1 Einleitung

Roman Pichler · Stefan Roock

»Wenn ich Scrum einführe, begutachte ich die Entwicklungspraktiken. Manchmal sind sie in Ordnung, manchmal behindern sie das Team und manchmal sind sie nicht vorhanden. Ich benutze das Daily Scrum, um Defizite zu ermitteln. Dann arbeite ich mit dem Team und dem Management, um [die Praktiken] zu verbessern.« [Schwaber & Beedle 2002, S. 59]

1.1 Agile Softwarewicklung und Scrum

Scrum ist ein mächtiges und zugleich einfaches agiles Managementframework, das nicht vorschreibt, wie Software entwickelt wird. Der Fokus von Scrum liegt auf der Bereitstellung des richtigen Prozesses. Über den Einsatz der Entwicklungspraktiken entscheidet das Team. Denn dieses organisiert seine Arbeit selbst und bestimmt, wie hochpriore Product-Backlog-Einträge in Produktinkremente umgewandelt werden. Und das ist auch gut so. Schließlich kennt das Team seine Fähigkeiten am besten und ist für die Erreichung des Sprint-Ziels verantwortlich.[1]

Dennoch spielt die Entwicklung qualitativ hochwertiger Software in Scrum eine zentrale Rolle. Wie das oben stehende Zitat zeigt, weist bereits das erste Scrum-Buch *Agile Software Development with Scrum* [Schwaber & Beedle 2002] auf den Einsatz der richtigen Entwicklungspraktiken hin. Das hat einen guten Grund: Ohne die richtigen Praktiken ist das Team nicht in der Lage, zuverlässig das Sprint-Ziel zu erreichen und solide Produktinkremente zu entwickeln. Letztere bilden aber die Grundlage für die effektive Anwendung von empirischem Management: dem Schaffen von Klarheit über den Entwicklungsfortschritt durch das Vorführen der Software, der Reflexion des Erreichten (engl. *inspect*) und dem Ableiten von Verbesserungsmaßnahmen (engl. *adapt*). Ohne ein qualitativ hochwertiges Produktinkrement ist das Einholen von hilfreichem Feedback von Kunden, Anwendern und anderen Interessenvertretern nicht möglich, die Software

1. Wir haben uns bei den Rollennamen am *Scrum Guide*, *http://www.scrum.org/scrumguides*, orientiert. *Team* bezeichnet die Mitarbeiter, die Produktinkremente erstellen, *Scrum-Team* umfasst Team, ScrumMaster und Product Owner.

lässt sich nicht frühzeitig ausliefern, die Releaseplanung gerät zum Roulettespiel und neue Anforderungen können nur langsam und aufwendig umgesetzt werden.

Glücklicherweise haben sich verschiedene agile Entwicklungspraktiken im Scrum-Kontext als besonders hilfreich erwiesen. Hierzu zählen inkrementeller Entwurf, Continuous Integration, testgetriebene Entwicklung und Refactoring – Praktiken, die zum großen Teil durch eine andere agile Methode populär wurden: Extreme Programming (XP). Wie die meisten unserer Koautoren sind auch wir durch Extreme Programming zur agilen Softwareentwicklung gekommen. Bereits 1999 entdeckten wir agile Entwicklungspraktiken wie Collective Ownership, Pair Programming und testgetriebene Entwicklung. Scrum haben wir erst danach schätzen und lieben gelernt. Teams, die heute agil arbeiten möchten, starten meist mit Scrum und kennen agile Entwicklungspraktiken oftmals nicht. Dabei muss nicht jeder Entwickler ein Fan sämtlicher agiler Entwicklungspraktiken sein. Auch muss nicht jedes Scrum-Projekt alle Praktiken einsetzen. Agile Entwicklungspraktiken sollten aber zum Handwerkszeug eines Scrum-Entwicklers gehören. Denn nur so ist das Team in der Lage, über den Einsatz der richtigen Praktiken zu entscheiden.

Softwareentwickler im Scrum-Team sollten die gängigen agilen Entwicklungspraktiken nicht nur gut beherrschen, sondern auch verantwortlich einsetzen. Dazu zählt, selbst bei Zeitnot auf die Arbeit stolz sein zu können und qualitativ hochwertige Software zu erstellen – ohne dabei den Geschäftswert aus den Augen zu verlieren. Diese Kombination aus Berufsehre, Handwerkszeug und Wertorientierung nennt Robert Martin *Craftsmanship* [Martin 2008]. Kent Beck spricht von *Responsible Development* [Christensen 2007] und fragt das Team: »Würdet Ihr Euer eigenes Geld in dieses Team investieren?«

1.2 Zielgruppe und Zielsetzung

Unser Buch richtet sich an Softwareentwickler, Softwarearchitekten, Programmierer, Tester, ScrumMaster und Entwicklungsleiter. Wir setzen voraus, dass der Leser über Softwareentwicklungserfahrung verfügt und Grundkenntnisse in Scrum besitzt. Für eine Einführung in Scrum empfehlen wir die Lektüre von [Schwaber & Beedle 2002] oder [Pichler 2008].

Unser Ziel ist es, dem Leser die gängigen agilen Entwicklungspraktiken zusammen mit ihrem Einsatz in Scrum zu vermitteln. Natürlich ist das Ausprobieren und Anwenden der Praktiken unerlässlich, um diese kompetent handhaben zu können. Denn für agile Entwicklungspraktiken gilt »Probieren geht über Studieren« und »Übung macht den Meister«.

Das Buch dient außerdem zur Vorbereitung auf eine Zertifizierung wie Certified Scrum Developer (CSD) und Professional Scrum Developer (PSD).

1.3 Überblick über das Buch

Keine Entwicklung beginnt mit der ersten Sprint-Planungssitzung. Kapitel 2 führt in das Konzept der *Architekturvision* ein und klärt, welche Anteile der Architektur vor Entwicklungsbeginn fixiert werden müssen und welche mithilfe der Techniken dieses Buches während der Entwicklung entstehen sollten.

In Kapitel 3 werden die Herausforderungen dargestellt, mit denen das Team, bedingt durch Scrums iterative-inkrementelle Vorgehensweise, konfrontiert wird: der *inkrementelle Entwurf*. Der größte Teil des Systems samt seiner Architektur, seinem Design, dem Code und den Tests muss inkrementell entworfen und entwickelt werden. Das Kapitel führt in die Qualitätskriterien und Entwurfsprinzipien ein, denen inkrementelle Entwürfe genügen müssen.

Kontinuierliche Integration (engl. *Continuous Integration*) verfolgt den Ansatz, Änderungen am System in kleinen Schritten vorzunehmen und diese Änderungen jeweils sofort in den gemeinsamen Quellcodebestand zu integrieren. Wie klein die Schritte sein können und wie man diese Praktik technisch umsetzt, zeigt Kapitel 4.

Kapitel 5 widmet sich der Entwicklungspraktik, die oft als die wichtigste agile Praktik bezeichnet wird: *testgetriebene Entwicklung* (Test-Driven Development, TDD). In diesem Kapitel wird nicht nur beschrieben, dass die Tests vor dem Code entwickelt werden, sondern auch wie in kleinen Schritten der Entwurf durch Tests vorangetrieben wird.

Testgetriebene Entwicklung ohne Refactoring ist undenkbar. Trotzdem haben wir uns entschieden, zwei eigene Kapitel zu Refactoring zu spendieren. Schließlich kann Refactoring auch ohne testgetriebene Entwicklung sinnvoll verwendet werden, um existierenden Code aufzuräumen. Kapitel 6 führt in die Thematik ein und erläutert, wie Refactorings manuell durchgeführt werden können. Automatisierte Refactorings sind Gegenstand von Kapitel 7.

Testgetriebene Entwicklung fokussiert die Entwurfseinheiten der Entwickler – in objektorientierten Programmiersprachen sind dies die Klassen. *Akzeptanztests* sind End-to-End-Tests des integrierten Systems aus Anwendersicht. Die heute verfügbaren Technologien erlauben es, Akzeptanztests mit vertretbarem Aufwand zu automatisierten und dabei ebenfalls testgetrieben vorzugehen. Kapitel 8 zeigt, wie das geht.

Als Extreme Programming um das Jahr 2000 herum bekannt wurde, wurde es häufig auf eine besonders auffällige Technik verkürzt: das Programmieren in Paaren (engl. *Pair Programming*). *Pair Programming* und *Collective Ownership* (der Code gehört allen) stellen mächtige Werkzeuge dar, die für schnellen Wissensaustausch zwischen den Entwicklern sorgen und Bottleneck-Situationen im Team vermeiden. Kapitel 9 erklärt, wie die beiden Techniken funktionieren.

Code-Katas und Coding-Dojos klingen zwar nach asiatischem Kampfsport, helfen dem Team aber, agile Entwicklungspraktiken systematisch einzuüben und effektiver programmieren zu lernen – ohne dabei Fauststöße ausführen zu müs-

sen oder sich womöglich eine blutige Nase zu holen. Mehr hierzu finden Sie in Kapitel 10.

Kapitel 11 widmet sich der Frage, wie sich modellgetriebene Softwareentwicklung (Model Driven Software Development, MDSD) zu Scrum verhält. Welche Entwicklungspraktiken müssen ggf. angepasst werden? Werden vielleicht sogar Entwicklungspraktiken obsolet?

In den letzten Jahren ist ein Trend unübersehbar: Immer mehr Scrum-Teams arbeiten verteilt. Dabei kann Off- oder Nearshoring eine Rolle spielen. Es wird aber auch häufig innerhalb eines Landes verteilt entwickelt. Kapitel 12 stellt dar, welche Entwicklungspraktiken vor dem Hintergrund verteilter Teams in einem anderen Licht erscheinen und wie damit umzugehen ist.

1.4 Java als Beispielsprache

Wir haben uns aufgrund ihrer weiten Verbreitung dazu entschieden, alle Codebeispiele in diesem Buch in Java zu schreiben. Die Beispiele sollten sich aber auf andere Programmiersprachen übertragen lassen. Lediglich in Kapitel 7 »Automatisierte Refactorings« mag dies aufgrund der Möglichkeiten der verwendeten Entwicklungswerkzeuge nicht immer gelingen.

1.5 Danke

Ohne die Zusammenarbeit vieler Experten wäre dieses Buch niemals möglich gewesen. Unser Dank gilt daher den Autoren dieses Buches für ihren Einsatz und das geduldige Eingehen auf unser Feedback zu ihren Texten.

Wir möchten uns außerdem beim dpunkt.verlag und unserer Lektorin Christa Preisendanz für die Unterstützung in allen organisatorischen Belangen bedanken.

Ganz herzlich bedanken wir uns bei Markus Gärtner, Heiko Hahn und Manuel Küblböck, die das komplette Buch einem Review unterzogen und wertvolles Feedback geliefert haben.

Vielen Dank auch an Markus Andrezak für das Schreiben des Geleitworts.

Softwareentwicklung hat für uns auch eine ethische Komponente. Das Autorenhonorar dieses Buches spenden wir daher an die Wohltätigkeitsorganisation Oxfam.