Qualitätsbasierte Bewertung Agiler Entwicklungsmethoden mit dem AMMI

André Janus
André Janus – IT Consulting / Universität Magdeburg
Karlsruhe / Magdeburg
mail@andre-janus.de

Zusammenfassung

Qualitätssicherung nimmt in der Agilen Software-Entwicklung auf den ersten Blick keine große Rolle ein. Dennoch ist Qualität ein grundlegendes Prinzip von Agilen Entwicklungsmethoden, das implizit von vielen Agilen Praktiken umgesetzt wird. Dieser Beitrag stellt die Grundlagen Agiler Software-Entwicklung dar und gibt einen Überblick über zentrale Agile Praktiken. Schließlich wird das Agile Maturity Model Integration (AMMI) als Reifegradmodell für Agile Software-Entwicklungsmethoden vorgestellt. Außerdem werden die damit zusammenhängenden offenen Fragen und Herausforderungen für die Forschung beleuchtet, die im Ergebnis eine Bewertung von Agilen Software-Entwicklungsmethoden ermöglichen.

Schlüsselwörter

Empirische Fallstudien, Produkt- und Prozessmetriken, Best Practices, Agilität, Qualitätssicherung

1. Motivation und Ziel

Es ist eine immer größer werdende Verbreitung und Beliebtheit von Agiler Software-Entwicklung zu beobachten. Die ebenfalls größer werdende Anzahl erfolgreich mit Agilen Verfahren abgeschlossenen Projekte zeigt, dass es sich bei Agiler Software-Entwicklung nicht nur um einen "Hype" handelt.

Auf den ersten Blick scheint Agile Software-Entwicklung dem klassischen Ingenieurs-mäßigen Vorgehen in der Software-Entwicklung entgegen zu stehen, es scheint sogar Themen wie explizite Qualitätssicherung überflüssig zu machen, da es – so die Aussagen der Agile Community – hohe Qualität "automatisch" impliziert.

Dieser Beitrag zeigt, dass Agile Software-Entwicklung sehr wohl einem Ingenieurs-mäßigem Vorgehen entspricht und wirft einen genaueren Blick auf die Mechanismen, die zu der höheren Qualität der Software führen und diese sicherstellen sollen.

2. Grundlagen der Agilen Software-Entwicklung

Die Grundlage für Agile Software-Entwicklung bildet das Agile Manifest [1] mit seinen Werten und Prinzipien. Daraus abgeleitet stellen die einzelnen Agilen Software-Entwicklungsmethoden konkrete Praktiken zur Verfügung, die diesen Werten und Prinzipien folgend Aufgaben und Prozess für die Software-Entwicklung definieren.

2.1 Das Agile Manifest

Die Basis für Agile Software-Entwicklungsmethoden wie eXtreme Programming (XP) [2] oder Scrum [3] ist das Agile Manifest. Das Agile Manifest stellt vier Werte heraus, die das Fundament der Agilen Software-Entwicklung bilden:

- Individuen und Interaktionen gelten mehr als Prozesse und Tools.
- Funktionierende Programme gelten mehr als ausführliche Dokumentation.
- Die stetige Zusammenarbeit mit dem Kunden steht über Verträgen.
- Der Mut und die Offenheit für Änderungen stehen über dem Befolgen eines festgelegten Plans.

Zusätzlich nennt das Agile Manifest zwölf Prinzipien, denen die Software-Entwicklung zugrunde liegen soll, die etwas konkretere Leitsätze für die Software-Entwicklung bieten, aber wie die Werte recht abstrakt bleiben. Die konkreten Praktiken werden von den jeweiligen Agilen Methoden bestimmt (**Abbildung 1**).

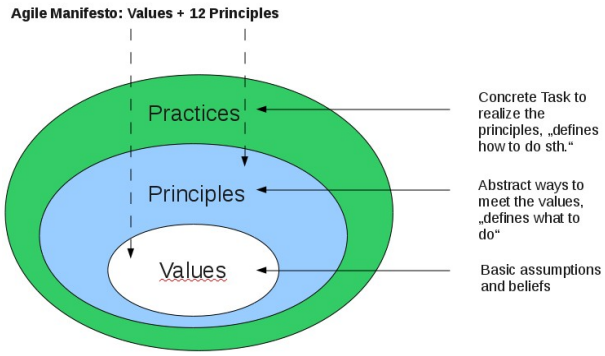

Abbildung 1: Werte, Prinzipien und Praktiken

Gemeinsam ist Agilen Software-Entwicklungsmethoden der iterative und inkrementelle Entwicklungs-Prozess. Die typischen Phasen der Software-Entwicklung werden mehrfach durchlaufen (iterativ) und die Software in jedem Durchlauf um weitere Funktionen erweitert (inkrementell). Das Ergebnis einer Iteration ist ein Software-Inkrement, das prinzipiell für den produktiven Einsatz geeignet sein muss. Ob tatsächlich aus dem Software-Inkrement ein Release, also eine Produktivversion wird, ist nicht festgelegt. In der Regel werden größere Releasepakete aus mehreren Iterationen zusammengefaßt.

Eine weitere Gemeinsamkeit ist, dass Software-Entwicklungsmethoden leichtgewichtig sind. Dies bedeutet, dass außer der Software selbst wenige zusätzliche Artefakte wie Spezifikationen oder

Dokumentationen entstehen. Im Gegensatz dazu bilden schwergewichtige Prozesse, die z. B. auf dem Wasserfallmodell basieren, die komplette Funktionalität der Software in den frühen Lebenszyklus-Phasen in umfangreichen Dokumenten wie Lastenheften, Feinspezifikationen etc. ab. Die Agile Software-Entwicklung geht davon aus, dass sich die Anforderungen während eines Projekts ändern und bietet mit den sich wiederholenden Iterationen eine Möglichkeit auf diese Veränderungen zu reagieren. Durch Feedback am Ende jeder Iteration kann das Vorgehen in der folgenden Iteration angepaßt werden.

2.2 Praktiken der Agilen Software-Entwicklung

Eine Grundlage von Agilen Praktiken ist das schnelle Feedback sowie das Lernen aus diesem Feedback. Fehler oder Verbesserungspotential sollen möglichst früh und damit möglichst kostengünstig erkannt und behoben bzw. genutzt werden. So kann sich das Team mit jeder Iteration verbessern, wobei sowohl Änderungen an den Praktiken selbst, sowie am Entwicklungsprozess möglich sind, um Verbesserungen zu erreichen.

Agile Software-Entwicklungsmethoden formen aus den eher abstrakten Werten und Prinzipien konkrete Praktiken für den Software-Entwicklungsprozess (**Abbildung 2**). Hier werden beispielhaft die Praktiken des eXtreme Programming (XP) betrachtet. eXtreme Programming (XP) ist eine der ältesten Agilen Software-Entwicklungsmethoden und Vorbild für viele daraus entwickelte Varianten. Außerdem ist XP die Basis für das im Industrieprojekt verwendete Vorgehen gewesen, aus dem die Umfragewerte und Einschätzungen stammen.

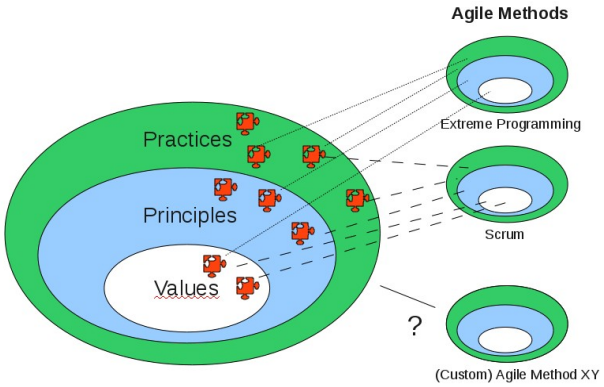

Abbildung 2: Werte, Prinzipen und Praktiken in Agilen Methoden

2.3 Qualitässichernde Aspekte der Agilen Praktiken

Die Agilen Praktiken bestimmen durch ihre Vorgaben an die Software-Entwicklung implizit auch ihre Qualität. Die einzelnen Praktiken berühren dabei sowohl die interne, für den Entwickler relevante Qualität, als auch die vom Kunden und Anwender wahrgenommene externe Qualität. Sie decken die Dimensionen Produkt-, Prozess- und Ressourcenqualität ab [4]. Hinsichtlich dieser Aspekte können die o. g. Praktiken des eXtreme Programming (XP) wie in **Abbildung 3** zu sehen eingeordnet werden.

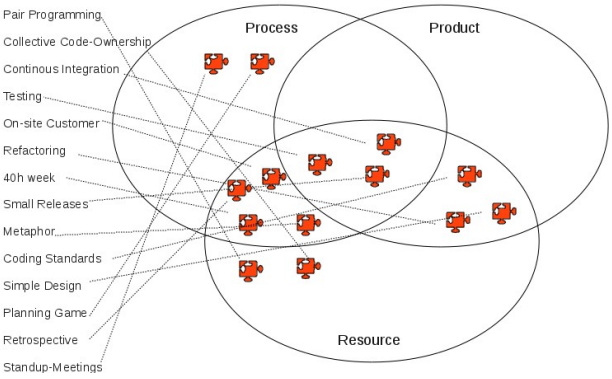

Abbildung 3: Zuordnung Agiler Praktiken zu Qualitäts-Dimensionen

3. Das Agile Maturity Model Integration (AMMI)

Aufgrund der hohen Popularität Agiler Software-Entwicklung bezeichnen sich viele Projekte selbst oder ihre verwendeten Software-Entwicklungsprozesse als "Agil". Dies wird dabei als "Qualitäts-Zeichen" zu Marketing-Zwecken verstanden, ohne dass es klare Kriterien gibt, wann ein Software-Entwicklungsprozess als Agil gelten kann. Bei genauerem Hinsehen wird oft klar, dass weder die Werte, noch die Prinzipien oder die Grundidee der Agilen Software-Entwicklung in vielen dieser Projekte oder Prozesse verwirklicht sind. Es gibt keine Bewertungsmethode, die die Agilität eines Software-Entwicklungsprozesses misst oder die Frage beantwortet, wie ein Software-Entwicklungsprozess agiler werden kann.

Zusätzlich stellt sich die Frage, welche Auswirkung die Einführung Agiler Software-Entwicklung bzw. einzelner Praktiken tatsächlich auf die Software-Qualität [5] hat. Es ist zu entscheiden, welche Agilen Praktiken eingesetzt werden sollen. Doch es gibt weder eine Entscheidungshilfe oder ein Bewertungsschema, welches die wichtigsten Agilen Praktiken sind oder welche Praktiken zwingend erforderlich sind oder welche weiteren kritischen Faktoren es gibt.

Vor diesem Hintergrund wurde das Agile Maturity Model Integration (AMMI) entwickelt **(Abbildung 4)**, das Entwicklungprozesse nach "Agility Levels" bewertet.

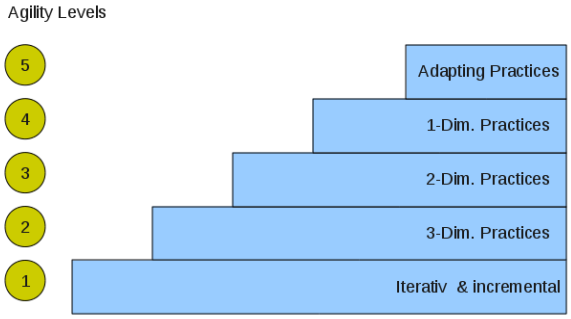

Abbildung 4: Agile Maturity Model Integration (AMMI)

Ähnlich dem Reifegradmodell CMMI (Capability Maturity Model Integration) [6] sind Level von 1 bis 5

möglich, wobei 5 die höchst mögliche Reife eines Software-Entwicklungsprozesses darstellt. In den folgenden Kapiteln werden die Anforderungen an die einzelnen Reifegrade genauer erläutert.

3.1 Level 1: iterative & incremental

Das Agility Level 1 (**Abbildung 5**) beschreibt einen Agilen Basis-Prozess mit einem iterativen Vorgehen, das es erlaubt (schnelles) Feedback zu bekommen und dieses umzusetzen. Außerdem ist der Prozess von einem inkrementellen Vorgehen gekennzeichnet, d.h. dass die Software mit jeder Iteration um ein Inkrement wächst. Planung und Dokumentation werden leichtgewichtig gehalten, d.h. es wird ad hoc für die nächste Iteration geplant und nur minimal dokumentiert.

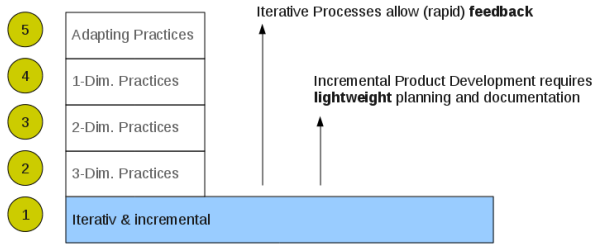

Abbildung 5: Level 1

3.2 Level 2: 3-Dim. Practices

Das Agility Level 2 (**Abbildung 6**) beschreibt einen Prozess, der zusätzlich zu den Anforderungen aus Level 1 eine ausreichende Anzahl Agiler Praktiken einsetzt, gleichzeitig die Qualität der Dimensionen Prozess, Produkt und Ressourcen sicherstellt.

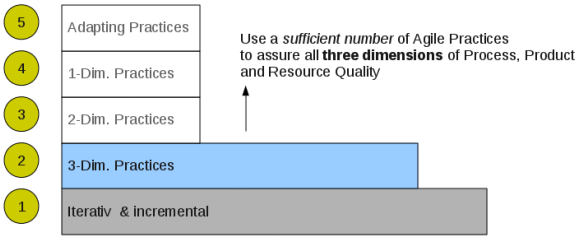

Abbildung 6: Level 2

3.3 Level 3: 2-Dim. Practices

Um Agility Level 3 (**Abbildung 7**) zu erreichen muss ein Prozess zusätzlich zu den Praktiken aus Level 2 eine Mindestmenge von Praktiken einsetzen, die zumindest zwei der drei Qualitätsdimensionen betreffen.

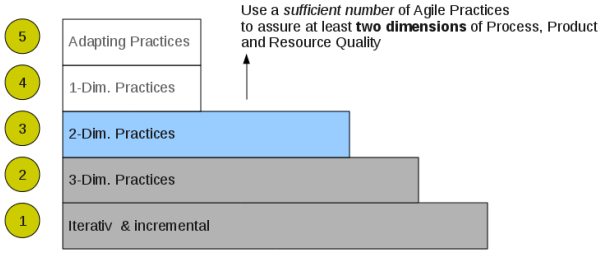

Abbildung 7: Level 3

3.4 Level 4: 1-Dim. Practices

Das Agility Level 4 (**Abbildung 8**) erfordert zusätzlich zu den Praktiken aus Level 3 eine Mindestmenge von Praktiken, die einer Qualitätsdimension zuzuordnen sind.

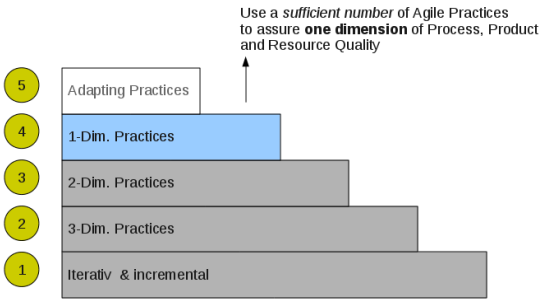

Abbildung 8: Level 4

3.5 Level 5: Adapting Practices

Um Agility Level 5 (**Abbildung 9**) zu erreichen muss ein Software-Entwicklungsprozess zusätzlich zu den Anforderungen aus Level 4 die Agilen Praktiken selbst anpassen bzw. neue Praktiken für das eigene Vorgehen adaptieren.

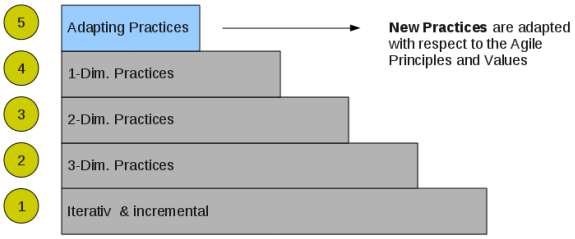

Abbildung 9: Level 5

4. Herausforderungen und offene Fragen

Es bleibt zu klären, wie die Einhaltung der Agilen Werte und Prinzipien über die reine Benutzung Agiler Praktiken gesichert werden kann. Zwar sind die Agilen Praktiken an sich schon Verwirklichungen der Agilen Werte und Prinzipien, aber dennoch kann der Software-Entwicklungsprozess als ganzes oder in Teilen nicht Agil sein, in dem Praktiken oder Vorgehensweisen verwendet werden, die den Agilen Werten und Prinzipien entgegenstehen.

Der in Level 1 geforderte iterative und inkrementelle, sowie leichtgewichtige Software-Entwicklungsprozess stellt eine Grundlage der Agilen Software-Entwicklung dar. Die genauen Auswirkungen dieser Vorgehensweise auf die Software-Qualität sind aber noch nicht vollständig verstanden worden. Bezüglich der Agility Levels 2 bis 4 gilt zu klären, welche Qualitäts-Dimensionen einzelnen Agilen Praktiken zuzuordnen sind und dies schlüssig zu begründen. Hier sind vor allem neuere Agile Praktiken z. B. aus der überarbeiteten Version des eXtreme Programming oder Scrum zu untersuchen.

Hier gilt es den tatsächlichen Einfluß auf die Software-Qualität zu messen, die ggf. von der subjektiven Einschätzungen der Projekt-Beteiligten abweicht.

In den Agility Levels 2 bis 4 wird eine "ausreichende Anzahl" Agiler Praktiken gefordert. Die geschätzte Anzahl von ca. 3 bis 5 Praktiken sollte empirisch belegt werden.

5. Fazit und Zusammenfassung

Qualität ist ein inhärenter Bestandteil von Agiler Software-Entwicklung. Dies wird sowohl in den Werten und Prinzipien ausgedrückt als auch in den Agilen Praktiken direkt umgesetzt.

Um einen (Agilen) Software-Entwicklungsprozess als Ganzes und auch seine einzelnen Vorgehensweisen beurteilen zu können, ordnet das Agile Maturity Model Integration (AMMI) einen Software-Entwicklungsprozess verschiedenen Agility Levels von 1 bis 5 zu, die die Reife des Prozesses in Hinblick auf (qualitätssichernde) Agilität bewertet.

Wie auch CMMI garantiert AMMI keine erfolgreichen Projekte oder hoch-qualitative Software von sich aus, aber es sichert dem verwendeten Software-Entwicklungsprozess notwendige Eigenschaften zu, die den Rahmen geben und damit helfen hoch-qualitative Software zu entwicklen und Projekte erfolgreich abzuschließen.

Referenzen

[1] Beck, K. et al.: Manifesto for Agile Software Development, 1999, http://www.agilemanifesto.org/

[2] Beck, K.: eXtreme Programming explained - Embrace Change, 1st Edition 1999, Addison-Wesley Professional

[3] Schwaber, K.; Beedle, M.: Agile Software Development with Scrum, Pearson International Edition, Pearson Education, 2002

[4] Dumke, R.; Rombach, D.: Software-Messung und -Bewertung, 2002, Deutscher Universitätsverlag, Wiesbaden

[5] Dumke, R.: Software Engineering - Systeme, Erfahrungen, Methoden, Tools, 2001 Vieweg-Verlag, Braunschweig-Wiesbaden

[6] Software Engineering Institute: Capability Maturiyt Model Integration (CMMI), http://www.sei.cmu.edu/cmmi/