Zeitschrift: Bulletin Electrosuisse
Herausgeber: Electrosuisse, Verband für Elektro-, Energie- und Informationstechnik
Band: 101 (2010)
Heft: (10)

Artikel: Scrum : lieber spät als nie!
Autor: Jocham, Ralph
DOI: https://doi.org/10.5169/seals-856141

Nutzungsbedingungen
Die ETH-Bibliothek ist die Anbieterin der digitalisierten Zeitschriften auf E-Periodica. Sie besitzt keine Urheberrechte an den Zeitschriften und ist nicht verantwortlich für deren Inhalte. Die Rechte liegen in der Regel bei den Herausgebern beziehungsweise den externen Rechteinhabern. Das Veröffentlichen von Bildern in Print- und Online-Publikationen sowie auf Social Media-Kanälen oder Webseiten ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Mehr erfahren

Conditions d'utilisation
L'ETH Library est le fournisseur des revues numérisées. Elle ne détient aucun droit d'auteur sur les revues et n'est pas responsable de leur contenu. En règle générale, les droits sont détenus par les éditeurs ou les détenteurs de droits externes. La reproduction d'images dans des publications imprimées ou en ligne ainsi que sur des canaux de médias sociaux ou des sites web n'est autorisée qu'avec l'accord préalable des détenteurs des droits. En savoir plus

Terms of use
The ETH Library is the provider of the digitised journals. It does not own any copyrights to the journals and is not responsible for their content. The rights usually lie with the publishers or the external rights holders. Publishing images in print and online publications, as well as on social media channels or websites, is only permitted with the prior consent of the rights holders. Find out more

Download PDF: 13.07.2025

ETH-Bibliothek Zürich, E-Periodica, https://www.e-periodica.ch

Scrum – lieber spät als nie!

Wie empirische Softwareentwicklung zum Erfolg führt

Nicht definiert, sondern empirisch; «Leadership and Collaboration» anstatt «Command and Control» – für diese Denkweise steht Scrum. Dieser empirische Ansatz, der der Komplexität der meisten Softwareprojekte gerechter wird als sequenzielle Entwicklungsprozesse, hat die Chance, die Softwareentwicklung zu revolutionieren.

Ralph Jocham

Bis heute hat es die Softwareindustrie nicht geschafft, die Entwicklung unter Kontrolle zu bringen. Im Jahr 2008 sind knapp 30 % der Projekte fehlgeschlagen. Was waren die Gründe? Die noch weitverbreiteten definierten und sequenziellen Entwicklungsprozesse können die Komplexität der Softwareentwicklung nicht genügend abbilden. Solche Prozesse wurden Ende der 1960er-Jahre als Antwort auf das Ad-hoc-Programmieren eingeführt. Wohlgemerkt waren zu diesem Zeitpunkt schon positive Erfahrungen mit disziplinierter iterativer und inkrementeller Entwicklung bekannt. Einige dieser Projekte wurden in den späten 1950er-Jahren durchgeführt. Doch der Wasserfallprozess, Synonym für alle sequenziellen Entwicklungsprozesse, hat sich breit durchgesetzt. Warum? Dies geht zum Grossteil auf Winston Royce zurück. In seinem Artikel «Managing the Development of Large Software Systems» (Proceedings of IEEE, 1970) beschreibt Royce den Wasserfall als Beispiel. Sein Kommentar: «Ich glaube an das Konzept, jedoch die Umsetzung, die oben beschrieben wird, ist voller Risiken und schwört Fehler herbei. (...) Die Testphase, welche zum Schluss der Entwicklung vorkommt, ist das erste Ereignis, bei welchen Zeitaspekte, Speicher, Ein-Ausgabe, Übertragung usw. im Unterschied zur Analyse erlebt werden.» Er verwendete den Wasserfall also als Negativbeispiel und beschreibt auf den folgenden Seiten einen Prozess mit Feedbackschleifen. Doch leider hat so gut wie niemand den Artikel zu Ende gelesen, und so ist nur die Darstellung des Wasserfalls in Erinnerung geblieben. Somit erlangte Royce als «Erfinder» des Wasserfallprozesses undankbarerweise Weltruhm. David Meyerbeer ging es nicht anders: Er verfasste den Standard DOD MIL-STD 2167A, der 1988 vom US Departement of Defense als Softwareentwicklungsprozess vorgeschrieben wurde. Dieser Wasserfallprozess war die Vorlage für andere Prozesse weltweit: JSP-188 (Grossbritannien), GAM-T-17 (Frankreich) und V-Model (Deutschland), der als Basis für Österreich und die Schweiz (Hermes) diente. Diese Entscheidung kam den Steuerzahler sehr teuer zu stehen. 1994, nach nur sechs Jahren, wurde der DOD MIL-STD 2167A durch den DOD MIL-498 abgelöst. Dieser neue Prozess unterstützt bewusst iterative und inkrementelle Softwareentwicklung. Leider bekamen die anderen Länder dies nicht mit oder ignorierten die bedeutende Änderung. Der Glaube an ein definiertes Vorgehen mit sequenziellen Prozessen hat sich dort länger gehalten oder hält sich immer noch.

Definiert funktioniert nicht

Dass die Softwareentwickler in den 1960er-Jahren auf definierte und sequenzielle Entwicklungsprozesse setzten, lässt sich erklären: Zum einen ging man damals noch fest von dem Gedanken aus, dass Computerprogramme wie Brücken gebaut werden können: Einige wenige analysieren das Problem, planen und entwerfen das Projekt. Bauarbeiter erhalten die Pläne und setzen diese um. «Plan the work, work the plan» – dieses Denken war damals dominant. Ebenfalls wurde viel Hoffnung in die Fähigkeit von Computern und deren Programme gesetzt, speziell im Bereich der künstlichen Intelligenz. Diese Erwartungshaltung ist auch in Stanley Kubricks Film «2001: A Space Odyssey» aus dem Jahr 1968 klar zu erkennen: Der neurotische Computer H-A-L 9000 an Bord von Discovery One ist weltbekannt. Solche sich selbst programmierende Computer wurden damals als Standard in 20 Jahren vorausgesagt.

Wo stehen wir heute, nach der doppelten Zeitspanne, also über 40 Jahre später? Der grosse Durchbruch lässt noch immer auf sich warten. Die erfolgreichsten Innovationen in der Softwareentwicklung sind die Konzepte der Objektorientierung und virtueller Maschinen. Die wichtigste Erkenntnis ist jedoch nicht bei der Technologie, sondern im Management zu finden: Es ist die Einsicht, dass ein definiertes Vorgehen im Allgemeinen nicht funktioniert.

Zu hohe Komplexität

Warum funktioniert «definiert» nicht? Das Cynefin-Framework, 1999 vom IBM-Forscher Dave Snowden entwickelt, veranschaulicht die Problematik. Cynefin ist ein walisisches Wort und bedeutet so viel wie Lebensraum. In diesem Raum sind vier Abschnitte definiert, nach denen Projekte kategorisiert werden können. Simple (einfach), Complicated (kompliziert), Complex (komplex), Chaotic (chaotisch).

Einfache und komplizierte Projekte können noch durch ein definiertes Vorgehen umgesetzt werden. Komplexe Projekte, bei denen weder die Anforderungen noch die Technologie einfach und

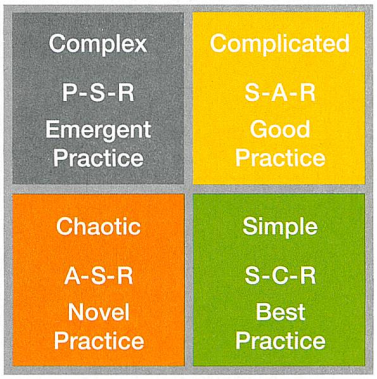


Bild 1 Das Cynefin-Framework definiert vier Räume, nach denen Projekte kategorisiert werden können: Simple (einfach), Complicated (kompliziert), Complex (komplex), Chaotic (chaotisch).

ganz verstanden sind, jedoch nicht mehr. Jedes nicht triviale Softwareprojekt fällt in diese Kategorie. Sie benötigen ein empirisches Vorgehen. Mit definierten sequenziellen Prozessen wird hier also das falsche Management-Tool eingesetzt. Grosse Probleme sind von Anfang an programmiert.

Im Durchschnitt ändern sich während der Entwicklung 35 % der Anforderungen, 60 % der implementierten Anforderungen werden selten oder nie vom Endkunden verwendet. Dies zeigt, dass ein Requirements Engineering (Definition der Anforderungen) zu Beginn des Prozesses nur sehr bedingt funktioniert. Anforderungen müssen bis zum letzten Moment entdeckbar und änderbar sein.

Zudem sind die heutigen mehrschichtigen Architekturen mit heterogenen Technologien alles andere als einfach. Kurz: Wir leben auch in der Softwareentwicklung in einer komplexen Welt.

Scrum: Einfache Regeln, komplexe Umsetzung

Inspiriert vom Konzept Lean aus dem Total Quality System von Toyota sowie vom Artikel «The New New Product Development Game» von Takeuchi und Hirotaka in der Harvard Business Review begann Ken Schwaber gegen Ende der 1980er-Jahre, den Scrum-Prozess zu entwickeln. Sein Ziel war, einen iterativen und inkrementellen Entwicklungsprozess zu definieren, der die oben genannten Einsichten berücksichtigt. Dies war kein leichtes Unterfangen, doch 1995 stellten Ken Schwaber und Jeff Sutherland an der OOPSLA in Austin den Scrum-Prozess der Öffentlichkeit vor.

Auf den ersten Blick erscheint Scrum sehr einfach. Der Scrum-Guide, der den Prozess ausführlich beschreibt, hat gerade mal 24 Seiten. Scrum ist wie Schach, die Regeln sind schnell gelernt. Doch um ein Meister zu werden, braucht es viel Praxis. Am besten wird man Lehrling bei einem Meister und lernt direkt von ihm. Daher bietet Scrum.org auch Trainings an. Sie vermitteln praktische Erfahrungen aus erster Hand von einem erfahrenen Scrum Master. Kombiniert mit eigenem Engagement und Selbststudium, bilden diese Trainings die besten Voraussetzungen, um sich für die Herausforderungen eines Scrum Masters zu rüsten. Denn es ist nicht trivial, Scrum in eine Firma einzuführen – viele Scrum Master scheitern daran. Trotzdem: Auch eine gescheiterte Scrum-Einführung bringt in der Regel bereits eine gewisse Produktivitätssteigerung. Viele Firmen geben sich damit bereits zufrieden. Erfolgreiche Scrum Master denken jedoch in anderen Grössenordnungen: Sie erwarten eine deutliche Steigerung der Produktivität bis zu einem Faktor zwei.

Der Sprint

Scrum ist iterativ. Eine Iteration, genannt Sprint, dauert maximal 30 Tage. Ein Sprint darf nie verlängert werden. Eine Verkürzung ist erlaubt, wenn sich erhöhte technische Risiken zeigen oder die Anforderungen noch sehr unklar sind. Kurze Sprints von zwei oder drei Wochen sind nicht unüblich, kürzer als eine Woche ist jedoch in der Regel nicht sinnvoll. Nach einem Sprint, also nach spätestens einem Monat, wird dem Kunden funktionierende Software vorgeführt. Dieses sogenannte Software-Inkrement ist voll funktionell, das heisst, es muss dem Kunden einen Wert bieten. Dies bedingt meistens, dass die Entwickler durch alle Architekturschichten hindurch programmieren, von der Benutzeroberfläche bis hinunter zur Datenspeicherung. Diese Arbeitsweise hat positive Effekte: Ein schönes User Interface allein ist zwar ansprechend, aber ohne funktionierende Business-Logik ohne Wert. Ein zentrales Anliegen von Scrum ist die kontinuierliche Wertschöpfung. Jedes Inkrement am Ende eines Sprints ist ein Potentially Shippable Product Increment, muss also Lieferqualität aufweisen.

Klare Verantwortlichkeiten

Der Product Owner ist verantwortlich für die Produktanforderungen. Diese sind in einer Liste festgehalten, die nach Priorität der Anforderungen absteigend sortiert ist. In diesem sogenannten Product Backlog werden die Ausdrücke «muss», «soll» und «kann» bewusst nicht verwendet. In typischen realen Projekten werden 90 % der Anforderungen mit «muss» markiert. Doch welche dieser Anforderungen ist nun wichtiger als eine andere? Die klare Priorisierung von Scrum vermeidet solche Konflikte. Der Product Owner pflegt den Product Backlog beständig und stellt zu jedem Zeitpunkt sicher, dass das richtige und nicht einfach das ursprünglich spezifizierte Produkt entwickelt wird. Über den Product Owner laufen alle Fragen zum Produkt, und er kann den Product Backlog jederzeit aufgrund neuer Erkenntnisse anpassen. Dabei kann er sich mit anderen Fachleuten beraten, die Entscheidungen liegen letztlich in seiner Hand. Es gibt somit eine klar verantwortliche Person. Der Erfolg der Entwicklung liegt voll und ganz in den Händen des Product Owners. Mit anderen Worten: Der Product Owner ist die Personifizierung des Return on Investment (ROI).

Die Organisation der Arbeiten innerhalb eines Sprints, einer Iteration, liegt hingegen nicht in der Kompetenz des Product Owners. Diese werden im Sprint Planning Meeting vom Team definiert. Die wichtigsten Requirements, das «Was», werden vom Product Owner zusammen mit dem Sprint-Goal vorgestellt. Das Sprint-Goal ist ein kurzer Text, der

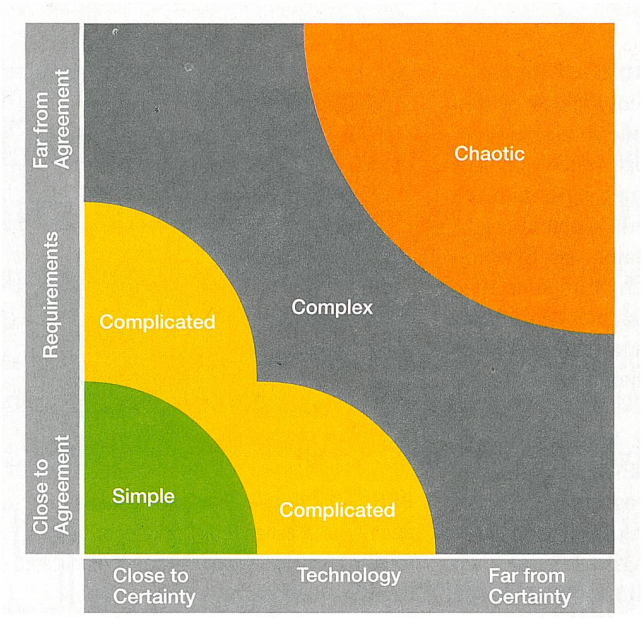
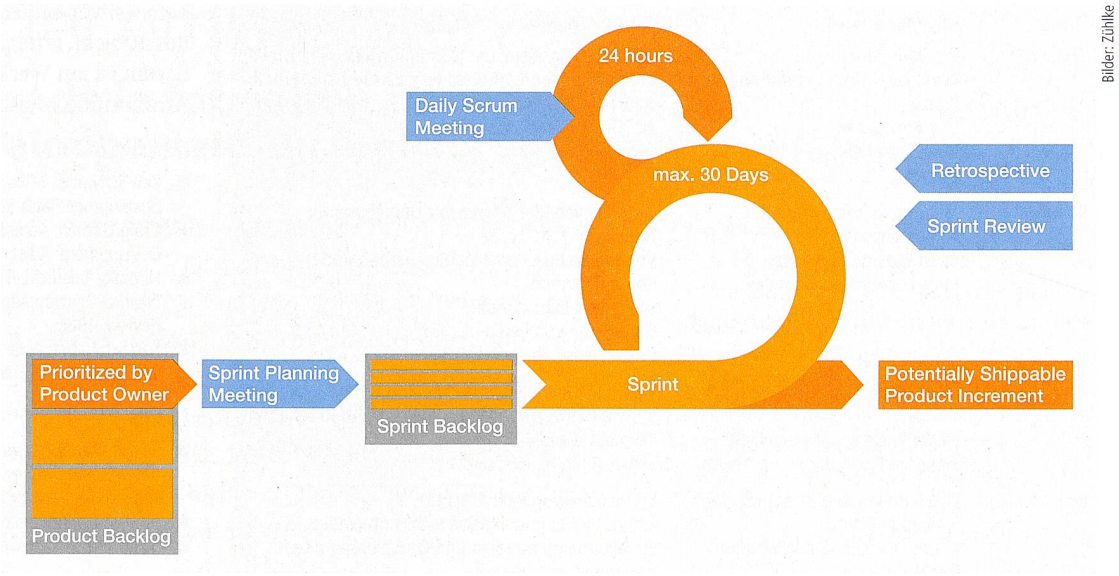


Bild 2 Je weniger klar die Anforderungen und je unsicherer die Technologie, desto komplexer wird ein Projekt. Einfache und komplizierte Projekte können noch durch ein definiertes Vorgehen umgesetzt werden. Komplexe Projekte jedoch nicht mehr – sie benötigen ein empirisches Vorgehen.



Bilder: Zühlke

Bild 3 Ablauf eines Scrum-Prozesses.

das Ziel in Worten beschreibt. Alle Aktivitäten im kommenden Sprint dienen dem Erreichen des Sprint Goals. Während die Requirements vom Product Owner beschrieben werden, hat das Team die Möglichkeit, klärende Verständnisfragen zu stellen. Wenn alle Fragen beantwortet wurden, wird der Aufwand geschätzt. Durch Erfahrungswerte aus vorausgegangenen Sprints kann das Team abschätzen, wenn es seine Kapazität erreicht hat. Anschliessend erarbeitet das Team, «wie» das «Was» erreicht werden kann. Die resultierenden Tasks bilden den Sprint-Backlog. Dieser gehört ausschliesslich dem Team.

Im anschliessenden Sprint trifft sich das Team täglich zum Daily Scrum. In diesem Meeting beantwortet jedes Mitglied drei Fragen: Was habe ich seit dem letzten Daily Scrum gemacht? Was mache ich bis zum folgenden Daily Scrum? Habe ich ein Problem, das mich vom Fortschritt abhält? Das Daily Scrum dient der Pflege der Feedbackkultur im Team und ist ein wesentlicher Bestandteil der Qualitätssicherung und Steigerung der Teameffizienz. Der Sprintfortschritt wird auf einer Sprint Burndown Chart täglich gemessen und einsehbar gemacht.

Implementierung prüfen

Am Ende des Sprints werden alle implementierten Anforderungen dem Product Owner und anderen Stakeholdern vorgeführt. Als implementiert gilt eine Anforderung, wenn sie zu 100 % erfüllt ist. Was 100 % oder «potentially shippable» bedeutet, wird in der sogenannten Definition of Done festgelegt. Sie ist verbindlich. Genügt eine Anforderung nicht zu 100 % der Definition of Done, so gilt sie als nicht implementiert. Teilimplementierungen werden nicht anerkannt. Die implementierten Anforderungen werden am Review vorgeführt. Jeder ist zu diesem Meeting eingeladen und aufgefordert, Feedback abzugeben. Die Anforderungen werden aufgrund der Rückmeldungen überprüft, allenfalls adaptiert und fliessen in einen angepassten Product Backlog ein. In der abschliessenden Retrospektive reflektiert das Team den letzten Sprint: Was lief schlecht? Was war gut? Daraus werden Verbesserungen für den folgenden Sprint abgeleitet. So wird die Teamarbeit laufend überprüft und verbessert.

Auch die Sprint-Retrospektive dient der Pflege der Feedback-Kultur im Team als weiterer wesentlicher Bestandteil der Qualitätssicherung und Steigerung der Teameffizienz. Am folgenden Tag startet der nächste Sprint mit dem Sprint Planning Meeting – der Kreis schliesst sich.

Drei Säulen, drei Rollen

Das empirische Prozessführungsmodell von Scrum basiert auf drei Säulen: Transparenz, Inspektion und Adaption. Transparenz ist notwendig, damit der momentane Zustand jederzeit oder zu wohldefinierten Zeitpunkten überprüft werden kann. Transparenz gilt für alle Stufen in Scrum, von der Definition der Anforderungen über das Release Goal und die einzelnen Sprint Goals bis hin zum täglichen Stand der Dinge, dem Daily Scrum und dem Sprint Burndown. Zum Schluss des Sprints wird das Produktinkrement transparent gemacht und zur Inspektion freigegeben. Dies ermög-

Typ	Beschreibung
Simple (einfach)	Das Verhältnis zwischen Ursache und Wirkung ist allen offensichtlich **Vorgehen** Sense, Categorize, Respond (Wahrnehmen, Kategorisieren, Antworten) Best Practices können verwendet werden
Complicated (kompliziert)	Das Verhältnis zwischen Ursache und Wirkung erfordert eine Analyse oder eine andere Form von Nachforschung und/oder die Anwendung von Expertenwissen **Vorgehen** Sense, Analyse, Respond (Wahrnehmen, Analysieren, Antworten) Good Practices können verwendet werden
Complex (komplex)	Das Verhältnis zwischen Ursache und Wirkung kann nur im Nachhinein wahrgenommen werden, aber nicht im Voraus **Vorgehen** Probe, Sense, Respond (Erforschen, Wahrnehmen, Antworten) Emergent Practices werden wahrgenommen
Chaotic (chaotisch)	Es gibt kein Ursache-und-Wirkungs-Verhältnis am Systemlevel **Vorgehen** Act, Sense, Respond (Handeln, Wahrnehmen, Antworten) Novel Practices werden entdeckt

Wikipedia

Tabelle 1 Beschreibung der Charakteristik von Projekten nach dem Cynefin-Framework.

Typ	Charakteristik	Leader-Aufgabe
Chaotisch	Viel Turbulenz Kein klares Ursache-und-Wirkungs-Prinzip Unerkennbares Viele Entscheidungen und keine Zeit	Sofortiges Handeln, um Ordnung wiederherzustellen Priorisierung und Auswahl von ausführbarer Arbeit Fokus auf das, was funktioniert, anstatt Perfektion Act, Sense, Respond
Komplex	Mehr unvorhersehbar als vorhersehbar Aufstrebende Antworten Viele konkurrierende Ideen	Erstellen von klar begrenzter Umgebung für Handlung Verstärkter Level an Zusammenarbeit und Kommunikation Dienender Führungsstil Generieren von Ideen Probe, Sense, Respond
Kompliziert	Mehr vorhersehbar als unvorhersehbar Faktenbasiertes Management Experten lösen Ungereimtheiten	Verwenden von Experten, um Einsicht zu erhalten Verwenden von Metriken, um Kontrolle zu erhalten Command and Control Sense, Analyze, Respond
Einfach	Wiederholende Muster und gleich bleibende Events Klares Ursache-und-Wirkungs-Prinzip Klar bekanntes Wissen Faktenbasiertes Management	Verwenden von Best Practices Extensive Kommunikation nicht notwendig Erstellung von Mustern und Optimierung dieser Command and Control Sense, Categorize, Respond

Ken Schwaber, Scrum.org

Tabelle 2 Beschreibung der Charakteristik von Projekten nach dem Cynefin-Framework.

licht, fortlaufend die nötigen Adaptionen vorzunehmen.

In Scrum gibt es nur drei Rollen: Product Owner, Scrum Master und Team. Der Product Owner ist, wie der Name schon sagt, für das Produkt zuständig. Es gehört ihm allein. Der Scrum Master ist verantwortlich für den Prozess und stellt sicher, dass Scrum im Team richtig gelebt wird. Er sorgt für das Team und löst aufkommende Probleme, welche das Team behindern, effizient zu arbeiten. Dies hört sich nach einer einfachen Aufgabe an, doch die Tücken stecken im Detail, gerade wenn Scrum neu in einer noch nicht agilen Firma eingeführt wird. Es ist in dieser Situation sinnvoll, wenn ein externer Coach die Rolle des Scrum Masters übernimmt. Er vermittelt dem Unternehmen, wie Scrum funktioniert und gelebt wird. Hat er dieses Wissen weitergegeben, zieht er sich zurück und trainiert seinen Nachfolger.

Das Team ist verantwortlich für die Entwicklung der Potentially-Shippable-Product-Inkremente, welche die Definition of Done erfüllen. Das Team muss daher funktionsübergreifend sein, das heisst, es besitzt alle nötigen Kenntnisse, um ein Produktinkrement zu entwickeln. Im Team gibt es keine unterschiedliche Rollen wie Entwickler, Architekt oder Tester. Alle Teammitglieder sind in ihrer Stellung gleichwertig und bringen ihr unterschiedliches Know-how mit. Das Team organisiert sich selbst und setzt sein Know-how mit dem Ziel ein, höchste Effizienz zu erreichen. Um effektiv kommunizieren zu können, arbeiten ideal alle Teammitglieder im selben Raum.

Die Zukunft

Scrum steht für eine völlig neue Denkweise. Eine Denkweise, die akzeptiert, dass Softwareentwicklung nicht definiert, sondern empirisch ist. Eine Denkweise, die nicht auf Command and Control, sondern auf Leadership and Collaboration aufbaut. Entscheidungen, die ein Projektteam ohne Einfluss des Managements fällt, bieten erhebliches Potenzial, indem sie Eigenverantwortlichkeit, Effizienz und Qualitätsbewusstsein im Team fördern. Einigen Managern fällt es schwer, dies zu akzeptieren. Doch die Erfolge von einigen Unternehmen sprechen für sich. Ist es nicht beeindruckend, wie eben noch unbekannte Firmen in wenigen Jahren unser Weltbild verändern und dominieren konnten? Solche Kraftakte sind nur in einem Kollektiv mit gefordertem und bemächtigtem Teamgeist zu erreichen. Das Management muss sich als Servant Leader, nach dem Motto «Führen heisst dienen», neu erfinden.

Ob es den Begriff Scrum in zehn Jahren noch geben wird, ist irrelevant. Entscheidend ist, dass die Ideen und das dahinterstehende Potenzial in der Projektkultur weitergetragen werden. Scrum hat dazu den Grundstein gelegt und soll auch so betrachtet werden. Scrum bietet die Chance, einen Industriezweig zu revolutionieren, der seit den Anfängen mit Termin-, Kosten- und Qualitätsproblemen zu kämpfen hat. Scrum ist jedoch kein «silver bullet», das automatisch jedes Projekt erfolgreich zum Ziel bringt. Scrum ist ein Werkzeug, das bei richtiger Anwendung Meisterstücke ermöglicht.

Literatur

- Ken Schwaber, Mike Beedle: Agile Software Development with Scrum, Prentice Hall, 2001.
- Craig Larman, Addison Wesley: Agile and Iterative Development: A Manager's Guide, Boston, 2003.
- Hirotaka Takeuchi, Ikujiro Nonaka: The New New Product Development Game, Harvard Business Review, 1986.
- Ken Schwaber: Agile Project Management with Scrum, Microsoft Press, 2004.

Links

- Scrum-Guide: Frei verfügbar auf dem Internet unter Scrum.org (www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%20DE.pdf)
- http://en.wikipedia.org/wiki/Cynefin
- Scrum-Kurse: http://courses.scrum.org/

Angaben zum Autor

Ralph Jocham, Change Agent und Agile Coach bei Zühlke Engineering. In dieser Position hilft er Kunden bei der Einführung von Agile und Scrum. Ralph Jocham hat über 13 Jahre internationale (DE, UK, USA, CH) Erfahrung mit der Anwendung von «Best Practices» in verschiedenen Domänen und Positionen. Seine Stärken liegen im Bereich agiles Coaching und Produktivitätserhöhung. Er ist Agile und «Test Infected» seit 2000. Ralph ist Europas erster Professional Scrum Master (PSM) und Professional Scrum Developer (PSD) Trainer für Scrum.org. Er ist ebenfalls ein CSM, CSP.

Zühlke Schweiz, 8952 Schlieren, ralph.jocham@zuehlke.com

Résumé

Scrum – mieux vaut tard que jamais !

Ou comment le développement empirique de logiciels mène au succès

Jusqu'à aujourd'hui, l'industrie des logiciels n'est pas parvenue à maîtriser pleinement les développements. La plupart du temps, les processus définis et séquentiels encore très répandus – les processus en cascade – ne sont pas en mesure de reproduire de manière satisfaisante la complexité du développement logiciel.

Cet article identifie les raisons pour lesquelles le processus en cascade reste prédominant et présente une alternative pour que les projets aboutissent à un meilleur résultat avec une meilleure efficacité : le développement itératif et incrémentiel de logiciels – aussi appelé la méthode Scrum. Il s'agit d'une approche méthodologique qui accepte que le développement de logiciels ne suive pas une méthode définie, mais se base sur une démarche empirique. Cet article décrit les règles, la répartition des responsabilités et le déroulement d'un processus Scrum. No