Universität Hamburg
Fachbereich Informatik

Seminar „Softwareentwicklung in der Wissenschaft“

Hausarbeit

Agile Softwareentwicklung

Wie entwickelt man agil und wann ist es überhaupt sinnvoll?

vorgelegt von

Roberto Seidel
Matrikelnummer: 6537468
Studiengang: Wirtschaftsinformatik

eingereicht am 15. September 2015

Betreuer: Dr. Julian Kunkel

Inhaltsverzeichnis

1 Einleitung
1.1 Motivation

Schon 1956 wurde von Herbert D. Benington in [BENIN1956] unter Bezug auf Jay W. Forrester festgestellt, dass die Bedeutung von Computersystemen für die Steuerung öffentlicher Infrastruktur und für die Wirtschaft zunimmt und daraus die Wichtigkeit erfolgreicher Softwareentwicklung abgeleitet, die als Ergebnis verlässliche und nützliche Software liefern müsse. Aus diesem Grund werden Vorgehensmodelle verwendet, die die Einhaltung der Kundenvorgaben, die weitgehende Fehlerfreiheit der Software, sowie teilweise die spätere Wartbarkeit durch Dritte sicherstellen sollten. Agile Vorgehensmodelle wollen das auf eine effiziente Weise sicherstellen, die den speziellen Anforderungen der Softwareentwicklung angepasst ist.

Diese Arbeit behandelt dabei neben den Grundlagen zwei Modelle der agilen Softwareentwicklung – Scrum und Extreme Programming („XP“). Dabei wird gezeigt, welche Ideen agilen Vorgehensweisen zugrunde liegen, auf welche Probleme sie versuchen, Antworten zu geben und wie effektiv und sinnvoll die Anwendung agiler Vorgehensweisen in der Softwareentwicklung ist.

1.2 Zu bewältigende Probleme der Softwareentwicklung

Die Entwicklung neuer Software stellt in vielen Fällen eine komplexe Aufgabe[1] dar, da es für die erfolgreiche Bewältigung dieser keine einfachen Regeln gibt. Nur für einzelne Bestandteile eines Softwareentwicklungsprojekts gibt es erprobte Muster und Hilfsmittel, die die Erfolgswahrscheinlichkeit erhöhen und den Aufwand minimieren können. Die agilen Vorgehensmodelle versuchen, die Komplexität zu reduzieren und Lösungen anzubieten, um Probleme gar nicht erst auftreten zu lassen. Dabei greifen sie gerade für technische Probleme auch auf schon bekannte Muster zurück.

Unabhängig von (agilen und anderen) Vorgehensmodellen der Softwareentwicklung existieren als Muster für den Entwurf und die Implementierung selbst insbesondere Design Patterns, Idiome und Architekturmuster. Frameworks und Toolkits können dabei die Anwendung solcher Muster unterstützen oder eine abstraktere und einfachere Implementation ermöglichen. Trotzdem kann durch einen unvorteilhaften Entwicklungsprozess der Erfolg der Entwicklung verhindert werden, wenn beispielsweise bereits die zu implementierenden Entwürfe nicht den tatsächlichen Bedürfnissen angemessen sind oder Anforderungen vergessen werden. Außerdem liefern diese technischen Muster und Frameworks keine Lösungen für den Umgang mit sozialen Aspekten der Softwareentwicklung wie der effizienten Gestaltung der Teamarbeit bei mehreren Beteiligten oder der Vermeidung von opportunistischem Handel.

Um den Entwicklungsprozess insgesamt und speziell auch die sozialen Komponenten effizient zu steuern, ist ein gutes Projektmanagement notwendig. Vor allem hier setzen agile Vorgehensmodelle an. Durch die Kombination von technischen Mustern in der eigentlichen Entwicklung und Vorgehensmodellen im Projektmanagement soll so die Erfolgswahrscheinlichkeit für Softwareentwicklungsprojekte gesteigert werden können. Auch das Projektmanagement kann dabei speziell dafür entwickelten Mustern folgen und Vorgehensmodelle zur Projektplanung und -überwachung nutzen. Ältere Modelle, wie das Wasserfall- oder das ursprüngliche V-Modell,

[1] nach der Darstellung des Komplexitätsbegriff durch Schoeneberg in [SCHOE2014, Kapitel 2] und Zimmermann in der Vorlesung „Integrierte Software- und Organisationsentwicklung“

verfolgen dazu eher lineare Vorgehensweisen. Neuere Ansätze wie das Spiralmodell und der Unified Process empfehlen hingegen ein iteratives Vorgehen. Gleiches gilt für die agilen Modelle.

Ein Komplexitätstreiber bei vielen Entwicklungs-Projekten ist ihr Umfang. Bei Programmen mit mehreren 1.000 Codezeilen sind Ursache-Wirkungsbeziehung einzelner Code-Abschnitte oder deren Änderungen nicht mehr offensichtlich. Daraus ergeben sich zudem verschiedene Probleme, wenn nicht jeder Entwickler den vollständigen Code kennt. Zum einen müssen einzelne Entwickler die Funktionalität anderer Programm-Teile verstehen, um sie für eigene Aufgaben nutzen zu können. Zum anderen führen Wissensmonopole bei einem Ausfall eines Entwicklers dazu, dass die übrigen oder die neuen Entwickler sich zunächst intensiver einarbeiten müssen. Auch dieser Problematik versuchen agile Vorgehensmodelle entgegenzuwirken, indem sie einen regelmäßigen Austausch fördern und häufig die gemeinsame Verantwortlichkeit aller Entwickler für den gesamten Code voraussetzen.

Abschließend können als weitere Gründe für die Komplexität der Softwareentwicklung die Auftraggeber und Anwender gesehen werden, weil sie ebenfalls eine soziale Komponente und teilweise wechselnde Ansprüche oder unklare Wünsche einbringen. Auch andere, beispielsweise gesetzliche, Rahmenbedingungen können zu Veränderungen der Vorgaben während es Projekts führen. Agile Modelle betrachten Veränderungen aber als willkommene Chancen und nutzen möglichst viele soziale Aspekte der Zusammenarbeit statt beides als Problem zu identifizieren.

2 Prinzipien agiler Softwareentwicklung

Als gemeinsame Grundlage der agilen Vorgehensmodelle zur Softwareentwicklung können das Agile Manifest und die dazu verfassten 12 Prinzipien gesehen werden.[2] Mit dem Agilen Manifest wurde eine Gewichtung von Vorgehensweisen und Bestandteilen des Entwicklungsprozesses vorgenommen, die die Grundwerte agiler Vorgehensmodelle bilden. Die Prinzipien dazu sind von den konkreten Vorgehensmodellen abstrahiert, werden aber in der Regel zumindest implizit umgesetzt. Sowohl das Manifest als auch die Prinzipien wurden 2001 auf einem Treffen von Vertretern der zu dem Zeitpunkt bekannten agilen Vorgehensmodelle ausgearbeitet, die nach [AGILH2001] auch die Bezeichnung „agil“ prägten. Dieses Kapitel stellt im Wesentlichen diese gemeinsame Grundlage der agilen Modelle dar und bewertet einige der wesentlichen Prinzipien.

Die Absicht der damals gegründeten Agile Alliance war es, eine Alternative zu „dokumentations-getriebenen, schwerfälligen Softwareentwicklungsprozessen“[3] zu bieten. Damit stellt agile Softwareentwicklung ein Gegenkonzept zu linearer Entwicklung und starren Vorgehensmodellen dar, die früher verwendet wurden.

2.1 Die Werte des Agilen Manifests

Als Kernidee identifiziert Siepermann in [GABLER01] die Einteilung der Aufgaben in „einfache und […] bewegliche“ Teilprozesse. In den Vorgehensmodellen ist dies umgesetzt, indem die Entwicklung inkrementell und iterativ durchgeführt wird und zusätzlich innerhalb einer Iteration sowie iterationsübergreifend eine Aufteilung auf verschiedene Aufgaben, Arbeitsinhalte oder Softwarefunktionen stattfindet. So soll auch spät in der Entwicklungsphase noch auf neue Anforderungen reagiert werden können, was die wesentliche Auswirkung agiler Vorgehensweisen

[2] abrufbar unter http://agilemanifesto.org/
[3] übersetztes Zitat aus [AGILH2001, Abs. 1]

ist. Entsprechend misst auch das Agile Manifest dem Reagieren auf Veränderungen einen höheren Wert zu als der Ausarbeitung eines Plans, der dann stur befolgt wird.

Eine weitere Abwägung im Manifest ordnet die Wichtigkeit von Individuen und deren Zusammenarbeit über der Bedeutung von Werkzeugen und festen Prozessen ein. Damit passen agile Vorgehensweisen zum Trend, Arbeitssysteme als soziotechnische Systeme zu betrachten und neben der technischen auch die soziale Teilkomponente zu nutzen.

Bezüglich der Arbeitsergebnisse messen agile Vorgehensweisen funktionierender Software einen höheren Wert zu als einer umfassenden Dokumentation. Der Hintergrund ist, dass umfangreiche Dokumentationen bei Veränderungen einen hohen zusätzlichen Wartungsaufwand zu ihrer laufenden Aktualisierung verursachen und so die Agilität einschränken, ohne dass sie in der Praxis häufig genutzt würden.

Aus einem ganz ähnlichen Grund empfehlen die Urheber des Manifests den Entwicklern, mit dem Kunden zusammenzuarbeiten statt nur auf Vertragsverhandlungen zu setzen. Wenn nämlich bloß bei Vertragsschluss ein Austausch stattfindet und alle Anforderungen an das Produkt endgültig festgeschrieben werden, lässt dies keinen Raum für Veränderungen und agiles Verhalten.

2.2 Prinzipien zur Umsetzung agiler Vorgehensweisen

Auffälligstes Ziel und nach [AGILP2001] auch das wichtigste Prinzip agiler Vorgehensweisen ist es, den „Kunden“ zufriedenzustellen und die entwickelte Software schneller einsetzen zu können. Der „Kunde“ muss dabei kein Außenstehender sein. Diesem Ziel lassen sich die übrigen 11 Prinzipien unterordnen. Obwohl dies von der Agile Alliance bei der Ausarbeitung nicht umgesetzt wurde, können die Prinzipien gut in die 3 Kategorien „Arbeitsorganisation“, „Vorgehen in der Entwicklung“ und „Strukturierung des Entwicklungsprozesses“ eingeteilt werden.

Die Prinzipien zur Arbeitsorganisation besagen, dass selbstorganisierte Teams die besten Ergebnisse erzeugen. Es sei aber notwendig, dass die Beteiligten motiviert sind, sie unterstützt werden und dass ihnen Vertrauen in ihre Arbeit entgegengebracht wird. So nehmen beispielsweise Schwaber und Sutherland an, dass die Entwickler selbst den Arbeitsaufwand besser einschätzen können und besser als Außenstehende (z.B. Vorgesetzte) wissen, wie sie ihr Ziel erreichen. Allerdings sollen bereits während der Entwicklung Entwickler und Fachexperten eng zusammenarbeiten. Dabei sei ein mündlicher, direkter Informationsaustausch zu bevorzugen. Verschiedene Studien – darunter [STUDY2004] von Abrahamsson – haben aber gezeigt, dass eine permanente Anwesenheit des Kunden oder der Anwender nicht notwendig ist. Die Empfehlung persönlicher Kommunikation jedoch ist anhand der Media Richness Theorie von Daft und Lengel nachvollziehbar, wenn man davon ausgeht, dass bei in Entwicklung befindlicher Software Informationen oder Anforderungen häufig nicht klar und eindeutig sind. Diese Annahme ist generell bei komplexen Aufgabenstellungen und der Zusammenarbeit von Menschen unterschiedlicher Fachgebiete naheliegend und außerdem richten sich agile Vorgehensmodelle insbesondere an Projekte, in denen die Anforderungen anfangs nicht klar sind.

Zur längerfristigen Strukturierung von Entwicklungsprojekten bezeichnet die Agile Alliance nicht etwa den Codeumfang oder die Dokumentation, sondern funktionierende Software als primäres Fortschrittsmaß. So wird nur real geschaffener Nutzen als Fortschritt gewertet, da zur Bemessung nur verwendbare Teile des Hauptprodukts, der Software, beachtet werden. Die Software sollte in möglichst kurzen, regelmäßigen Zyklen hergestellt und bei Bedarf auch herausgegeben werden. Bei der Arbeit ist den Prinzipien nach auch darauf zu achten, dass von Anfang an eine Arbeitsbelastung

und Geschwindigkeit gewählt wird, die alle Beteiligten dauerhaft halten können. Das zyklische und inkrementelle Vorgehen wird auch in vielen anderen, nicht-agilen Vorgehensmodellen angewendet.

Während der einzelnen Entwicklungsphasen und Iterationen sehen die Prinzipien zum agilen Manifest vor, dass das Entwicklungsteam regelmäßig über die bisherige Arbeit reflektiert und falls nötig sein Verhalten anpasst. So sollen direkt während der Entwicklung Erkenntnisse zum fachlichen Kontext und zur Arbeitsweise genutzt werden. Anforderungsänderungen des Kunden sollen nicht als Problem gesehen werden, sondern als Chance, den Vorteil der eigenen Agilität für sich selbst und zum Vorteil des Kunden zu nutzen. Dabei geht es nicht nur um die bereits erwähnte Konkretisierung von Anforderung, sondern gegebenenfalls auch darum, sich auf völlig neue oder veränderte Anforderungen einzulassen. Tatsächlich kommen späte Änderungen in der Praxis vor und die Anwender agiler Methoden berichten von verringerten Kosten im Vergleich zu Projekten mit klassischen Vorgehensmodellen, wie Rumpe und Schröder zum Beispiel für Extreme Programming in [SURVEY2002] zeigen. Bei der eigentlichen Entwicklungsarbeit fördere „technische Exzellenz und gutes Design“ die Agilität, wobei immer einfache Lösungen zu bevorzugen seien.

Ein Vorgehensmodell, das diese Werte, Ziele und Prinzipien ebenfalls verfolgt, kann in der Regel als agiles Vorgehensmodell bezeichnet werden.

3 Scrum

3.1 Basis und Ziele

Scrum ist ein konkretes Rahmenwerk, das insbesondere zur agilen Softwareentwicklung entwickelt wurde. Der Scrum Guide von Schwaber und Sutherland enthält die „offizielle“ Definition von Scrum und ist die Basis dieses Kapitels. Wie bei einigen Vorgehensmodellen bezeichnen auch die Entwickler von Scrum jeden Bestandteil von Scrum als „unentbehrlich“. In der Praxis können bei geübten Entwicklungs-Teams aber Komponenten vernachlässigt werden. Im Vergleich zu anderen Rahmenwerken ist Scrum sehr einfach verständlich und nicht besonders umfangreich, was die Einführung erleichtern kann. Unter den agilen Vorgehensmodellen ist es nach [HEISE2010] und [KOMUS2014] das verbreitetste.

Als wissenschaftliche Basis diente Takeuchis und Nonakas Paper „The New New Product Development Game“, in dem sie erklären, welche Arbeitsweise notwendig ist, um effizient neue, komplexe Produkte zu entwickeln. Sie empfehlen dazu empirische Prozesse statt detaillierte Vorausplanungen. Diese sollen von selbstorganisierten, kleinen Teams durchgeführt werden, die Zielvorgaben statt konkreter Aufgaben erhalten. Exakt dieser Empfehlung folgen Scrum und die Agile Alliance. Das Ziel von Scrum ist dabei, Risiken und Prognose-Unsicherheiten zu minimieren und den Wert der Software zu steigern. Um das zu erreichen setzt Scrum neben den selbstständigen, interdisziplinären Teams auf die drei „Säulen“ Transparenz, regelmäßige Überprüfung und Anpassung. Zudem gibt Scrum vor, dass ein einmal gesetzter Qualitätsanspruch nie gesenkt werden, sondern mit der Zeit steigen sollte.

3.2 Drei Säulen

Die Entwickler-Teams interdisziplinär aufzustellen ist eine Vorgabe, die ermöglichen soll, dass die Teams unabhängig von externen Personen arbeiten können. Das diene der Optimierung von Flexibilität, Kreativität und Produktivität. Sinnvoll ist die Abdeckung aller erforderlichen Kompetenzen aber auch vor dem Hintergrund, dass die Entwickler bei agiler Softwareentwicklung

eng zusammenarbeiten sollen und sowohl Arbeitsorganisation als auch Aufwandsschätzungen den Teams überlassen bleiben. Neben der fachlichen Interdisziplinarität fordert Scrum deswegen auch, die Entwicklungsteams mit den notwendigen Berechtigungen zur eigenständigen Arbeit auszustatten. Auch innerhalb der Entwicklungsteams soll es keine weitere Hierarchie oder fachliche Unterteilung sondern eine gemeinsame Verantwortung für die Umsetzung geben.

Das Transparenzziel beinhaltet in Scrum im Wesentlichen, dass alle relevanten Prozessaspekte jederzeit für die Personen sichtbar und verständlich sein müssen, die für ein Ergebnis verantwortlich sind. Gemeinsame Standards sollen zudem sicherstellen, dass alle Betrachter ein gemeinsames Verständnis über Absprachen, Notizen und Ziele haben.

Die regelmäßige Überprüfung des aktuellen Vorgehens und des aktuellen Entwicklungsstandes findet bei Scrum nicht nur durch Tests der Entwickler und Abnahmen der Kunden statt, sondern zusätzlich begleitend zur eigentlichen Entwicklungsarbeit durch „fähige Prüfer“ innerhalb des Teams. In Scrum wird so versucht, besonders zielorientiert vorzugehen. Das fördert auch „technische Exzellenz und gutes Design“, was nach dem Postulat in den Prinzipien agiler Softwareentwicklung zur Agilität beiträgt. Um diese nicht durch zu viele Überprüfungen wieder zu beeinträchtigen, wird durch Scrum die eigentliche Entwicklungsarbeit aber insofern priorisiert, als dass die Überprüfungen diese Arbeit nicht behindern sollten.

Ebenso wie Überprüfungen können auch Anpassungen des Vorgehens, der Aufgabenstellung oder des Codes jederzeit im Prozess auftreten, wenn eine Überprüfung vermuten lässt, dass die aktuelle Abweichung vom Ziel zu einem inakzeptablen Produkt führen würde oder wenn in einer Besprechung Hürden für das weitere Vorgehen ersichtlich werden.

3.3 Elemente, Teams & der Prozess in Scrum

Obwohl diese Vorgaben unter Umständen die Vorstellung eines sehr unbeweglichen, abgeschlossenen Prozesses wecken, setzt Scrum stattdessen nur auf viele formalisierte Elemente. Siepermanns Kernidee der einfachen und beweglichen Teilprozesse kann mit Scrum so sehr einfach realisiert werden. Als Elemente gibt Scrum zum einen Ereignisse vor, die den Projektablauf strukturieren, und stellt zum anderen mithilfe bestimmter „Artefakte“ eine gewisse Transparenz sicher. Eine maximale Dauer ist für alle Ereignisse vorgegeben. Eine Iteration – von Scrum als „Sprint“ bezeichnet – soll beispielsweise maximal einen Monat dauern. Zudem beschreiben drei unterschiedliche, fest definierte Rollen die Aufgaben der Beteiligten in einem Scrum-Team.

Ein *Scrum Master* ist weniger an der direkten Wertschöpfung beteiligt, sondern sorgt eher dafür, dass Scrum korrekt angewendet wird und alle anderen Beteiligten die Ideen und Elemente von Scrum verstehen. Dazu vermittelt er auch benötigte Techniken zur Organisation oder zu effektiver Teamarbeit, beseitigt Arbeits-Hindernisse des Teams und unterstützt die Durchführung der Scrum-Ereignisse. Vorgesehen ist auch, dass die Einführung von Scrum in Organisationen durch erfahrene Scrum Master durchgeführt wird. Außerdem ist der Scrum Master dafür zuständig, Personen außerhalb des Scrum-Teams die Arbeitsweise verständlich zu machen und gegebenenfalls Kommunikation zwischen dem Team und Außenstehenden effizient zu gestalten.

Im Gegensatz zum Scrum Master, der für die allgemeine Vorgehensweise verantwortlich ist, steuert der *Product Owner* die konkrete Arbeit des Teams. Er trägt die alleinige Verantwortung für das Product Backlog und soll den Nutzen der Entwicklungsarbeit und des Produkts steigern. Im Gegenzug ist er als einziger gegenüber dem Entwicklungsteam weisungsbefugt bezüglich der zu bearbeitenden Anforderungen.

Die Entwickler beziehungsweise das *Entwicklungsteam* erstellen letztlich die Produktinkremente. Da alle an der eigentlichen Umsetzung beteiligten Mitarbeiter diesem Team angehören, führt das Team neben Entwurfs- und Implementierungs-Arbeiten auch die das grafische Design etc. durch.

Zur Planung, Aufgabenverteilung und Fortschrittskontrolle gibt es für jedes Produkt genau ein „Product Backlog“. Es ist gewissermaßen eine nach Prioritäten geordnete ToDo-Liste und enthält sämtliche Anforderungen, Features, Verbesserungsideen und Fehlerbehebungen, die in das Produkt einfließen müssen oder könnten. Durch die Sortierung der Backlog-Einträge legt der Product Owner (oder dessen Beauftragter) die zukünftige Reihenfolge und Priorität der Aufgaben fest. Er ist auch dafür verantwortlich, dass das Backlog verständlich und transparent ist.

Neben einer Beschreibung der Aufgabe enthalten die Backlog Einträge auch eine Aufwands- und Werteinschätzung und eine Angabe, ob sie bereit zur Bearbeitung innerhalb einer Iteration ist. Diese Angeben werden während des Entwicklungsprozesses durch den Product Owner und das Entwicklungsteam regelmäßig verfeinert, erweitert und aktualisiert. Die endgültige Aufwandschätzung obliegt aber alleine demjenigen, der den Eintrag umsetzen wird.

„Sprint Backlogs“ gelten immer nur für die aktuelle Iteration und beinhalten die Einträge aus dem Product Backlog, die in der Iteration bearbeitet werden sollen.

Außer den Backlogs werden das „Inkrement“ und gelegentlich auch die „Definition of Done“ im Scrum Kontext als Artefakte bezeichnet. Das Inkrement ist eine Version des unfertigen Produkts, die nach jeder Iteration abgeschlossen sein sollte, also der Definition of Done entsprechen und einsatzfähig sein muss. Das jeweils aktuellste Inkrement beinhaltet auch alle vorherigen Inkremente. Die „Definition of Done“ ist dabei eine Übereinkunft aller am Produkt beteiligten Scrum Teams (bzw. mindestens eines vollständigen Teams), welche Kriterien erfüllt sein müssen, um eine Aufgabe als erledigt und letztlich das Inkrement als fertiggestellt zu bezeichnen.

Erst nachdem das Product Backlog angelegt wurde, kann im ersten „Sprint“ mit der Arbeit am Inkrement begonnen werden.

Durch die Aufteilung der Entwicklung in Sprints gibt es häufiger Gelegenheiten für Feedback und nach dem Scrum Guide beschränke das das Risiko auf die Kosten eines Sprints. Jeder Sprint beginnt dabei mit einem Sprint Planning in dem das Product Backlog verfeinert und das Sprint Backlog angelegt wird. Die im Sprint zu bearbeitenden Backlog Einträge wählt primär der Product Owner, das Entwicklungsteam legt aber die Anzahl und damit den Arbeitsumfang fest. Gemeinsam wird auch ein Sprint-Ziel formuliert, dass den Zweck des nächsten Inkrements beschreibt.

Während des Sprints werden in kurzen „Daily Scrums“ der bisherige Fortschritt überprüft, Tagesziele gesetzt und Hindernisse identifiziert. Es ist als Stand-Up Meeting gedacht, das nicht für Detailfragen sondern für schnelle Entscheidungen und die Fokussierung des Teams dient. Diskussionen zu Detailfragen sollen hingegen in informellen Treffen – teilweise direkt im Anschluss an das Daily Scrum – geklärt werden.

Zum Abschluss des Sprints wird das Inkrement in einem „Sprint Review“ vorgeführt. Dabei sollten alle Stakeholder anwesend sein, um Fragen und Rückmeldungen an das Team richten zu können. Bei Bedarf wird hier auch die Prognose zum Fertigstellungszeitpunkt korrigiert und das Product Backlog angepasst. Darauf folgt eine „Sprint Retrospektive“, an der nur das Scrum-Team teilnimmt. Sie dient der Überprüfung der eigenen Arbeit, wobei auch die konkrete Anwendung der Scrum Elemente besprochen wird. Verbesserungsmöglichkeiten für das Vorgehen werden ggf. hier direkt priorisiert

und ihre Umsetzung konkret geplant. Zusätzlich kann zu diesem Zeitpunkt die Definition of Done angepasst werden, um die künftige Produktqualität zu verbessern.

3.4 Beurteilung

Insgesamt betrachtet bietet Scrum keinen Automatismus für Agilität oder eine hohe Software-Qualität. Es enthält aber mehrere Maßnahmen und Elemente, die die Entwickler bei diesen Aspekten unterstützen können, wenn diese breit sind, sich selbst zu organisieren und im Prozess und den Scrum Ereignissen anfallende Erkenntnisse zu nutzen. Besonders stark setzt Scrum dazu auf Transparenz und direkte, regelmäßige Kommunikation. Die konkrete Umsetzung der vorgegebenen Elemente und Anforderungen wird aber den Entwicklern und den Organisationen überlassen, wodurch es letztlich trotzdem von deren Disziplin und Ideen zur Ausgestaltung, abhängt, ob Scrum wirklich zu einer Wertsteigerung der Produkte und einer höheren Agilität beiträgt. Grundsätzlich ermöglicht das Vorgehensmodell aber praktisch jederzeit Reaktionen auf Änderungen der Kundenanforderungen oder der Technologie, indem es ein iteratives Vorgehen vorgibt und Reflexion und Anpassung fördert. Dadurch und indem es den Entwicklungsprozess durch die verschiedenen Elemente strukturiert, wird die Komplexität der Softwareentwicklung zwar nicht vermieden oder reduziert, aber etwas beherrschbarer.

4 Extreme Programming (XP)

Extreme Programming ist wie Scrum ein agiles Vorgehensmodell. Es basiert nicht auf einer wissenschaftlichen Arbeit, sondern wird in vielen Aspekten aus Best Practices der Softwareentwicklung abgeleitet. Insbesondere nach der Überarbeitung 2004 will der Urheber, Kent Beck, damit nicht nur technische und organisatorische, sondern auch soziale Aspekte der Softwareentwicklung verbessern. Verglichen mit Scrum macht er dazu wesentlich umfangreichere und konkretere Vorgaben zur Methodik, gibt also die Arbeitsweise, die zur Agilität führen soll eher direkt vor. Risiken im Entwicklungsprozess versucht XP nicht zu minimieren, sondern aktiv anzugehen auch auf die Gefahr hin, dass der Versuch fehlschlägt.

Neben einem noch stärker iterativen Vorgehen steht hier das Verhalten der Beteiligten im Mittelpunkt. Als zentrale Werte für eine erfolgreiche Arbeit an Softwareentwicklungsprojekten identifiziert Beck: Kommunikation, Einfachheit, Feedback, Mut (i.S.v. Ehrlichkeit) und Respekt.

Außer diesen grundlegenden Aspekten enthält XP noch viele weitere Prinzipien und Vorgehensweisen bis hin zu Hinweisen zur Verbesserung der Zusammenarbeit und des Arbeitsumfeldes. Auch gibt es Erklärungen zu einzelnen Aufgaben, Rollen der Beteiligten und den zugrundeliegenden Überlegungen des Vorgehensmodells. Dieser Umfang deckt zwar einen größeren Teil der Softwareentwicklung ab als Scrum, macht XP allerding auch schwerer erlernbar. Dieses Kapitel beschränkt sich auf eine stark verkürzte Einführung in das wesentliche Vorgehen und wichtige sowie häufig genutzte Arbeitsweisen. Als umfassende Einführung und Erklärung empfiehlt sich Becks [XP2005].

Die Rollenaufteilung bei XP unterscheidet sich im Kern nicht stark von der in Scrum. Das nicht weiter unterteilte Entwicklerteam ist für die eigentliche Umsetzung verantwortlich, ein Product Owner priorisiert Aufgaben und verantwortet den Projekterfolg und der Kunde entscheidet, was überhaupt gemacht werden soll und gibt Rückmeldungen. Der Unterschied beim Kunden ist allerdings, dass er als aktiver Mitarbeiter im Projekt gesehen wird, um maximal bedarfsgerechte

Software entwickeln zu können. Der Product Owner kann zudem nach diesem Vorgehensmodell auch ein Vertreter des Kunden oder des Produktmanagements der Organisation sein. Bei der Ausgestaltung und Besetzung der einzelnen Rollen lässt XP mehr Möglichkeiten als Scrum, außerdem bietet es noch weitere Rollen, die überwiegend der Fokussierung und der Risiko-Abwägung dienen.

Als Ausgangspunkt der Entwicklung stehen bei XP „User Stories“ und teilweise auch „Themes“. Sie sind im weitesten Sinne vergleichbar mit den Product Backlog Einträgen bei Scrum und repräsentieren die Anforderungen des Kunden. Die User Stories werden vom Kunden allerdings nicht auf eine technische oder in einer distanzierten Weise beschrieben, sondern enthalten konkrete Beschreibungen der Anwender-Handlungen, die durch die Software und in der Software ermöglicht werden sollen. Themes werden als Ergänzung genutzt, um ein Gesamtbild der Software zu vermitteln. Beides soll primär durch den Kunden aber gemeinsam mit dem Team erstellt werden. Vollständige Projektspezifikationen sollen beim XP explizit nicht erstellt werden, da sie bei späteren Änderungen hohe Kosten und einen hohen Aufwand verursachen.

In einem „Planning-Game“ zu Beginn jeder Iteration wird dann gemeinsam vom Entwicklerteam und dem Kunden der Aufwand der User Stories geschätzt und anhand dieser Stories die nächste Version spezifiziert. Für den Kunden werden so die Kosten eines Features transparenter. Dabei gilt, dass bloß wirklich notwendige Features ausgewählt werden dürfen, um ein überflüssiges „design for the future“ – also die Umsetzung von nur vielleicht zukünftig einsetzbaren Features – zu verhindern. Als Kriterien der Priorisierung gibt XP den Nutzen und das Risiko vor. Der Idee entsprechend, Risiken aktiv anzugehen, sollen zunächst User Stories mit hohem Nutzen und hohem Risiko, erst nach deren Fertigstellung die mit hohem Nutzen bei geringem Risiko und abschließend die mit geringem Risiko und geringem Nutzen umgesetzt werden. Risiken ergeben sich dabei beispielsweise durch fehlende Erfahrungswerte oder potenziell hohen finanziellen und zeitlichen Verlusten bei Fehlern.

Resultat des Planning-Games ist ein Release-Plan für ca. ein Quartal, der mehrere Iterationen beinhaltet. Tatsächlich veröffentlicht wird die Software aber nur, wenn sie als geschlossenes Produkt erscheint und ausreichend Neuerungen integriert wurden. Jede Iteration innerhalb des Release-Plans soll nur eine Woche dauern, aber die Umsetzung einer oder mehrerer User Stories beinhalten. Das soll die Beteiligten auch dazu führen, Aufgaben möglichst atomar und User Stories möglichst einfach zu entwerfen.

Zur konkreten Bearbeitung einer User Story wird diese zunächst in mehrere Aufgaben geteilt, deren Aufwand separat geschätzt wird und nicht mehr als einige Stunden umfassen sollte. Anschließend übernehmen die Entwickler die offenen Aufgaben oder Aufgabenpakete mit ähnlichen oder technisch zusammengehörigen Aufgaben. Ihren Code sollten sie regelmäßig nach wenigen Stunden zur Verfügung stellen. Er gilt dann als kollektives Eigentum des Entwicklungsteams, wodurch auch eine kollektive Verantwortung erzeugt werden soll, sodass sich jeder dem Projekt verpflichtet fühlt. Als Ideal nennt XP außerdem das Prinzip des 10-Minuten-Builds, bei dem alle Tests und das Build bis hin zu einem fertigen Prototypen nach dem aktuellen Stand 10 Minuten dauern soll, um häufige Tests zu erleichtern.

Wie bei Scrum finden täglich Stand-Up Meetings statt, um die Kommunikation zu fördern. Situationsbedingt werden für die Bearbeitung der Aufgaben Arbeitspaare gebildet, die Pair-Programming anwenden. Auch Test Driven Development ist Bestandteil von XP, damit spätere aufwendige Korrekturen vermieden werden können. Die Implementation der Modul-Tests soll aber möglichst automatisiert werden.

Die ursprüngliche Version von XP sieht vor, dass neben der Implementation der Funktionalität kontinuierlich Refactoring betrieben wird, zumindest sobald es sinnvoll ist. Hier stellt sich die Frage, ob nicht eine weitergehende Vorausplanung den Refactoring-Bedarf effizient mindern würde. Aus eigener Erfahrung stellt das Vorgehen zudem hohe Anforderungen an die Entwickler, weil bei der Implementation neuer Funktionalitäten auch neue Abhängigkeiten in der Software entstehen, durch die ein gleichzeitiges Refactoring zu Inkonsistenzen führen kann. Um das zu vermeiden, muss schon von vornherein sehr strukturiert und geplant entworfen und implementiert werden, was der Grundlage entgegensteht, anfangs keine umfangreiche Spezifikation zu erstellen und inkrementell zu entwickeln. Damit ist dies ein Beispiel für ein Prinzip von XP, dass die Komplexität und die Schwierigkeit eher erhöht. Das steht zwar Agilität nicht zwangsweise entgegen, fördert sie aber auch nicht. Zwar geht die aktuelle Edition von XP nicht mehr auf diesen Punkt ein, allerdings findet man diese Vorgehensweise auch noch in Sekundärliteratur wie [XPSEK01] und [XPSEK02].

Verglichen mit anderen Modellen sieht XP Akzeptanztests durch den Kunden sehr häufig vor, nämlich im Abstand weniger Tage. Zusätzlich wird eine Begutachtung durch die Stakeholder nach jeder Iteration empfohlen. Das entspricht prinzipiell der Vorgehensweise mit Scrum, findet aber durch die deutlich kürzeren Iterationen viel häufiger statt. Dieses Vorgehen stellt so wesentlich höhere Anforderungen an den Kunden, bindet ihn aber im Gegenzug stärker mit ein. Verschärft wird das noch, indem XP die ständige Anwesenheit eines Vertreters des Kunden vorsieht, der auch jederzeit in die Entwicklung eingreifen und Anforderungen aktualisieren oder korrigieren darf.

Das agile Prinzip der nachhaltigen Arbeitsweise wird ganz direkt umgesetzt, indem unter „Energized Work“ als eine der primären Vorgehensweisen von einer Überarbeitung auf Kosten eines gleichbleibenden Arbeitstempos abgeraten wird.

5 Nutzen von Extreme Programming

Generell kann zu XP festgestellt werden, dass es jedes agile Prinzip beachtet. Kurze Iterationen und auch generell kurze Zeiträume bilden hier die Basis für Agilität, die auch direkt durch eine starke Einbeziehung des Auftraggebers genutzt wird. Methoden und Richtlinien wie Test Driven Development, kontinuierliche Code-Integration und Pair Programming helfen dabei, Fehler zu vermeiden, die auch spätere Änderungen erschweren würden und fördern den Wissensaustausch zwischen den Entwicklern.

Für sich genommen basieren diese Argumente aber bloß auf Theorien und noch viel mehr auf Annahmen. Um den tatsächlichen Nutzen von agilen Vorgehensmodellen in der Softwareentwicklung und von XP im Speziellen zu beurteilen, können allerdings Studien wie die von Abrahamsson und Koskela oder von Rumpe und Schröder herangezogen werden. Rumpe und Schröder haben 2002 in einer Befragung von 45 Teilnehmern über ihre Erfahrungen mit Projekten, die XP einsetzten, die Auswirkungen von agilem Vorgehen ermittelt. Abrahamsson und Koskela hingegen haben eine Fallstudie über den Zeitbedarf, die Leistung und die resultierende Softwarequalität von Extreme Programming durchgeführt. Die Ergebnisse beider Studien werden im Folgenden zusammengefasst.

Die Befragung haben Personen aus unterschiedlichen Projekten, Organisationen, Branchen, Ländern und mit unterschiedlichen Rollen im Team beantwortet, nachdem auf verschiedenen Kommunikationswegen (E-Mail, persönliche Ansprache, Kontaktpersonen in Organisationen) darum gebeten wurde. 22 der Teilnehmer äußerten sich zu laufenden Projekten, 23 Teilnehmer hatten das betreffende Projekt bereits abgeschlossen. Da nur 45 Fragebögen von beliebig „gewählten“

Teilnehmern ausgewertet wurden, ist die Studie nicht repräsentativ. Trotzdem liefert sie Einblicke in Praxiserfahrungen mit XP – allerdings nach der ursprünglichen Variante von XP. Ganz allgemein kommen die Urheber der Studie zu dem Schluss, dass dieses Rahmenwerk besonders für innovative Unternehmen geeignet und generell tatsächlich ein interessanter Ansatz zur Softwareentwicklung sei.

Überraschend ist, dass von den Befragten 44 ihr Projekte als „erfolgreich“ und einer als „teilweise erfolgreich“ beurteilt haben. Es gab keine Rückmeldungen über fehlgeschlagene Projekte. Da der Anteil an scheiternden Softwareprojekten im Allgemeinen merklich höher ist[4], spricht dieses Ergebnis entweder für den Nutzen von XP oder – was wahrscheinlich erscheint – ebenfalls dafür, dass die Studie nicht repräsentativ ist und es zusätzlich womöglich eine Wahrnehmungsdifferenz zwischen Entwicklern und Kunden bezüglich des Erfolgs gibt. Außerdem fällt auf, dass Spezialfälle wie ein Abbruch der Nutzung von XP durch das Umfragedesign nicht abgedeckt wurden.

Zur Beurteilung der Studie weisen auch die Urheber selbst darauf hin, dass es sich um eine frühe Studie handelt, an der dementsprechend eher Early Adopter mit Interesse an agiler Softwareentwicklung teilgenommen haben. 75% der Projekte wurden zudem mit Teams, bestehend nur aus erfahrenen Entwicklern oder teilweise aus erfahrenen Entwicklern, durchgeführt. Das spricht für eine methodenunabhängige höhere Erfolgswahrscheinlichkeit, aber auch für einen kompetenteren Vergleich der Vorgehensweisen.

Der Vergleich ist auch eins der interessantesten Ergebnisse. Die Befragten beurteilen nämlich die Erreichung ausgewählter Ziele von XP als tatsächlich deutlich besser mit XP erreichbar als mit traditionellen Vorgehensmodellen. Konkret hätten die Einhaltung des Zeitplans, der „Fun Factor“ der Arbeit, die Softwarequalität und auch die Verringerung der Kosten für späte Änderungen sich mit XP verbessert, wie Abb. 1 zeigt. Die Daten zeigen allerdings auch, dass die Senkung der Änderungskosten im Vergleich zu traditionellen Vorgehensweisen bei sehr weit vorangeschrittenen Projekten als nicht so stark wahrgenommen wird, wie dies bei frühen Änderungen der Fall ist.

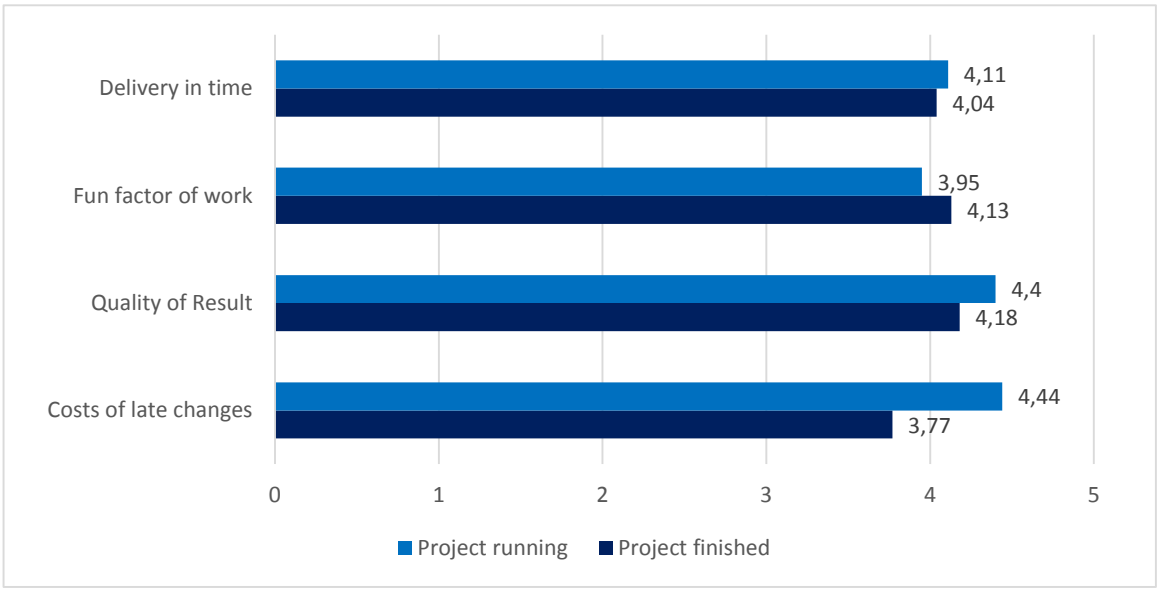

Abb. 1 Erreichung ausgewählter XP Ziele im Vergleich zu traditionellen Methoden
(Skala: -5 „deutlich schlechter“ bis 5 „vollständig erreicht“)
nach: [Survey2002], Seite 5

[4] Generelle Aussagen und Studien wie [ELEMAM2008] weisen auf einen üblicherweise höheren Anteil fehlgeschlagener Softwareprojekte hin.

Ähnlich positiv haben die Befragten den Nutzen einzelner Elemente von XP bewertet. Nur für den in der ursprünglichen XP-Version noch empfohlenen Einsatz von Metaphern und den dauerhaft anwesenden Kunden gab es einzelne Bewertungen, die darin ein Hindernis für den Erfolg sahen. Die anderen Kernelemente sahen jeweils mindestens 80% der Befragten als Hilfreich, die übrigen nicht als kritisch aber als verbesserungswürdig an. Das Ergebnis und die bewerteten Elemente zeigt Abb. 2.

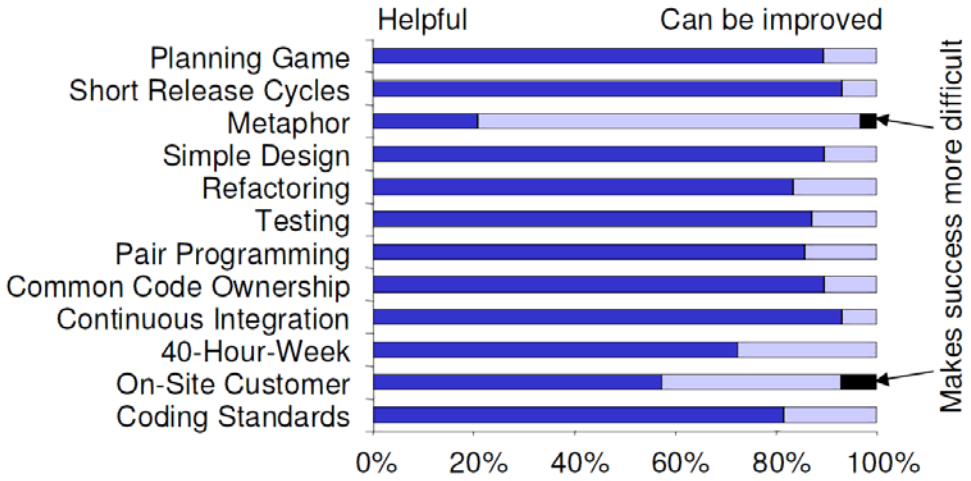

Abb. 2 Nutzen einzelner XP Elemente
(Auswahlmöglichkeiten: „helpful, improvable, or even making-difficult")
aus: [Rumpe2002], Seite 4

Auch bei der freien Frage nach festgestellten Risiken für den Projekterfolg gab es wenig Kritik, die ein spezifisches Problem von XP ist. Unter den am häufigsten genannten Risiken sind nur Probleme mit dem mitarbeitenden Kunden XP-spezifisch, da die starke Kundeneinbindung ein Prinzip von XP ist. Die übrigen genannten Risiken können ebenso bei anderen Vorgehensmodellen auftreten. Abb. 3 zeigt zudem, dass selbst die häufigsten Risiken eher selten genannt wurden.

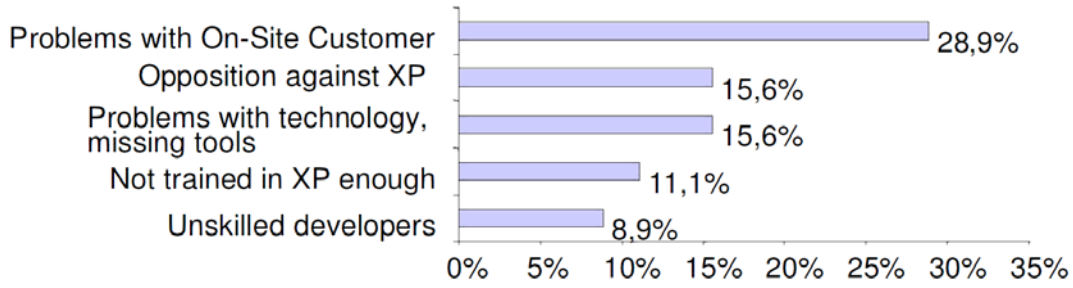

Abb. 3 Größte Risiken für den Projekterfolg
(Freier Text – nachträglich zu Kategorien zusammengefasst)
aus: [Rumpe2002], Seite 5

Abrahamssons Fallstudie untersucht zwar andere Aspekte, kommt insgesamt aber ebenfalls zu einem eher positiven Ergebnis. Hauptsächlich wurden in der Studie der Aufwand verschiedener Tätigkeiten, die Leistung (in Code-Zeilen) und die Produktqualität (anhand der Fehlerzahl) ermittelt. Auch Veränderungen während des Projekts im Vorgehen der Entwickler wurden so untersucht. Im Rahmen der Studie wurde ein Entwicklerteam aus vier Studenten mit Praxiserfahrung in objektorientierter Entwicklung in XP eingeführt. Anschließend haben sie in 6 Iterationen ein web-basiertes

Datenmanagementsystem für Forschungsdaten entwickelt, das für den tatsächlichen Einsatz in einem Forschungszentrum vorgesehen war. Die Urheber der Studie nahmen zusammen mit Vertreter des Forschungszentrums die Rollen des Kunden ein. Wie in den meisten realen Situationen waren die Ressourcen festgelegt, die Funktionalitätsanforderungen aber verhandelbar.

Indem die Studie zeigt, dass nicht alle Prinzipien von XP notwendig sind, was vor allem für den „Kunden vor Ort“ galt, der nur 21% der Zeit in Anspruch genommen wurde, entkräftet sie den Kritikpunkt, dass zwangsweise alle Bestandteile des Vorgehensmodells umgesetzt werden müssen. Außerdem fielen somit die Personalkosten des Kunden für die Bereitstellung eines Mitarbeiters geringer aus als erwartet. Gleichzeitig bewerteten die Entwickler die Anwesenheit des Kunden als Motivation und Ausdruck einer Wertschätzung ihrer Arbeit. Von Kundenseite war die Einbeziehung unter anderem positiv, da die Verschiebungen von Feature-Releases so nicht unerwartet stattfanden.

Anders als es aus der Literatur erwartet wurde, machte der Anteil der Programmierarbeit nur 54,7% der Arbeit aus, obwohl gerade XP der eigentlichen Programmierarbeit eine hohe Bedeutung einräumt. Positiv war hingegen, dass der Prozess flexibel genug ist, um spezielle Anforderungen wie die Datenerhebung einfach zu integrieren. Zudem sei laut den Autoren ein Lerneffekt bei der Aufgabenunterteilung erkennbar, sodass im Verlauf der Studie der Zeitverlust durch fehlerhafte Aufwandsschätzungen abnahm. Dies kann – allerdings nur in begrenztem Maße – anhand der Abbildungen 4 und 5 nachvollzogen werden.

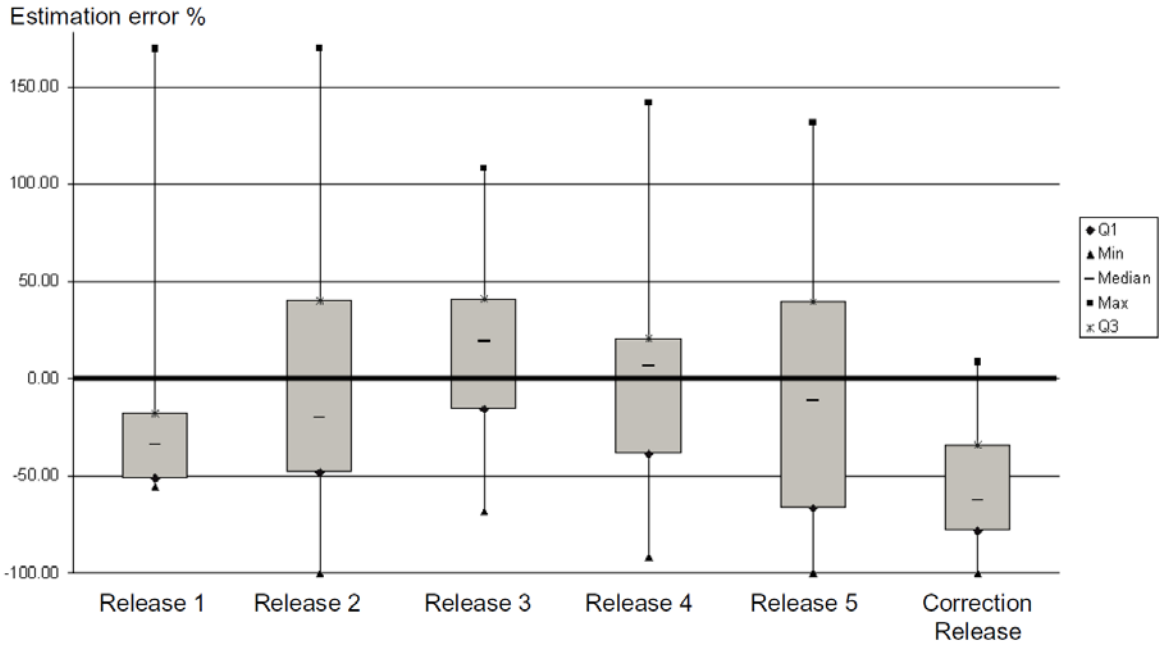

Abb. 4 Relative Abweichungen der Aufwandsschätzungen je Iteration
aus: [Abrahamsson2004], Seite 6

Die obere Abbildung zeigt nach Abrahamsson, dass anfangs eine Tendenz zur Überschätzung des Aufwandes bestehe. Der Median nähere sich dann einer Abweichung von 0% an, wobei die Varianz in den Abweichungen der Aufwandsschätzungen hoch bleibe. Abb. 5 zeige, dass sich die absolute Abweichung der Schätzungen verringert. Das ist hauptsächlich darauf zurückzuführen, dass die Entwickler Aufgaben feiner gliedern und könnte zu weniger falsch zugewiesenen Ressourcen und einem geringeren Zeitverlust führen. Anhand der eingezeichneten Datenpunkte ist aber nur ein schwacher Trend erkennbar, der auch andere Gründe haben könnte und nicht zwangsweise auf eine Stärke von XP hinweist.

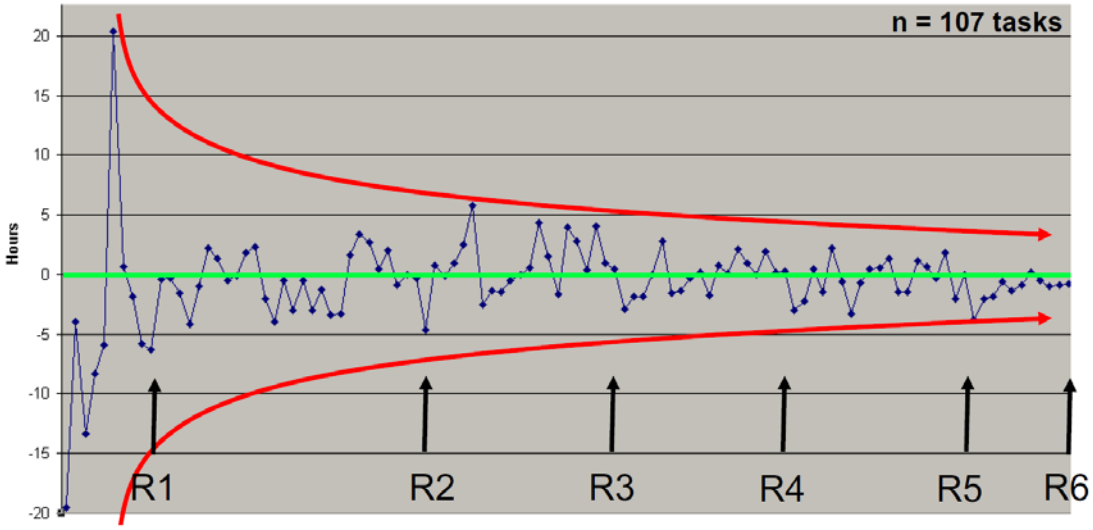

Figure 3. XP Pulse: Hours lost by faulty estimates

Abb. 5 Absolute Abweichungen der Aufwandsschätzungen
aus: [Abrahamsson2004], Seite 6

Insgesamt zeigen diese Studien allerdings ein positives Bild von dem praktischen Nutzen von XP, das zu den eingangs und auch von den Urhebern agiler Vorgehensmodelle getroffenen Annahmen passt. Obwohl dieses Bild weder alle Aspekte der Softwareentwicklung abdeckt, noch völlig eindeutig und repräsentativ ist, kann selbst aus einem pessimistischen Blickwinkel festgestellt werden, dass beide Studien zumindest keinen Grund liefern, die Wirksamkeit agiler Vorgehensweisen anzuzweifeln.

6 Gründe für agile Vorgehensweisen und Grenzen des Einsatzgebietes

Die eingangs erwähnte Komplexität der Softwareentwicklung ist eine generelle Schwierigkeit, für die nicht nur agile Vorgehensweisen Lösungsansätze liefern. Zu diesem allgemeinen Problem größerer (Weiter-)Entwicklungsprojekte kommt häufig noch ein sich schnell änderndes Umfeld, in dem die Software eingesetzt werden soll. Dies ist ein Hauptgrund, der für den Einsatz agiler Vorgehensmodelle spricht.

In der Wissenschaft können sich Modelle und damit unter Umständen auch Anforderungen an die Software durch neue Erkenntnisse verändern. Software für Wirtschaftsunternehmen hingegen sollte sich den Geschäftsprozessen anpassen, die aber durch Kundenwünsche, Trends, Innovationen oder Konkurrenzdruck ebenfalls Veränderungen unterworfen sind. In beiden Fällen können sich zudem

Prioritäten ändern oder die für die Entwicklung verfügbaren Technologien und die Systemumgebung für den Einsatz der Software.

Falls der – offenbar eher seltene[5] – Fall vorliegt, dass von Anfang an alle Anforderungen und Rahmenbedingungen für das System vollständig spezifiziert werden können und sich im Lauf der Entwicklungszeit auch nicht verändern werden, ist hingegen kein agiles Vorgehen nötig. Dennoch können agile Modelle angewendet werden, bieten dann aber höchstens geringe Vorteile.

Allerdings bleiben auch dann noch die Vorteile aus dem Grundprinzip, möglichst nach jeder Iteration nutzbare Software zu haben. Falls einzelne Funktionen der Software also auch vor dem Endstadium sinnvoll eingesetzt werden können, um Erkenntnisse für die Entwickler, gegebenenfalls wissenschaftliche Erkenntnisse oder Einnahmen zu generieren, spricht dies insbesondere bei Entwicklungsprojekten, die nicht bloß wenige Wochen dauern, für die Anwendung agiler Vorgehensweisen.

Empfehlenswert ist das auch, wenn ansonsten zukünftige Anwender der Software nicht in die Entwicklung einbezogen werden würden. Bedingt durch die iterative Entwicklung können diese schon früh Feedback geben oder sich zumindest bereits mit der Software beschäftigen. Innerhalb des Change Managements wird davon ausgegangen, dass dies üblicherweise die Akzeptanz steigert. Eventuell verringert sich so auch der nötige Schulungsaufwand.

Aber auch wenn alle Gründe für eine Anwendung agiler Modelle vorliegen, können trotzdem Gegenargumente überwiegen – nämlich dann, wenn die Voraussetzungen der einzelnen Methoden und Vorgehensmodelle nicht gegeben sind. Insbesondere bei zu kleinen Teams mit etwa 3 Personen oder weniger[6] und Ablehnung auf Kundenseite sollten Alternativen zur agilen Softwareentwicklung in Betracht gezogen werden.

Gemäß der Prinzipien agiler Softwareentwicklung und der Anforderungen vieler Vorgehensmodelle eignen sich diese außerdem nicht, wenn die Entwicklerteams geografisch verteilten sind und größere Distanzen zur Kommunikation überbrücken müssen. Gerade für Open Source Projekte oder globale Forscherteams aber auch immer häufiger für kommerzielle Softwareprojekte stellt dies eine Hürde dar. Denkbar ist zum Beispiel die Überwindung dieser Hürde, falls und indem die Entwicklungsaufgabe modularisiert und an verschiedenen Orten Entwicklungsteams gebildet werden können. Aus der Praxis gibt es sowohl positive[7] als auch negative[8] Rückmeldungen über die Vereinbarkeit von agiler Softwareentwicklung und größeren Distanzen zwischen den Entwicklern.

Auch wenn der Auftraggeber agile Vorgehensmodelle begrüßt und bereit ist, mitzuarbeiten, birgt es zusätzliches Konfliktpotenzial, wenn Veränderungen an den Anforderungen des Kunden grundsätzlich akzeptiert werden sollen. Dieses Vorgehen empfiehlt sich deswegen nur bei einer ausreichenden Vertrauensbasis und einer angepassten Abrechnung der Entwicklungsleistungen. Angebote mit einem Festpreis sind dann kaum möglich, da der Aufwand und die Gesamtdauer anfangs nicht abschätzbar sind und sich prinzipiell jederzeit ändern könnten. Als Lösungsansatz empfiehlt beispielsweise Beck ein Pay-per-Use Modell[9], da es zusätzlich ein direktes Kundenfeedback darstelle und den Anreiz verstärke, für den Kunden passende Software zu entwickeln. Ansonsten bietet sich zumindest eine pauschale Abrechnung jeder notwendigen Iteration

[5] Das ist beispielsweise daran erkennbar, dass Anforderungsänderungen inzwischen als üblich angesehen werden, wie es z.B. im Artikel [CHANGE2004] eingangs dargestellt wird.
[6] entspricht beispielsweise der Empfehlung im Scrum Guide [SCRUM2013]
[7] beispielsweise von Escher in [DISTANZ01]
[8] beispielsweise von Rosenstein in [DISTANZ02]
[9] siehe [XP2005] Seite 69f.

ergänzt um eine Aufwandsschätzung der anfänglichen Anforderungen an. So zahlt letztlich der Kunde für seine späteren Änderungswünsche einfach selbst und kann dabei die Entwicklungsarbeit und die Leistungen in einer Iteration durch die intensive Einbeziehung besser verstehen.

Selbstverständlich kann auch bloß intern agil entwickelt werden, wobei ein Mitarbeiter des Entwicklerteams primär die Perspektive des Kunden übernimmt und als Basis zum Beispiel trotzdem klassische Pflichten- und Lastenhefte nutzt.

7 Zusammenfassung

Agile Vorgehensmodelle basieren auf iterativer Entwicklung und vermeiden es, Hindernisse für spätere Änderungen zu verursachen. Sie enthalten Ansätze zum Vorgehen bei der Planung und zur Arbeitsweise und ergänzen so die technischen Frameworks und Muster um solche zur Organisation des Prozesses insgesamt. Indem sie die persönliche, direkte Zusammenarbeit zwischen den Entwicklern und zwischen Entwicklern und Kunden forcieren und auf selbstorganisierte Teams setzen, mindern sie die ansonsten ggf. durch Organisationsstruktur und Kompetenzverteilung bzw. der sozialen Komponente erhöhte Komplexität. Auch der, mit großem (Code-)Umfang einhergehenden, Unklarheit über Ursache-Wirkungsbeziehungen in der Software, die durch Wissensmonopole einzelner Entwickler verstärkt wird, wird so entgegengewirkt, was bei der Beherrschung der Komplexität in der technischen Komponente der Softwareentwicklung hilft.

Im Mittelpunkt stehen bei agiler Softwareentwicklung qualitativ hochwertige Software und die beteiligten Menschen. Auf diese Weise wird versucht, die generellen Ziele einer nützlichen und verlässlichen Software zu erreichen. Overhead wird dabei zugunsten der Konzentration auf das eigentliche Produkt vermieden, um effizienter und agiler zu werden. Der Nutzen des Produkts wird bei agilen Modellen gesteigert, indem alle anfallenden Erkenntnisse genutzt werden und die Software früh ausgeliefert und möglichst passend zu den tatsächlichen Anforderungen hergestellt wird.

Dabei ist es eine zentrale, aber nicht zwingend zu erfüllende Annahme, dass Anforderungen und Vorgaben anfangs unvollständig oder unklar sind oder sich während des Projekts ändern.

Als Vorteil können ein früher Nutzen und frühe Erfolgserlebnisse durch einsatzfähige Produktinkremente und eine intelligente Anforderungspriorisierung gesehen werden. Die stärkere Einbeziehung des Kunden und unter Umständen auch der Anwender kann die Akzeptanz bei der späteren Einführung steigern. Zuletzt ermöglicht es eine hohe Agilität, Projektabbrüche zu vermeiden, selbst wenn sich die Zielsetzung vergleichsweise stark verändert. Das erspart dem Kunden zumindest den Totalverlust seiner bisherigen Investitionen in die Software.

Die konkrete Vorgehensweise unterscheidet sich zwischen verschiedenen agilen Vorgehensmodellen trotz des gemeinsamen Grundgedankens teilweise stark. Scrum und Extreme Programming verfolgen unterschiedliche Ansätze, um letztlich gleiche Ziele zu erreichen. Scrum richtet sich eher an das (Projekt-)Management und ist als Modell sehr simpel und schnell erlernbar. Die zentralen Ideen sind Transparenz und Empirie. Extreme Programming hingegen behandelt zwar viele organisatorische Fragestellungen, bleibt dabei aber in einigen Aspekten wesentlich stärker technikorientiert. Es setzt stark auf erwiesene Best Practices. Hier sind Kommunikation, unbeschränkte Anforderungsänderung, sowie schnelle, kleine Schritte die zentralen Ideen.

Falls man agile Methoden verwendet, sollte man keine falschen Schlüsse aus deren Prinzipien ziehen. Dass die Entwickler ihre Entwicklungsarbeit selbst organisieren sollen, macht Manager nicht überflüssig, da eine Organisation auch als Ganzes funktionieren muss. Und die Fokussierung auf die

eigentliche Software und Anpassungsfähigkeit bedeutet selbst für die Entwickler der agilen Vorgehensweisen nicht, dass Prozesse und Werkzeuge, angemessene Dokumentationen, Vertragsverhandlung oder das Befolgen eines Plans unnötig sind. Außerdem sind die existierenden Vorgehensmodelle nicht endgültig, sondern die Methoden und Prozesse sollten weiterhin verbessert werden und auch selbst an veränderte Herausforderungen angepasst werden.

Abschließend lässt sich sagen, dass es in vielen Situationen der Softwareentwicklung gute Argumente für die Anwendung agiler Vorgehensmodelle gibt. Studien weisen ebenfalls darauf hin, dass der versprochene Nutzen nicht bloß theoretisch existiert. Allerdings sollten Kontext und Aufgabenstellung beachtet werden. Bei vollständig spezifizierten Systemen, die wenig Interaktion mit Nutzern erfordern, bieten agile Modelle wenige Vorteile. Zudem gibt es kommerzielle wie wissenschaftliche Projekte, die sich aufgrund der Teamstruktur weniger für diese Modelle eignen.

8 Literatur- und Quellenverzeichnis

Quellen, auf die im Text verwiesen wird:

[AGILH2001] Jim Highsmith: History: The Agile Manifesto.
Snowbird (USA) 2001
http://agilemanifesto.org/history.html (letzter Abruf: 06.09.2015)

[AGILM2001] Kent Beck, Mike Beedle et al.: Manifesto for Agile Software Development.
Snowbird (USA) 2001
http://agilemanifesto.org/ (letzter Abruf: 06.09.2015)

[AGILP2001] Kent Beck, Mike Beedle et al.: Principles behind the Agile Manifesto.
Snowbird (USA) 2001
http://agilemanifesto.org/principles.html (letzter Abruf: 06.09.2015)

[BENIN1956] Herbert D. Benington: Production of Large Computer Programs

[CHANGE2004] Allan Kelly: Why do requirements change? (aus: Overload Journal Nr. 59).
ACCU, 2004
(http://accu.org/var/uploads/journals/overload59-final.pdf)

[DISTANZ01] Cédric Escher: Agil Distanzen überwinden (aus: ERNI Experience Nr. 55).
ERNI Management Service AG, Zürich 2012
(http://www.erni-consultants.com/digital-publication/de/experience/55/international-projects)

[DISTANZ02] Aviva Rosenstein: The UX Professionals' Guide to Working with Agile Scrum Teams.
2013
http://boxesandarrows.com/the-ux-professionals-guide-to-working-with-agile-scrum-teams/ (letzter Abruf: 07.09.2015)

[ELEMAM2008] Khaled El Emam, A. Günes Koru: A Replicated Survey of IT Software Project Failures.
Ottawa (Kanada) 2008
(http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4602680)

[GABLER01] Markus Siepermann: Stichwort „Agile Softwareentwicklung" (aus: Gabler Wirtschaftslexikon).
Springer Gabler Verlag
http://wirtschaftslexikon.gabler.de/Archiv/381707695/agile-softwareentwicklung-v5.html (letzter Abruf: 05.09.2015)

[HEISE2010] Alexander Neumann: „Studie: Agile Softwareentwicklung ist Mainstream".
Heise Medien GmbH & Co. KG, Hannover 2010
http://heise.de/-912207 (letzter Abruf: 03.09.2015)

[KOMUS2014] Ayelt Komus et al.: Status Quo Agile 2014.
Koblenz 2014
http://www.hs-koblenz.de/rmc/fachbereiche/wirtschaft/forschung-projekte/forschungsprojekte/status-quo-agile/ (letzter Abruf: 03.09.2015)

[SCHOE2014] Klaus-Peter Schoeneberg: Komplexitätsmanagement in Unternehmen. Springer Gabler Verlag, Wiesbaden 2014

[SCRUM2013] Ken Schwaber, Jeff Sutherland: „Der Scrum Guide – Der gültige Leitfaden für Scrum: Die Spielregeln“ (offizielle deutsche Übersetzung). 2013 (http://scrumguides.org/)

[STUDY2004] Pekka Abrahamsson, Juha Koskela: „Extreme Programming: A Survey of Empirical Data from a Controlled Case Study“. Oulu (Finnland) 2004 (http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=1334895)

[SURVEY2002] Bernhard Rumpe, Astrid Schröder: Quantitative Survey on Extreme Programming Projects. 2002 (http://arxiv.org/abs/1409.6599)

[XP2000] Kent Beck et al.: „Extreme Programming Explained: Embrace Change“. Addison-Wesley, Boston (USA) 2000

[XP2005] Kent Beck et al.: „Extreme Programming Explained: Embrace Change – 2. Edition“. Addison-Wesley, Boston (USA) 2005

[XPSEK01] Don Wells: Refactor Mercilessly (in: „Extreme Programming: A gentle introduction“). http://www.extremeprogramming.org/rules/refactor.html (letzter Abruf: 13.09.2015)

[XPSEK02] Jonathan Rasmusson: Extreme Programming (in: Agile In a Nutshell; insbesondere der Abschnitt: The Practices). http://www.agilenutshell.com/xp (letzter Abruf: 13.09.2015)

Quellen ohne Verweis im Text:

[AGIL01] Agile software development (insbesondere Kapitel 1.1 Predecessors). http://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=663229066#Predecessors (letzter Abruf: 14.06.2015)

[SCRUM01] The History of Scrum. http://www.scrumguides.org/history.html (letzter Abruf: 10.09.2015)

[XP01] Extreme Programming. http://de.wikipedia.org/w/index.php?title=Extreme_Programming&oldid=142876153 (letzter Abruf: 15.06.2015)

[XP02] eXtreme Programming (XP). it-agile GmbH, Hamburg http://www.it-agile.de/wissen/methoden/extreme-programming/ (letzter Abruf: 15.06.2015)

9 Abkürzungsverzeichnis

Abb. – Abbildung

bzw. – beziehungsweise

ca. – circa

ggf. – gegebenenfalls

i.S.v. – im Sinne von

XP – Extreme Programming

z.B. – zum Beispiel