Alexander Volland et al. (Hrsg.): Projektmanagement & Vorgehensmodelle 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 25

Weg vom unvollständigen Scrum! Hin zum vollständigeren „Scrum++“

Philipp Diebold[1], Sven Theobald[1], Anna Schmitt[1] und Cordula Schmidt[2]

Abstract: Agile soweit das Auge reicht! Scrum als dominierende Agile Methode. Für viele ist Scrum damit auch ein Entwicklungsprozess. Als Projektmanagement-Framework sagt Scrum jedoch wenig über das „Wie“ der Software-Entwicklung. Demnach stellt sich die Frage nach sinnvollen oder gängig genutzten Ergänzungen zu Scrum, speziell aus den Bereichen Entwicklung, Testen und Delivery. In diesem Beitrag beschreiben wir unser methodisches Vorgehen zur Erweiterung von Scrum um sinnvoll zusammenhängende Agile Praktiken und stellen den finalen Output „Scrum++“ vor.

Keywords: Scrum, Scrum-But, Agile Methoden, Agile Praktiken, Prozessbausteine, Prozessverbesserung, Prozessmanagement, Prozesseinführung, Ensemble, Orchester.

1 Einleitung & Motivation

Agile Entwicklung – Wer tut es nicht? Zu einem gewissen Grad ist fast jeder bereits agil. Scrum [ScS13] als die gängigste Agile Methode [Ve16] wird jedoch oftmals angepasst [Di15]. Abhängig von den individuellen Bedürfnissen werden entweder einzelne Agile Praktiken adaptiert oder die gesamte Methode um Einzelteile erweitert.

Doch es stellt sich häufig die Frage, warum ich noch mehr als Scrum nutzen muss. Selbst der Scrum-Guide [ScS13] sieht Scrum als Container für andere Methoden und Praktiken. Die Begründung ist einfach: Der Fokus von Scrum liegt klar auf dem Bereich des Projekt- und Anforderungsmanagements. Scrum sieht in der Retrospektive vor, unter dem Prinzip von „Inspect and Adapt“ Schwachstellen zu identifizieren, um dann den Prozess zu verbessern. Das reine Engineering, von der Konzeption über die Entwicklung bis hin zum Testen, wird daher zunächst vernachlässigt. Aufgrund dieser Sichtweise wird Scrum auch als unvollständige Software Engineering-Methode bezeichnet [Di16]. Nachdem Scrum dennoch die am weitesten verbreitete Methode ist [Ve16] [KK14], stellen sich die Fragen, (1) was Unternehmen in der Praxis unternehmen, um die Problematik der Unvollständigkeit von Scrum zu lösen, und (2) wie deren modifiziertes Scrum aussieht.

Aus diesem Grund haben wir herausgearbeitet, wie durch Ergänzung einzelner Praktiken eine erweiterte Methode „Scrum++“ entsteht, welche auch in verschiedensten Formen Anwendung in der Praxis findet. Hierfür wird in Kapitel 2 zunächst der aktuelle Stand der Wissenschaft und Technik, speziell die Unvollständigkeit von Scrum, diskutiert. Die Methodik, auf deren Grundlage Erweiterungselemente für nahezu jede (Agile) Methode definiert werden können, erläutern wir in Kapitel 3. Diese Methodik wird in Kapitel 4 am Beispiel von Scrum angewandt. Der Beitrag schließt mit einer Zusammenfassung, gefolgt

[1] Fraunhofer IESE, Process Engineering, Fraunhofer-Platz 1, 67663 Kaiserslautern, {vorname.nachname}@iese.fhg.de
[2] BLUECARAT AG, Albin-Köbis-Str. 4, 51147 Köln, cordula.schmidt@bluecarat.de

von einem Ausblick über die Möglichkeiten der Weiterverwendung und Weiterführung dieser Arbeit.

2 Stand der Wissenschaft und Technik

Studien [Ve16] [KK14] zeigen: Scrum ist die gängigste aller Agiler Methoden. Auch wird darauf eingegangen, wie mögliche Erweiterungen dieser Methode aussehen: Zum einen anhand der Kombination mit anderen Methoden, z.B. „ScrumBan“, einer Mischung aus Scrum und Kanban, und zum anderen durch die Erweiterung um einzelne Agile Praktiken, wie z.B. User Stories oder Unit Testing.

2.1 Gängige Kombinationen mit Scrum

Die verbreitetsten Methodenkombinationen, welche Scrum beinhalten, sind laut [Ve16] ScrumBan und Scrum+XP (eXteme Programming). Selbst Ken Schwaber, Mitentwickler von Scrum, sieht die Verknüpfung von Scrum und XP als sinnvoll, da „Scrum auf Management-Praktiken und XP auf Engineering-Praktiken fokussiert“ [Si02]. Somit werden die Schwachstellen beider Vorgehensweisen durch die jeweils andere adressiert.

Neben definierten Methodenkombinationen gibt es auch einzelne Agile Praktiken, welche häufig in Kombination mit Scrum verwendet werden. Die laut [Ve16] am häufigsten genannten – aber nicht zu Scrum gehörenden – Agilen Praktiken sind Release Planning, Kanban-Board, Product Roadmapping, Story Mapping, Unit Testing, Continuous Integration, Coding Standards, Refactoring, Test-Driven Development, Continuous Deployment und Pair Programming. Diese Liste weist bereits einen Fokus auf das reine Engineering mit den Phasen Codierung, Testen und Integration bzw. Deployment auf. Viele dieser Praktiken entstammen der Methode eXtreme Programming (XP) – einem Engineering Framework.

2.2 Unvollständigkeit von Scrum als Methode

Wie eben dargestellt spielen sich die gängigsten Ergänzungen von Scrum im Bereich des Engineerings ab. Dies passt zur oftmals gezogenen Schlussfolgerung, Scrum sei eine Projektmanagement-Methode [PQ11][Sc04], welche nichts oder zumindest nicht viel über das Engineering aussagt. Bei der Zerlegung von Scrum lassen sich viele seiner Praktiken in den Bereich der Planung und der Kontrolle[3] einordnen [Di16], während die Umsetzung vernachlässigt wird (siehe Tab. 1 in [Di16]). Ergänzend zeigt sich, dass die Summe aller einzelnen Scrum-Praktiken nicht zwingend alle in [SD16] genannten Verbesserungsziele adressiert. Wodurch wir zur, diesem Bericht zugrundeliegenden, Frage kommen:

Wie lassen sich diese Lücken von Scrum [ScS13] schließen?

[3] „Kontrolle“ bezieht sich hierbei nicht auf „Command & Control“ wie es in eher traditionellen Projekten oder Organisationen gelebt wird, sondern auf die Selbstkontrolle zur Koordination durch das Team.

3 Der Weg zu einer vollständigeren Methode am Beispiel von Scrum

Um Scrum also zu mehr Vollständigkeit zu verhelfen und Erweiterungsideen für dieses zu generieren, müssen sinnvolle Ergänzungen gefunden werden, um Unternehmen Verbesserungsmöglichkeiten vorzuschlagen. Hierfür wurden drei Schritte angewandt:

(1) Über ein Abhängigkeitsnetz einzelner Agiler Praktiken (Kapitel 3.1)

(2) wurden kleine Gruppen an Praktiken – sogenannte Ensembles – definiert (Kapitel 3.2)

(3) und diese ggf. wieder zu einer vollständigen[4] Agilen Methode – einem sogenannten Orchester – kombiniert (Kapitel 3.3).

3.1 Abhängigkeitsnetz

Werden Vor- und Nachbedingungen Agiler Praktiken betrachtet, so können diese Praktiken meist unabhängig von anderen Praktiken eingesetzt werden. Beispielsweise kann das „Daily Stand-Up“ auch ohne Anwendung anderer Praktiken durchgeführt werden. Es ist jedoch so, dass sich durch die Kombination verschiedener Praktiken Synergien ergeben. Deshalb werden gewisse Praktiken auch sehr häufig zusammen verwendet.

Um dem Aspekt der Kombination gerecht zu werden, haben wir ein Abhängigkeitsnetz zwischen allen Agilen Praktiken des Forschungsprojektes ProKoB[5] aufgebaut. In diesem existieren zwei Relationen: Zwei Praktiken (1) „harmonieren zusammen“ oder (2) „schließen sich gegenseitig aus“. Da sich im gesamten Netzwerk aber nur zwei Praktiken – Lessons Learned vs. Retrospektive – gegenseitig ausschließen, werden im Folgenden lediglich die positiven Zusammenhänge betrachtet.

Praktiken „harmonieren zusammen“, wenn es u.a. notwendige Abhängigkeiten bei deren Nutzung gibt; z.B. schreibe ich beim Test-Driven Development Unit Tests oder benötige einen Backlog als Voraussetzung zum Backlog Refinement. Weiterhin beinhaltet diese Beziehung auch, dass eine Praktik eine andere unterstützt: (a) Beim Refactoring sollte sich das Team mit Unit Tests gegen Fehler absichern; (b) Prototyping kann bei der Aufwandsschätzung helfen; (c) Brainstorming könnte in einer Retrospektive zum Einsatz kommen. Ebenfalls wurden ähnliche Praktiken miteinander verknüpft. Dadurch könnte das vorhandene Tech-Meeting auf andere Interessenfelder erweitert und so neue Communities of Practice gründet werden. Schließlich wurden auch Vorschläge einbezogen, in denen Praktiken einfach sinnvoll miteinander verwendet werden: Habe ich regelmäßige Planning Meeting, ist es von Vorteil, auch regelmäßig eine Produkt Demo/ Review zu halten.

Das Abhängigkeitsnetz bildet also einen Graphen, in welchem immer zwei Praktiken durch eine Kante verbunden sind (siehe Abb. 1). Diese Kanten sind aktuell ungerichtet, da die Beziehung in den meisten Fällen in beide Richtungen vorhanden ist. In unserem unter Mithilfe von mehreren Experten im Bereich Agiler Entwicklung aufgebauten Abhängigkeitsnetz sind 62 Praktiken enthalten, welche durch insgesamt 212 Kanten verbunden sind.

[4] nach unserer Kategorisierung [Di16]
[5] www.prokob.de

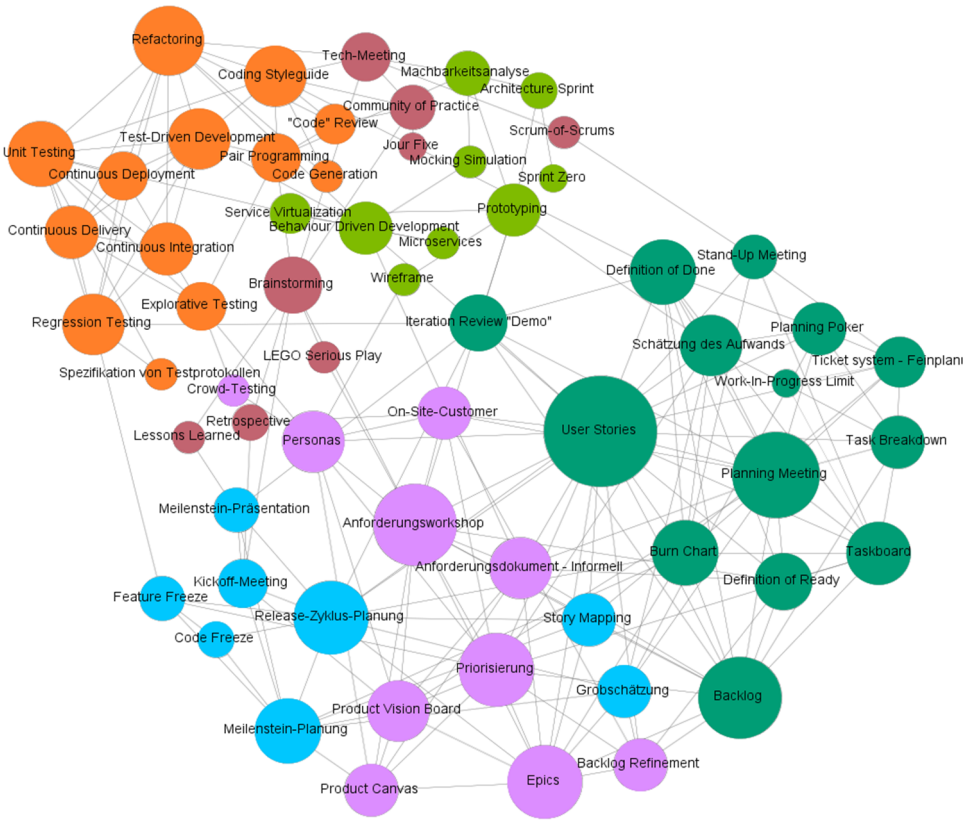

Abb. 1: Abhängigkeitsnetz mit durch einen Graphalgorithmus gruppierten und markierten Praktiken (Größe der Kreise = Anzahl anliegender Kanten; Farbe der Kreise = gebildete Cluster)

Zusätzlich zu unserem experten-basierten Abhängigkeitsnetz bietet die SwissAgile-Study [KMB16] Abhängigkeitsdaten zur gemeinsamen Verwendung von Agilen Praktiken [KK14]. Wir wollen die Daten dieser Studie verwenden, um damit unser Abhängigkeitsnetz zu validieren, auf eine breitere Datenbasis zu stellen und ggf. ergänzend anzureichern.

Auf Basis des Netzwerks bietet sich eine einfache Möglichkeit, geeignete Kombinationen an Agilen Praktiken zu bilden. Ein Graph visualisiert die Abhängigkeiten und unterstützt die Auswahl von Ensembles durch die semi-automatische Generation von Ensembles mit Hilfe von verschiedenen Graphalgorithmen. Abb. 1 zeigt beispielhaft einen Graphen, in dem mit einer geeigneten Parametrisierung sechs Gruppen als potentielle Ensembles vorgeschlagen und farblich markiert wurden.

3.2 Hierarchische Ensembles

Bei der Erstellung der Ensembles erschien es uns als nützlich, Ensembles nicht nur aus Praktiken, sondern auch aus anderen Ensembles zusammenzusetzen. Dies garantiert die nötige Flexibilität, um auch andere existierende Agile Methoden oder Frameworks durch diese Ensembles zu ergänzen. Somit ist ein Ensemble eine beliebige Kombination aus

einzelnen Agilen Praktiken und weiteren Ensembles, es kann also eine Hierarchie an Ensembles entstehen. Ein Ensemble muss mindestens drei Praktiken enthalten. Die Verbindung aus lediglich zwei Praktiken ist noch kein Ensemble, da sonst jede Verbindung im Abhängigkeitsnetz ein Ensemble ergeben würde.

Die Praktiken des orangenen Clusters (siehe Abb. 1) beschäftigen sich weniger mit Managementaufgaben, sondern fokussieren auf Engineering. Aus diesem Vorschlag heraus wurde das „Technische Ensemble“ abgeleitet, welches unter anderem die folgenden Teil-Ensembles enthält (siehe auch Abb. 2):

- „Test-Ensemble“ bestehend aus:
  - Regression Testing (RT)
  - Automatisiertes Testen (AT)
  - Test-Driven Development (TDD)
  - Unit Testing (UT)
  - Exploratives Testing (ET)
- „Entwicklungs-Ensemble“ bestehend aus:
  - Code Review (CR)
  - Coding Styleguide (CS)
  - Pair Programming (PP)
  - Test Driven Development
  - Refactoring

Wichtig hierbei: Es kann vorkommen, dass die beiden kombinierten Ensembles eine gleiche Praktik beinhalten, in unserem Fall TDD. Dies ist jedoch kein Problem, da das zusammengesetzte Ensemble die Menge aller Praktiken der Teil-Ensembles darstellt (analog zur mathematischen Vereinigung von Mengen), und TDD somit nur einmal enthält. Die Vereinigung der beiden Teil-Ensembles zum „Technischen Ensemble“ wird durch die einzelnen Praktiken „Continuous Integration“ und „Continuous Delivery“ erweitert, um eine Vollständigkeit über den kompletten Entwicklungszyklus zu erreichen. Diese Praktiken wurden nicht als eigenes Ensemble modelliert, da sie mit nur zwei Praktiken die Voraussetzung eines Ensembles nicht erfüllen. Zudem wird hiermit deutlich, wie Praktiken und Ensembles miteinander kombiniert werden können (siehe Abb. 2).

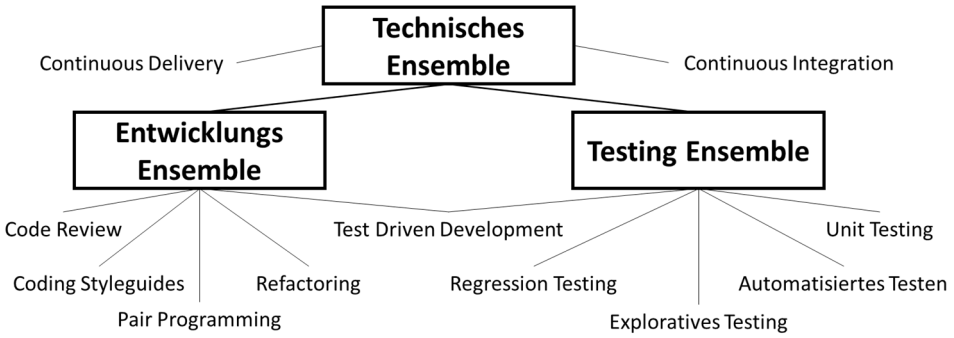

Abb. 2: Hierarchische Darstellung des „Technischen Ensembles“

3.3 Orchester

Die Vollständigkeit ist auch Mittelpunkt des Interesses bei der Überprüfung, ob ein Ensemble die Bedingungen eines Orchesters erfüllt. Sie ist über eine Matrix [Di16], bestehend aus den fünf PMI-Prozessgruppen [PMI13] und fünf abgeleiteten „Artefaktgruppen“ aus den fünf entwicklungs-spezifischen Phasen der ISO12207 [ISO08], zu überprüfen (siehe Abb. 3). Demnach ist ein Ensemble ein Orchester, wenn alle 25 Felder dieser sogenannten Prozessmatrix (5x5) abgedeckt sind. Dabei spielt es für unsere Definition eines Orchesters keine Rolle, ob die einzelnen Felder von einem oder mehreren Praktiken abgedeckt werden. Zum Beispiel sind die beiden oben genannten Ensembles keine Orchester, da ihre Praktiken die Prozessmatrix nicht komplett abdecken.

Wichtig ist zusätzlich, dass auch ein Orchester nicht zwingend ein kompletter Entwicklungsprozess sein muss, da ggf. Zellen mehrfach besetzt sein müssen, um alle Entwicklungsaufgaben zu erledigen. Als Beispiel würde es aus unserer Sicht nicht reichen, wenn die Zelle „Test x Umsetzung“ nur durch Regressionstests belegt ist, da weitere Testaktivitäten sinnvoll sind.

	Initiierung	Planung	Umsetzung	Controlling	Abschluss
Anforderung					
Entwurf					
Implementierung		TDD	TDD	TDD, UT, RT	
Test	TDD	TDD, ET, UT, RT	TDD, ET, UT, RT, AT	TDD, ET	TDD
Deployment					

	Initiierung	Planung	Umsetzung	Controlling	Abschluss
Anforderung					
Entwurf			PP, Refactoring	PP	
Implementierung	CS	CS, TDD	CS, CR, PP, TDD, Refactoring	CS, CR, PP, TDD	
Test	CS, TDD	CS, TDD	CS, CR, PP, TDD, Refactoring	CS, CR, PP, TDD	TDD
Deployment					

Abb. 3: Test-Ensemble (oben) und Entwicklungs-Ensemble (unten), dargestellt in der Matrix von [Di16] (Abkürzungen siehe Kapitel 3.2)

4 „Scrum++“

Da Scrum, die gängigste aller Agiler Methoden, nicht vollständig im Sinne der Prozessmatrix [Di16] ist, sollte mit Hilfe der in Kapitel 3 beschriebenen Methodik ein vollständigerer Prozess generiert werden. Das Ziel lautete daher: Die vollständige Abdeckung der

Prozessmatrix. Also eine Erstellung eines Orchesters, um mit einer neu definierten Agilen Methode den kompletten Entwicklungsprozess zu füllen. Beim Durchlaufen der Methodik lag der Fokus zuerst auf dem Entwicklungsprozess, um anschließend die Vollständigkeit nach unserer Orchester-Definition zu prüfen.

Da der Bereich Engineering in Scrum außen vor gelassen wird, wurde im Abhängigkeitsnetz speziell auf engineering-lastige Praktiken fokussiert, um Scrum mit diesen zu ergänzen. Die Visualisierung des Abhängigkeitsnetzes mit Hilfe eines gängigen Graphalgorithmus (Abb. 1) führte zu dem oben genannten „Technischen Ensemble“ (Kapitel 3.2). Zusätzlich zeigte das Abhängigkeitsnetz viele Verbindungen von Scrum-Praktiken zu gängigen Agilen Praktiken, welche ohnehin oftmals zusammen mit Scrum verwendet werden. Dies resultierte in einem „Anforderungs-Ensemble“, rund um die Verwendung von User Stories, und einem „Progress Tracking-Ensemble“, rund um die Verwendung von Taskboards und Burn-Down Charts. Zusätzlich wurde auf Basis unserer Erfahrungen noch das „Product Vision Board“ ergänzt, um die Projektinitiierung in Scrum zu unterstützen.

Diese Ergänzungen allein führen jedoch noch nicht zu einem Orchester, da speziell Deployment noch nicht vollständig abgedeckt wird. Aus diesem Grund wurden die Praktiken „Continuous Integration“ und „Continuous Delivery“ hinzugefügt, um die komplette Matrix abzudecken. Die daraus entstandene Agile Methode „Scrum++“ (Abb. 4) enthält somit final vier Teil-Ensembles und 30 Agile Praktiken: zehn davon entstammen dem initialen Ensemble Scrum, 20 sind neue AddOns.

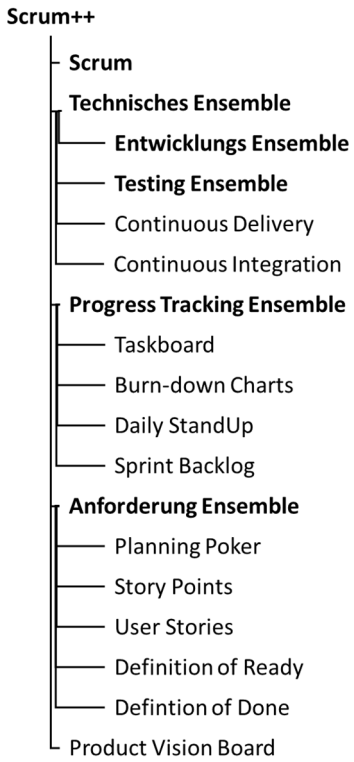

Abb. 4: Hierarchische Darstellung von „Scrum++“

Die Gesamtliste der 20 AddOn-Agilen Praktiken zeigen beim Vergleich mit den gängig genutzten Agilen Praktiken aus den Stand-der-Praxis-Umfragen [Ve16] [KK14], dass wir einen großen Teil dieser auch in unserem „Scrum++“ vorschlagen. Beispiele hierfür sind User Stories, Burndown Chart oder Pair Programming. Interessant ist, dass viele Manager laut [Ve16] auch Wert auf eine mittel- und langfristige Planung legen, da z.B. Praktiken wie Releaseplanung oder Produkt-Roadmap genutzt werden. Als Ergänzung zur Produkt-Roadmap und zum Anforderung-Ensemble wurde „Scrum++“ um das Product Vision Board ergänzt.

5 Zusammenfassung und Future Work

Wir präsentieren einen strukturierten Weg von der Agilen Methode Scrum mit Fokus auf Projektmanagement hin zu einem kompletten Entwicklungsprozess mit Hilfe von geeigneten Ergänzungen aus der Agilen Entwicklung. Sie sind keine voneinander losgelösten Ergänzungen, sondern zusammengehörige Mengen Agiler Praktiken, z.B. bzgl. Testen, Anforderungen oder Projektsteuerung. Somit besteht unsere neue Methode „Scrum++“ aus vier Teilensembles und insgesamt 30 Agilen Praktiken. Dennoch sollte „Scrum++“ nicht als neuer „Silver-Bullet“ gesehen werden, da es nicht zwingend alles abdeckt (z.B. Release Planung). Stattdessen zeigt es Ideen auf, wie Agile Methoden am Beispiel von Scrum um sinnvolle Gruppen von Praktiken erweitert werden können.

Als konkrete Folgearbeiten von „Scrum++“ ist aus unserer Sicht die Evaluierung der neuen Methode in der Praxis zu sehen. Außerdem gilt es, die zugrundeliegenden Daten des Abhängigkeitsnetzes zu validieren und zu ergänzen, z.B. durch [KMB16]. Letzteres dient dazu, die Auswahl bzw. Ergänzung für weitere Methoden zu verbessern. In Zukunft soll im Abhängigkeitsnetz zwischen den vorgestellten, verschiedenen Beziehungen unterschieden werden, z.B. ob eine Praktik die andere nur unterstützt oder eine notwendige Abhängigkeit besteht.

Aus unserer Sicht ist Scrum kein Allheilmittel. Neben diesem existiert noch eine Vielzahl anderer Agiler oder nicht-Agiler Methoden, die in der Praxis verwendet werden [Ve16] [KK14]. Unser Ziel ist es, auch die einzelnen vorgestellten und weiteren (Teil-)Ensembles, mit denen das originale Scrum ergänzt wurde, in die Praxis zu implementieren. Daher befindet sich die Veröffentlichung dieser Ensembles auf der Webseite unseres Forschungsprojektes ProKoB bereits in Arbeit. Des Weiteren ist vorstellbar, analog zu der in diesem Beitrag beschriebenen Arbeit andere Agile Methoden (XP, Kanban, etc.) und Skalierungsframeworks (SAFe, LeSS, etc.) zu analysieren und bei Bedarf um Praktiken oder Ensembles zu ergänzen.

Danksagung

Diese Arbeiten wurden im Rahmen des Forschungsprojektes ProKoB durch das Bundesministerium für Bildung und Forschung (BMBF 01IS15038) gefördert.

Literaturverzeichnis

[Di15] Diebold, P.; Ostberg, J.P.; Wagner, S.; Zendler, U.: What Do Practitioners Vary in Using Scrum?. In: Proceedings of 16th International Conference on Agile Software Development (XP2015), Helsinki 2015. Springer, Wiesbaden, S. 40-51, 2015.

[Di16] Diebold, P.; Zehler, T.; Schmitt, A.; Simon, F.; Kruser, B.: Prozessverbesserung durch fragmentierte Anwendung von Scrum & Co. In: Proceedings of Projektmanagement und Vorgehensmodelle 2016 (PVM'16). Paderborn 2016.

[ISO08] International Organization for Standardization: ISO/IEC 12207:2008 Systems and software engineering - Software life cycle processes. 2008.

[KK14] Komus, A.; Kuberg, M.: Status Quo Agile. Studie zu Verbreitung und Nutzen agiler Methoden, 2014.

[KMB16] Kropp, M.; Meier, A.; Biddle, R.: Agile Practices, Collaboration and Experiences. An Empirical Study about the Effect of Experience in Agile Software Development. In: Proceedings of 17th International Conference on Product-Focused Process Improvement (PROFES'16), Trondheim 2016. Springer, Wiesbaden, S. 416-431, 2016.

[PMI13] Project Management Institute: A Guide to the Project Management Body of Knowledge. 2013.

[PQ11] Pries, K.; Quigley, J.: Scrum Project Management. CRC Press, Taylor & Francis group, 2011.

[Sc04] Schwaber, K.: Agile Project Management with Scrum (Developer Best Practices). Microsoft Press 1/04, 2004.

[ScS13] Schwaber, K.; Sutherland, J.: The Scrum Guide. 2013.

[SD16] Schmitt, A.; Diebold, P.: Why do we do Software Process Improvement?. In: Proceedings of 17th International Conference on Product-Focused Process Improvement (PROFES'16), Trondheim 2016. Springer, Wiesbaden, S. 360-367, 2016.

[Si02] Silwa, C.: XP, Scrum Join Forces, http://www.computerworld.com/article/2586996/app-development/xp--scrum-join-forces.html, Stand: 2017

[Ve16] VersionOne: 10th Annual State of Agile Development Survey. 2016.