaler Optimierung, die einer End-to-End-Optimierung möglicherweise entgegenstehen würde.
- **Accessible:** Die Metrik wird nicht nur durch einen kleinen Kreis von Mitarbeitern ausgewertet, sondern alle betroffenen Mitarbeiter haben jederzeit Zugang zu den aktuellen Daten. Es gibt keine Reporting-Parallelwelt für das Management. Nur so kann sich die Lenkungsfunktion einer Metrik entfalten.
- **Trend-oriented:** Wenn Metriken unser Verhalten beeinflussen und ändern, werden sich in der Umkehr auch die gemessenen Werte ändern. Die Entwicklungsrichtung und Geschwindigkeit dieser Änderung sind wichtiger als absolute Zahlen.

Eine Randnotiz dazu: Der Scrum Guide [11] kennt keine Story Points. Fortschritt wird in fertiggestellten Product Backlog Items gemessen. Ein Item muss in einem Sprint lieferbar sein. Komplizierte Story-Point-Kalkulationen und Velocity-Paraden sind in diesem Framework nicht vorgesehen. Story Points können hilfreich sein, sie sind aber kein Selbstzweck. Im Mittelpunkt steht immer noch, fertige Software in jeder Iteration zu erzeugen.

Fazit

Auch wenn hier besonders Fragestellungen behandelt werden, die den Umgang mit Größe handhabbar machen sollen – die beste Skalierung ist so klein wie möglich, weil Adaptions- und Lernprozesse schneller ablaufen, je kleiner die Organisation ist. Und vergessen Sie nicht: Der Teil eines sozialen Systems, der sich am einfachsten ändert, sind Sie selbst!



Ulf Schneider ist Agile Coach bei Wincor Nixdorf und arbeitet gemeinsam mit seinen Kollegen an der Transition der weltweit operierenden Softwareproduktentwicklung hin zur kontinuierlichen Lieferung von Produkterweiterungen. Er teilt seine Gedanken zu agilen Arbeitsweisen auf <http://www.ulf.codes>.

Links & Literatur

- [1] Tuckman's stages of group development: https://en.wikipedia.org/wiki/Tuckman%27s_stages_of_group_development
- [2] Lencioni, Patrick: „The Five Dysfunctions of a Team: A Leadership Fable“, John Wiley & Sons, 2002
- [3] Schaber, Ken; Beedle, Mike: „Agile Software Development with Scrum“, Pearson International Education, 2002
- [4] Mayer, Tobias: „The Peoples's Scrum“, Dymaxicon, 2013
- [5] Godin, Seth: „Linchpin. Are You Indispensable?“, Portfolio, 2010
- [6] Schwaber, Ken: „The Enterprise and Scrum“, Microsoft Press, 2007
- [7] SAFe: <http://scaledagileframework.com>
- [8] ScALeD Principles: <http://www.scaledprinciples.org>
- [9] Pascale, Richard T.; Millemann, Mark; Gioja, Linda: „Surfing the Edge of Chaos“, Crown Business, 2001
- [10] Patton, Jeff: „User Story Mapping. Discover the Whole Story. Build the Right Product“, O'Reilly Media, 2014
- [11] Scrum Guide: <http://www.scrumguides.org>