Empirische Evidenz von agilen Methoden

Ausarbeitung im Rahmen der Veranstaltung
„Seminar in Software Engineering“

Seminararbeit
Wintersemester 2003/2004

Franc Uffer
99-913-006

Institut für Informatik
der Universität Zürich
Prof. Dr. Martin Glinz

Betreuer: Dipl.-Inf. Christian Seybold

Präsentation am 03.02.2004

INHALTSVERZEICHNIS

1. Einleitung

Seit einigen Jahren befinden sich agile Methoden nicht mehr nur in Insiderkreisen im Aufschwung. In der vorliegenden Arbeit sollen im Rahmen des Seminars in Software Engineering praktische Erfahrungen in der Anwendung von agilen Entwicklungsmethoden untersucht werden. Die Frage ist, welche Methoden unter welchen Gegebenheiten zum Erfolg führen können oder warum andere Methoden nicht geeignet sind oder gar nichts Neues hervorgebracht haben.

2. Was heisst „agil“?

So wie sich jedermann vor rund 15 Jahren damit brüstete, objektorientiert zu entwickeln, so deklarieren heute viele ihre Prozesse und Projekte als agil. Aber in Wirklichkeit geben die meisten nur vor agil zu sein, was in diesen Projekten Tag für Tag geschieht ist immer noch dasselbe wie vor der vermeintlichen Umstellung auf agile Methoden. Das ist traurig, denn das Manifest für agile Softwareentwicklung verspricht eigentlich eine neue Art zu arbeiten:

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

(Manifesto for Agile Software Development) [1]

Mehr als die traditionellen Methoden (Wasserfallmodell, Wachstumsmodell, Spiralmodell etc.) rücken die agilen Methoden vermehrt die Zusammenarbeit und die menschliche Komponente in den Vordergrund. Entwickeln von Software soll kein individualistischer Akt sein, sondern durchgehend Teamarbeit. Was heisst denn nun aber „agil“ im Zusammenhang mit Software Entwicklung? Der Wortbedeutung nach zu urteilen, ist der hauptsächliche Aspekt von agil nicht die Art der Zusammenarbeit, sondern vor allem die Anpassungsfähigkeit. Zentraler Punkt für alle Arten von agilen Methoden muss daher die Bereitschaft für Änderungen sein, wann immer sie auch anfallen sollten. Wo hier allerdings die Grenze zwischen spontaner Flexibilität und gewissenhafter Planung zu ziehen ist, ist eine strittige Frage, aus derer nicht zuletzt auch viele unterschiedliche Methoden hervorgingen. Was agile Methoden letzten Endes ausmachen, wird bereits in unzähligen Artikeln und Büchern aufgeführt. Zudem wurde in den vorangegangenen Vorträgen in diesem Seminar zum Teil bereits ausführlich darüber berichtet. Interessanter mag an dieser Stelle daher die Sammlung von Pete McBreen [2] sein. Er führt zehn Anzeichen auf, die auf lediglich vorgespielte Agilität schliessen lassen:

2.1 10 Anzeichen, dass ein Projekt Agilität nur vorgibt (nach McBreen)

10 Der Projektplan wurde soeben veröffentlicht und sieht das erste Release in 18 Monaten vor

Wenn das passiert, so kann Agilität ausgeschlossen werden. Die für agile Methoden unerlässlichen Feedbacks wären nicht in sinnvoller Zeit möglich. Das erste Release nach 10 Monaten ruiniert praktisch ein Projekt.

9 Der Projektmanager spricht von Ergebnissen, die die Systemanalysten den Softwarearchitekten übergeben

Warnung: Wasserfall voraus! Als nächstes werden die Architekten ihre Ergebnisse den Designern übergeben…

8 Die Systemanalysten und Softwarearchitekten sind stolz, beim letzten Projekt keinen Code geschrieben zu haben

Dieses Denken fördert die Ansicht, dass die Dokumentation wichtiger ist als die lauffähige Software – ein klarer Verstoss gegen das Manifest.

7 Die Struktur des Projektes sieht die Programmierer und Tester ganz am Schluss der Nahrungskette

Agile Projekte starten viel früher mit Codieren und Testen als traditionelle Ansätze, meistens innerhalb der ersten Wochen nach Projektbeginn.

6 Die Systemanalysten lassen die Anforderungsdokumente vom Benutzer unterschreiben

Die Anforderungen einzufrieren ist unter Umständen eine gute Idee, aber der agile Ansatz beruht auf enger Zusammenarbeit mit dem Kunden. So kann gewährleistet werden, dass die gelieferte Software die aktuellen Bedürfnisse deckt und nicht die aus der Zeit des Vertragsabschlusses.

5 Das Entwicklerteam beschwert sich über alle angenommenen Änderungswünsche

Dies ist ein einfacher Punkt aber dennoch ein oft auftretender. Agilität verlangt von allen Teilnehmern eine ständige Bereitschaft für Änderungen, unabdingbar v.a. wegen Punkt 6.

4 Das Projekt läuft seit über zwei Monaten, dennoch wurde noch keine Funktionalität demonstriert

Hier sind keine PowerPoint Vorstellungen gemeint. Der Benutzer muss früh eine erste funktionierende Version sehen, um mit Feedback überhaupt von Nutzen sein zu können. Nur so kann das Erreichen der gewünschten Ziele möglich gemacht werden.

3 Der Verlauf des Projektes führt zur Annahme, dass Dokumentation die Kommunikation überwiegt

Wenn ein Projekt zu viel Papier erzeugt und alle Entscheidungen aufgeschrieben werden nur um Anforderungen verfolgbar zu machen, dann steht es um die Agilität schlecht. Wenn eine Information wichtig genug ist, um sie in die Dokumentation aufzunehmen, so sollte das von einem professionellen technischen Autor vorgenommen werden. Wenn leichte Verständlichkeit keine Priorität hat, so kann wohl jeder Projektteilnehmer seine Dinge aufschreiben – mit agil hat das dann mit Sicherheit aber nichts mehr zu tun.

2 Testen und Qualitätssicherung sind kein integraler Bestandteil des Entwicklerteams

Alle agilen Ansätze beruhen auf frühem Testen und damit verbundenem Feedback zur Sicherstellung der Qualität. Testen und Qualitätssicherung starten zusammen mit dem Rest des Projektes am ersten Tag. Eine Verschiebung in eine spätere Phase ist ein sicheres Zeichen, dass der gewählte Prozess nicht agil ist.

1 Tasks werden an einzelne Mitarbeiter vergeben, die diese in Soloarbeit verrichten

Wann immer Mitarbeiter sich abschotten um „endlich vorwärts zu kommen“, so ist klar, dass die Verantwortlichen mit dem Begriff kollaboratives Arbeiten nichts anfangen können. Wenn Projektleiter diesen zentralen Punkt nicht begreifen, dann spielt es keine Rolle, welche Poster an der Wand hängen mögen: Das Projekt ist mit Sicherheit nicht agil.

3. Übersicht agiler Methoden

Auf den folgenden Seiten findet sich eine kurze Übersicht über die bekanntesten agilen Methoden. Die Beschreibung wurde sehr kurz gehalten, da zu den einzelnen Methoden bereits Unmengen an Material zu finden ist. Diese Übersicht soll dem Leser mit Vorkenntnissen der jeweiligen Methoden nur kurz die Hauptpunkte wieder in Erinnerung rufen. Folgende Übersicht zeigt die meist verbreiteten agilen Methoden:

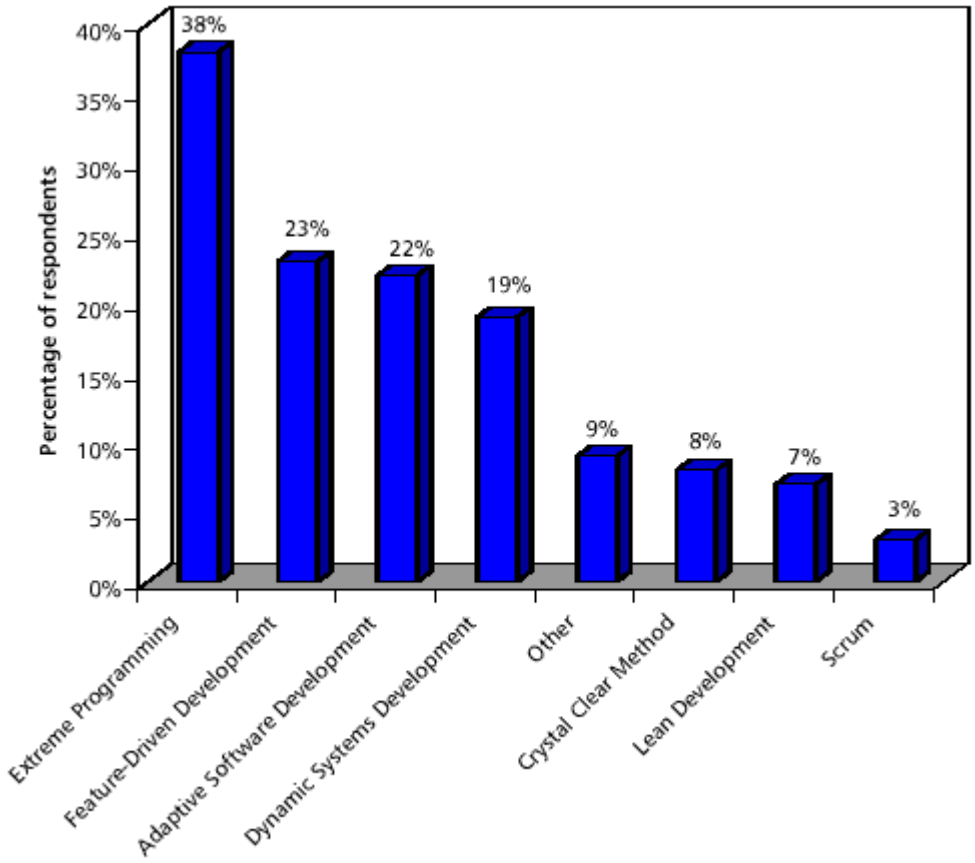

Abbildung 1: Die meist verwendeten agile Methoden [3]

3.1 Extreme Programming

Von allen leichtgewichtigen Methoden ist XP mit Sicherheit die bekannteste. Die Wurzeln von XP liegen im Smalltalk, insbesondere in der engen Zusammenarbeit von Kent Beck und Ward Cunningham. Der Sprung von informellen Vorgehensweisen zu einer eigenen Methode geschah im Frühling 1996. Kent wurde für ein Review eines Abrechnungsprojektes bei Chrysler zugezogen, welches historisch gesehen als das erste offizielle XP Projekt angesehen werden kann.
XP beginnt mit vier Grundwerten: Kommunikation, Feedback, Einfachheit und Mut. Der Rest ist eine Ansammlung von meist bekannten und mehrfach verwendeten Praktiken die sich nach Ansicht von Beck besonders eignen. "...none of the ideas in XP are new. Most are as old as programming. There is a sense in which XP is conservative -- all its techniques have been proven..." [4] Beck auf http://clabs.org/xpprac.htm

3.2 Dynamic System Development Method

Die 1995 von 16 Gründungsmitgliedern vorgestellte DSD-Methode [5] beinhaltet ein Framework an, das an sich sehr modular aufgebaut ist, hauptsächlich aber auf 9 essentiellen Prinzipien beruht.

1 Active user Involvement is Imparative
2 Teams must be Empowered to Make Decisions
3 Focus on Frequent Delivery
4 Fitness for Business is Criterion for Accepted Deliverables
5 Iterative and Incremental Development is Mandatory
6 All Changes During Development Must be Reversible
7 Requirements are Baselined at High-Level
8 Testing is Integrated Throughout the Lifecycle
9 Collaborative and Co-operative Approach

Gemäss dem DSDM Konsortium führt die Nichteinhaltung dieser Prinzipien zu einem Bruch mit der Philosophie des Frameworkes und somit zu einem massiv erhöhten Projekt Risiko.

3.3 Die Crystal Familie

Die Hauptphilosophie von Crystal [6] ist, dass Software-Entwicklung als ein kooperatives Spiel von Erfindung und Kommunikation angesehen wird, mit dem Hauptziel, nützliche und funktionierende Software auszuliefern, und dem zweiten Ziel, dafür zu sorgen, dass auch zukünftige Projekte erfolgreich sind.
Die Mitglieder der Crystal Familie gliedern sich nach der Kombination der Anzahl Teilnehmer und dem mit der Software verbundenen Risiko.

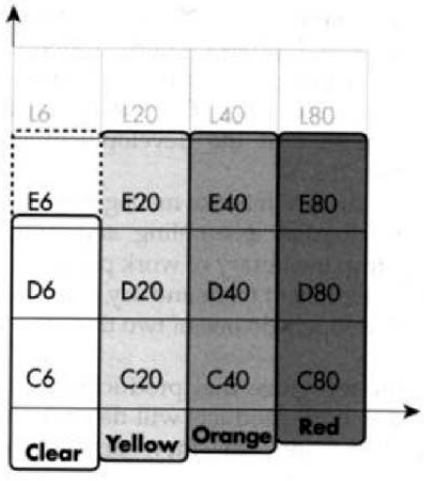

Abbildung 2: Crystal Family

Crystal Clear und Crystal Orange sind dabei die bekanntesten Vertreter.
Crystal Clear beschränkt sich auf kleine Teams und enthält nach Cockburn die wichtigsten Erfolgsfaktoren Kommunikation, häufige Releases, Integration des Kunden und Versionierungskontrolle.
Crystal Orange befasst sich mit grösseren Teams und ist dementsprechend auch mit schwergewichtigeren Methoden bestückt. Auch hier geht es darum, die bereits genannten Erfolgsfaktoren auf ein nun grösseres Projekt anzupassen.
Geradezu typisch für eine agile Methode ist gemäss der grafischen Übersicht ein Einsatz in der Entwicklung lebenskritischer Software nicht geplant – ein weiteres Indiz für die fehlende Marktreife der agilen Methoden im Allgemeinen.

3.4 Adaptive Software Development

ASD behandelt hauptsächlich Probleme, die bei der Entwicklung grosser und komplexer Systeme entstehen [7]. Die Methode beruht auf inkrementeller und iterativer Entwicklung, mit Einbezug von konstantem Prototyping. ASD „balanciert auf der Kante des Chaos“ – das Ziel ist ein Framework anzubieten, das Projekte gerade noch vor dem Chaos schützt, aber das

Entstehen und die Kreativität nicht unterdrücken. Ein ASD-Projekt wird in drei Phasen-Zyklen ausgeführt:
Spekulieren wird an Stelle von Planung verwendet, weil bei einem Plan Unsicherheit als Schwäche angesehen wird und von dem aus Abweichungen einen Misserfolg darstellen.
Zusammenarbeiten hebt die Bedeutung von Teamwork hervor, um sich ständig verändernde Systeme zu entwickeln.
Lernen hebt die Notwendigkeit hervor, Fehler zu erkennen und darauf zu reagieren und die Tatsache, dass Anforderungen sich während der Entwicklung ändern können.

3.5 Scrum

[8] Bei Scrum (ab 1994) handelt es sich um einen leichtgewichtigen Managementprozess, der sich auf Projekte von verschiedenen Grössen anwenden lässt. Die Grundeigenschaften von Scrum sind:

Kleine Teams, welche die Kommunikation und den informellen Informationsaustausch fördern.

Anpassungsfähigkeit bezüglich Veränderungen von Technologie und Benutzeranforderungen.

Häufige Erstellung von neuen Produktversionen, welche inspiziert, angepasst, getestet und dokumentiert werden können.

Aufteilung der zu erledigenden Arbeit in kleine, voneinander möglichst unabhängige Teilaufgaben.

Die Möglichkeit, ein Projekt jederzeit als beendet zu erklären, sei dies aus zeitlichen, finanziellen, wettbewerbstechnischen oder anderen Gründen.

3.6 Feature Driven Development

Die Begründer von FDD sind John deLuca und Peter Coad [7]. Indem man Informationen aus den Modellierungsaktivitäten und allen anderen Anforderungsaktivitäten nimmt, generieren die Entwickler eine Eigenschaftsliste. Als nächstes wird ein Grobplan erstellt und Verantwortlichkeiten werden zugewiesen. Danach werden kleine dynamische Teams gebildet, welche die Eigenschaften in Zeiteinheiten von maximal zwei Wochen implementieren.

4. Überprüfung des (agilen) Erfolges

Die Frage, ob agile Methoden in der Praxis funktionieren, lässt sich nicht ohne weiteres beantworten. Agile Methoden sind sehr neu und nur für die am meisten verbreiteten (XP und DSDM) lassen sich sinnvolle Erfahrungswerte finden. DSDM gehört zu den ältesten agilen Methoden und wurde erst Mitte der Neunziger Jahre definiert. XP, die wohl bekannteste und am meisten verbreitete agile Methode wurde zum ersten Mal in den späten Neunzigern beschrieben und erst im Jahre 2000 als Buch veröffentlicht.
Aufgrund dieser Neuheit ist es schlicht noch nicht möglich zu beweisen, dass agile Methoden in vielen Situationen funktionieren. Selbstverständlich existieren unzählige Anekdoten über jede einzelne Methode die besagt, dass sie funktioniert, aber ein statistischer Beweis aufgrund detaillierter Untersuchungen existiert noch nicht und das wird wohl auch einige Jahre noch so bleiben.

Einige Untersuchungen über gewisse Teilaspekte agiler Methoden wurden gemacht, beispielsweise der Nutzen des „Pair Programming“. Die Daten für umfassende Metriken sind erst im Entstehen, daher ist schlicht und einfach nur Geduld angesagt.
Die im Moment herrschende Diskrepanz resultiert aus den zwei vorhandenen Parteien im Umfeld der agilen Methoden. Es gibt diejenigen, die agile Methoden verwenden und die anderen, die zuerst einen Beweis möchten, bevor sie selbst agile Methoden einsetzen. Geoffrey Moore beschreibt in seinem Buch „Crossing the Chasm“ diesen Zustand mit folgender Grafik:

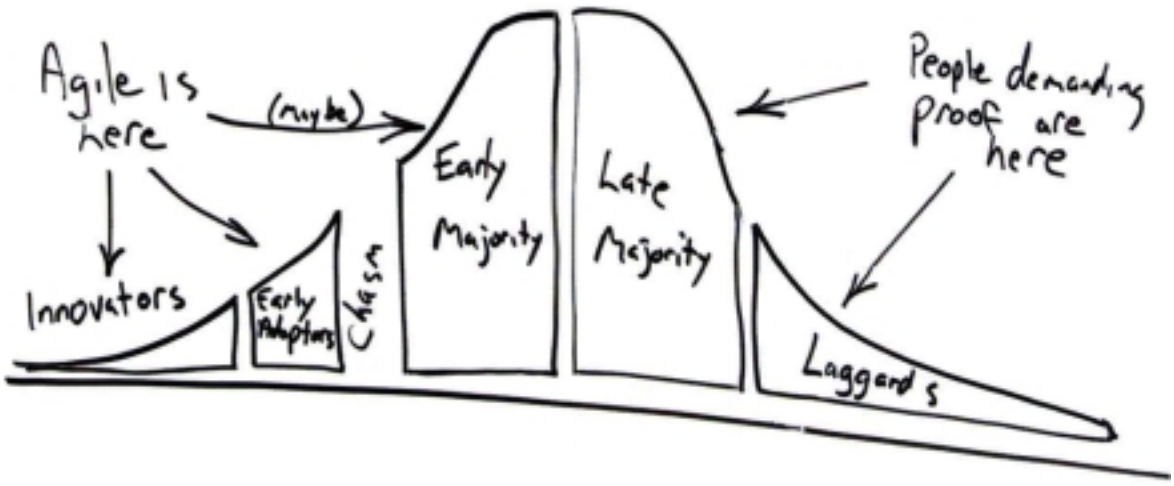

Abbildung 3: Lebenslauf agiler Methoden [9]

Die Frage nach dem Beweis wird typischerweise erst von der späten Mehrheit gestellt, da die agilen Methoden sich aber noch nicht in dieser Phase befinden, ist diese Frage wohl auch zu früh gestellt. Die Innovatoren, die bereits einige Jahre Erfahrung mit agilen Prozessen haben sind klassischerweise Praktiker, die nicht bereit sind, viel Zeit in theoretische Auseinandersetzungen mit ihren Techniken zu investieren. Dies ist ein weiterer Grund, dass verlässliche Untersuchungen wohl noch einige Jahre auf sich warten lassen werden.
Schliesslich ist es auch nicht klar, was überhaupt geprüft werden soll. Agile Ansätze bestechen durch ein volatiles Umfeld, in dem die Erfüllung mehrere Monate alter Pläne ein schlechtes Mass für Erfolg ist. Wenn die Agilität wichtig ist, sollte sie auch gemessen werden. Traditionelle Messweisen bevorzugen die Übereinstimmung mit Plänen. Agile Methoden wollen empfänglich für Änderungen sein. Dies führt zu einem Konflikt: Projekt Manager wollen Flexibilität, messen den Erfolg aber an der Plantreue. Solange agile Methoden mit klassischen Messweisen geprüft werden, kann auch keine sinnvolle Aussage über deren Tauglichkeit gemacht werden.
Scott W. Good meint abschliessend dazu: “There is currently significant anecdotal evidence that agile software development techniques work, you merely need to spend some time on newsgroups and mailing lists to discover this for yourself. If anecdotal evidence isn’t sufficient then agile software development processes aren’t for you... yet!” [9]

5. Ausgewählte Projekte

Das Problem an Fallbeispielen aus der Praxis ist, dass sie kaum existieren. Ausser für die bekanntesten unter den agilen Methoden lassen sich kaum unabhängige und damit sinnvolle Berichte über einen Einsatz in der Praxis finden. Entweder werden horrende Preise für solche Fallstudien verlangt, oder sie sind bis zur Unkenntlichkeit gekürzt worden und für

vorliegende Zwecke ebenfalls unbrauchbar. Damit reduziert sich die Anzahl der agilen Methoden mit brauchbaren Erfahrungsberichten massiv. Im Folgenden wird auf zwei Beispiele aus XP und DSDM näher eingegangen.

5.1 XP: Workshare Technology

Workshare Technology [10] ist eine Firma die sich auf die Entwicklung von Produkten für gemeinsame Dokumentenverarbeitung spezialisiert hat. Mit Niederlassungen in London, San Francisco, New York und Kapstadt erreicht die Firma über 3'500 Unternehmungen in 60 Ländern weltweit.
Die Umstellung auf XP verursachte bei Workshare einige Veränderungen in den Arbeitsabläufen in der Entwicklungsabteilung. Der Umstand, dass das Senior Management sowohl technisch erfahren und dennoch jung und voller Tatendrang war, half der Unternehmung bei der Umstellung sehr. Da diese Umstellung aber dennoch nicht alleine vollzogen werden konnte oder wollte, wurde externe Hilfe beigezogen. Als externer Berater wurde die Firma Object Mentor's Inc. verpflichtet. Die Rolle der Beratungsfirma bestand darin, die Teams zu lehren und ihnen den Einstieg in XP zu vereinfachen. Diese enge Zusammenarbeit fand über mehrere Wochen statt.
Es folgt eine kurze Auflistung, wie die Kernelemente von XP bei Workshare umgesetzt wurden.

1. Extreme Programming Practices – On-site Customer
Die Rolle des Kunden wurde erweitert: sie kontrollieren nun ebenfalls das Design, neue Features und entscheiden über Prioritäten. In Zusammenarbeit mit der Qualitätskontrolle bestimmen sie ebenfalls die „Showstopper" und setzen Prioritäten für die Korrekturen.

2. Extreme Programming Practices – Planning Game
Im Planning Game entscheiden Kunden und Entwickler gemeinsam, welche Arbeit im nächsten Schritt gemacht wird. Diese intensivere Zusammenarbeit hat zum Ziel, die Qualität der Ausgangsdaten, also der Spezifikationen zu erhöhen.

3. Pair Programming
Die Erfahrungen die bei Workshare gemacht wurden, decken sich in etwa mit der von Alistair Cockburn und Laurie Williams publizierten Studie über die Kosten und Nutzen des Pair Programming [11]. Die Fehlerrate wird um 70% gesenkt und der interne Wissensaustausch in der Firma erheblich verbessert.

4. Test First Design
Bei Workshare wird ebenfalls der Test First Ansatz verwendet. Damit, so die Meinung bei Workshare, wird garantiert, dass der Code korrekt ist und nur soviel Code als nötig geschrieben wurde, was einer Optimierung der Arbeit gleichkommt.

5. Refactoring
Refactoring verlangt unter anderem, das Design so einfach wie möglich zu halten und Klarheit in den Code zu bringen, sodass er selbsterklärend ist. Wenn ein Entwickler laut Workshare fünfmal soviel Code liest wie er schreibt, so ist einfach lesbarer Code auch einfacher zu unterhalten.

6. Coding Standards
Über die expliziten Code Standards bei Workshare werden keine Angaben gemacht.

7. Collective Code Ownership
Aufgrund des gemeinsamen Programmierens wird bei Workshare das Problem des unverzichtbaren Spezialisten verringert. Zusätzlich wird ein Refactoring durchgeführt, damit der Code für jeden Entwickler verständlich wird und bleibt.

8. Continuous Integration
Bei Workshare wird die Software automatisch zweimal pro Tag erstellt, was falls nötig jederzeit auch manuell gemacht werden kann. Die letzte Version ist immer verfügbar als Arbeitsprototyp und enthält nur Code, der sämtliche Unit Tests erfolgreich durchlaufen hat. Dies ermöglicht die für agile Methoden notwendige kurze Feedbackzeit des Kunden, da er jederzeit ein aktuelles und lauffähiges System testen kann.

Zusammenfassung:
Der Markt in dem sich Workshare befindet ist in einer schnellen, vermehrt auf vergleichbaren technischen Dokumenten angewiesenen Welt zuhause. Daher müssen das Design und die funktionalen Spezifikationen sehr flexibel sein. Die bis anhin verwendeten Methoden waren nicht mehr skalierbar genug und es wurde eine Methode gesucht, die eine schnellere Entwicklung von Software ermöglicht. Extreme Programming erfüllt nach Ansicht von Workshare all diese Anforderungen.

Diese Aussagen über den Einsatz von XP untermalt Workshare auch mit einigen Daten, die im eigenen Betrieb erhoben wurden. Abbildung 4 zeigt den Wechsel vom alten zum neuen Status Quo inklusive der Phasen der Einführung.

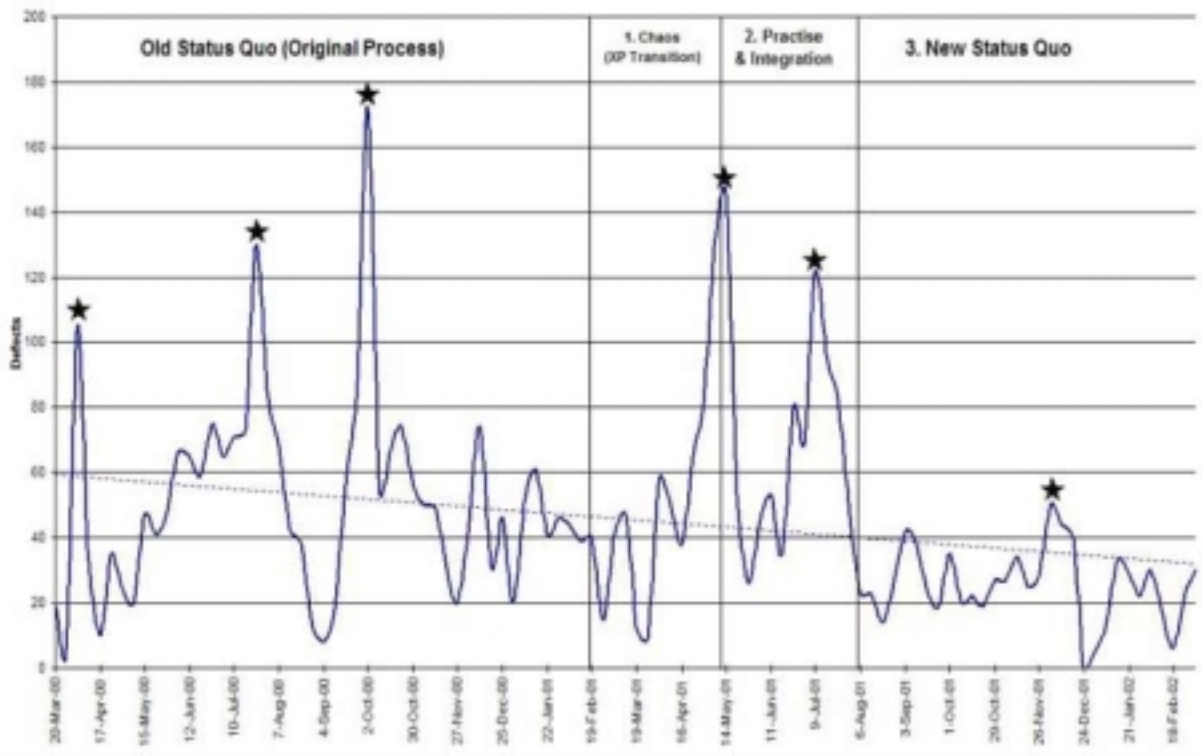

Abbildung 4: Status Quo bei Workshare Technology

Abbildung 5 zeigt die deutliche Senkung der produzierten Fehler pro Entwickler unter Verwendung von XP. Diese Verbesserung resultiert zu einem grossen Teil aus der konsequenten Anwendung des Pair Programming. Dabei ist aber auch zu beachten, dass unter Pair Programming auch weniger Code erzeugt wird. Eine gewisse Senkung war also durchaus zu erwarten, wenn auch nicht so markant wie in der vorliegenden Grafik.

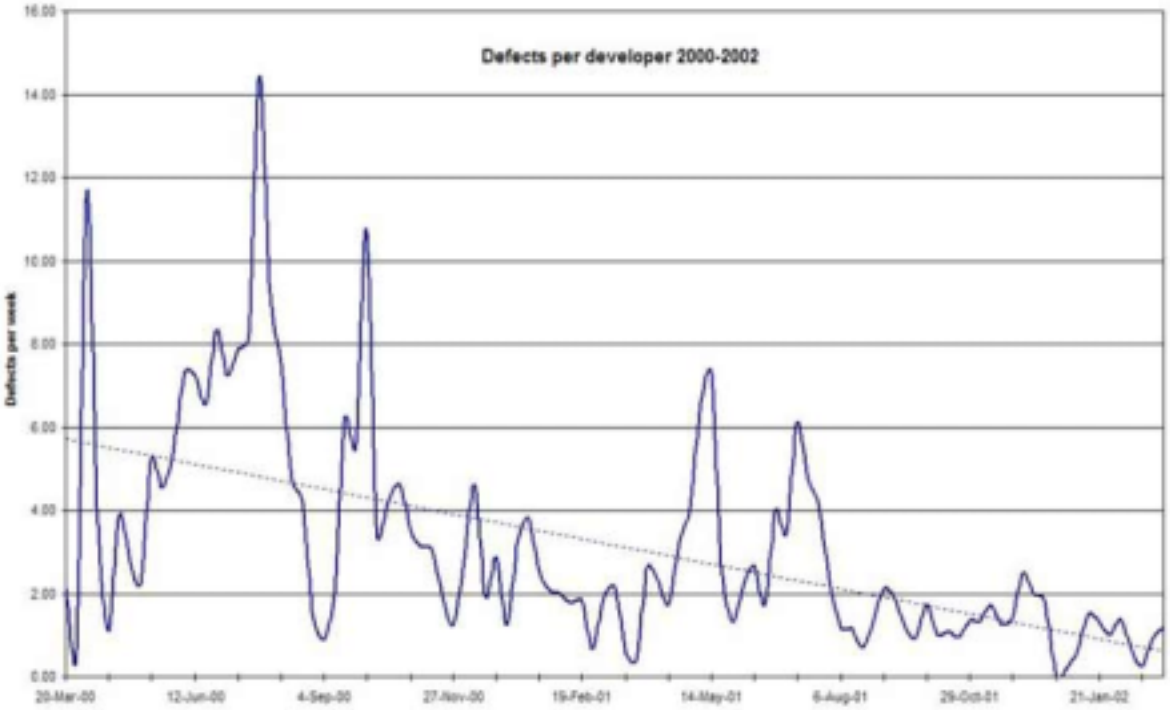

Abbildung 5: Anzahl Defekte pro Woche und Entwickler bei Workshare

5.2 DSDM: Deutsche Bahn

5.2.1 Hintergrund

Die Firma Carmen Systems AB erhielt den Auftrag für die Deutsche Bahn eine Lösung zur optimierten Personaleinsatzplanung zu entwickeln [12]. Das Team bestand aus acht Entwicklern, zwei Benutzern und einem Projektmanager. Die Erstellung persönlicher Einsatzpläne für Langstrecken Zugbegleiter war nur eines von drei Projekten mit einem engen Zeitrahmen. Erschwerend kam noch hinzu, dass die Kundenvertreter (DB) aus ganz Deutschland stammen, die Entwickler ihren Sitz aber in Schweden und Dänemark haben. Als Programmmanager wurde ein versierter DSDM Kenner eingesetzt.

5.2.2 Die neun Prinzipien

1 Active user Involvement is Imparative
Von Beginn weg wurde dem Projekt ein ehemaliger Teamplaner der Deutschen Bahn zugeteilt. Seine Rolle war der Einbezug der anderen Benutzer, Anforderungen zu definieren, priorisieren, reviewen, testen und Lieferungen zu akzeptieren. Ab der zweiten „Timebox“ wurde ein zweiter Benutzer vollumfänglich dem Projekt zugeteilt. Diese sachkundigen und engagierten Benutzer waren einer der Schlüssel zum Erfolg des Projektes

2 Teams must be Empowered to Make Decisions
Teammitglieder wurden für ihre Lieferungen verantwortlich gemacht. Jede Lieferung bedurfte einer Akzeptanz eines vertretenden Benutzers. Reguläre Teammeetings, vor allem bei Iterationen und „timebox“ Reviews waren wichtig für die Messung des Fortschrittes. Da das Projektteam gross war und die Gefahr zu langer Meetings dadurch stieg, nahmen nicht immer alle Mitglieder an diesen Treffen teil. Häufige kurze Meetings wurden wenigen langen vorgezogen.

3 Focus on Frequent Delivery
Eine “Timebox” war üblicherweise vier oder fünf Wochen lang, mit dazwischenliegenden Lieferungen. Im Team war eine allgemeine Übereinstimmung, dass ohne dem aus den Timeboxes resultierenden Fokus das Projekt nicht in der gesetzten Zeit hätte vollendet werden können.

4 Fitness for Business is Criterion for Accepted Deliverables
Da die ursprünglichen objektiven Anforderungsdokumente als ungenügend eingestuft wurden, wurden vom vertretenden Benutzer während der ersten Iteration Testfälle als Anforderungsprototypen erstellt. Durch diese sachkundige Priorisierung wurde eine objektive Sichtweise gewahrt.

5 Iterative and Incremental Development is Mandatory
Der iterative Prototyp Ansatz funktionierte sehr gut. Um gegenseitige Verständlichkeit der Anforderungen zu sichern, war eine enge und regelmässige Kommunikation zwischen dem vertretenden Benutzer und den Entwicklern notwendig.

6 All Changes During Development Must be Reversible
Ein Konfigurationsmanagementsystem wurde eingesetzt, um trotz der regelmässigen Lieferungen die Reversibilität der Änderungen sicherzustellen.

7 Requirements are Baselined at High-Level
In einer frühen Phase wurden etwa 50 High-Level Anforderungen erstellt. Diese wurden in „Timeboxes“ eingeteilt, wo sie auch detaillierter beschrieben wurden. Das Team wurde ermutigt, diese Anforderungen gemäss der priorisierten Anforderungsliste aufzuteilen. Diese Liste wurde für den Projektmanager das wichtigste Werkzeug um den Status von Anforderungen zu verfolgen.

8 Testing is Integrated Throughout the Lifecycle
Testen war absolute essentiell und wurde auch durch den ganzen Lebenszyklus hindurch verfolgt. Aufgrund des zu entwickelnden Systems waren Integration und Regressionstest sehr zeitaufwendige Arbeiten. Dem wurde dadurch Rechnung getragen, dass ein Entwickler nur mit dieser Aufgabe betraut wurde.

9 Collaborative and Co-operative Approach
Zu Beginn des Projektes verstärkte das häufige Testen die typische Kunde – Lieferant Verteilung. Später wurde daraus durch die enge Kommunikation und der zahlreichen Meetings ein echtes Team aus Mitarbeitern der Deutschen Bahn und Carmen. Das wurde als ein wichtiger Nebeneffekt hervorgehoben.

Das Projekt führte letzten Endes zum Erfolg, wenngleich nicht ohne Schwierigkeiten. Wichtig aus Sicht der Beteiligten waren in diesem Fall aber auch auf den ersten Blick weniger relevante Dinge. Da die Entwickler ab einem gewissen Punkt im Projekt hauptsächlich in

Frankfurt arbeiteten und somit für längere Zeit von ihrem privaten Umfeld getrennt waren, wurde viel unternommen, um die Teambildung auch ausserhalb der Arbeitszeit zu fördern. Dies zeigt abschliessend einen zentralen Punkt, der nicht nur DSDM, sondern allen agilen Methoden sehr zugrunde liegt: Die Motivation der Mitarbeiter ist noch wichtiger als bei klassischen Methoden!

6 Diskussion

Abschliessend beantworten lässt sich die Frage der Erfüllung der Anforderungen durch die agilen Methoden also noch nicht. Dennoch seien an dieser Stelle einige grundsätzliche Überlegungen erlaubt. Ziele der agilen Methoden sind letzten Endes die Ressourcen, also Zeit und Geld zu senken und gleichzeitig die Qualität zu maximieren. Dass bei solch schön klingenden Versprechungen auch das eine oder andere schwarze Schaf auftaucht und vom Kuchen etwas haben will ist da schon selbstverständlich. Daher sind die kritischen Faktoren bei Projekten allgemein zu durchleuchten und das zugehörige Verbesserungspotenzial der agilen Methoden zu bewerten.

Faktor Mensch:
Oft werden bei den agilen Methoden die Qualitäten der Mitarbeiter als sehr wichtig klassifiziert. Ein Entwickler soll motiviert, kompetent, teamfähig und kommunikativ sein. Kurzum, es wird ein Idealbild eines Mitarbeiters verwendet, das in der Praxis oftmals nicht anzutreffen ist. Ein Projekt, dessen Humankapital ausschliesslich aus hervorragenden, dem Idealbild entsprechenden Mitarbeitern besteht, wird vermutlich sogar ohne Planung erfolgreich verlaufen.
Weiter wird von den Mitarbeitern ein dynamisches Mitdenken erwünscht. Wenn jeder kritisch seine Meinung auch über fremde Gebiete äussert, können viele Fehler zum Voraus erkannt und beseitigt werden – soweit die agilen Methoden in der Theorie. In der Praxis würden bei grösseren Projekten schnell einmal enorme Verwaltungsaufwände entstehen, will man jede kritische Stimme wirklich voll mit einbeziehen.
Dazu sei ein Vergleich aus dem Leben erlaubt: Eine kleine Gruppe ist auf jeden Mitdenker angewiesen, der Kommunikationsaufwand für den Meinungsaustausch ist gering. In einer sehr grossen Gruppe mit entsprechenden Risiken des Auftrages, beispielsweise in der Armee, ist es nicht möglich, auf jede einzelne Meinung diversifiziert einzugehen. Der Kommunikationsaufwand der dabei entstehen würde wäre enorm und würde jeglichen Fortschritt verunmöglichen. Dies ist wohl der entscheidende Grund, warum agile Methoden in der Regel nur für kleine Teamgruppen empfohlen werden.[*]

Faktor Planung:
Einer der grossen Streitpunkte bei allen agilen Methoden ist der Planungshorizont. Einige proklamieren Kurzsichtigkeit zugunsten von Flexibilität, andere planen auch in agilen Methoden Monate voraus. Welche Methode hier zur Vermeidung eines Debakels führt, kann nicht generell gesagt werden. Der nötige Planungshorizont hängt nicht zuletzt von der Art des Projektes und den Erfahrungen in diesem Gebiet ab. Die Frage bezüglich des Planungshorizontes soll an dieser Stelle mit einem Beispiel verdeutlicht werden.
Jemand plant eine längere Reise in die Ukraine mit dem Auto. Wann soll dieser Jemand wie genau auf die Landkarte schauen und wie detailliert sollte sie sein? Hier spielt die Erfahrung eine grosse Rolle: war die betreffende Person schon einmal in der Ukraine? Weiss er oder sie

[*] *In eigener Sache sei an dieser Stelle noch erwähnt, dass der Schreiber dieses Textes kein Freund der Armee ist, dieses Beispiel aber für treffend hält.*

überhaupt wo dieses Land liegt? Wenn als Startpunkt Zürich angesehen wird, dann ist die geographische Grundrichtung schnell auszumachen. Um diese aber zu finden, muss das Endziel klar bekannt sein. Nur mit der negativ formuliert genannten Kurzsichtigkeit gewisser Methoden (Simple Design) würde der Fahrer Zürich vielleicht gar nie verlassen…

Faktor Feedback
Die nach Ansicht des Verfassers sinnvollste Änderung in den agilen Methoden ist die Forderung nach kurzen Feedbackzyklen und den damit zusammenhängenden Prozessen. Nur durch ständige Rücksprachen mit einem Kunden der Resultate sehen und bewerten kann ist der Erfolg des Projektes im Sinne der Kundenzufriedenheit wesentlich zu steigern. In diesem Punkt liegt auch der wohl markanteste Unterschied zu den klassischen Methoden der Software Entwicklung. Das Endprodukt soll nicht nur fehlerfrei funktionieren, es soll in erster Linie auch das tun, wofür der Kunde zu zahlen bereit ist. So gesehen können die agilen Methoden auch als verkaufsfördernde Werkzeuge angesehen werden.

Agile Methoden sind also mit Sicherheit kein Wundermittel, das jedes Projekt besser, schneller und günstiger zum Erfolg bringen kann. Bei genauem Betrachten stellen sich viele Elemente auch als neu verpackte klassische Ansätze dar, dennoch haben agile Methoden frischen Wind in die Entwicklerumgebung gebracht. Nach subjektiver Einschätzung ist der grösste Verdienst der agilen Methoden im Moment aber die Betonung der Kundenfigur. Dies ist nicht überraschend, da die Softwareindustrie vor allem in der heutigen Zeit eine Methode zu schätzen weiss, die eine hohe Kundenzufriedenheit verwirklichen soll.

7 Abbildungsverzeichnis

8 Quellenverzeichnis

[1] Agile Manifesto, abrufbar unter http://www.agilemanifesto.org/

[2] Pete McBreen: *Top Ten Signs That Your Project Is Just Pretending to Be Agile*, 2002, abrufbar unter http://www.informit.com/content/index.asp?product_id=%7BAC974D56-894D-4814-BD34-53330C477B88%7D

[3] Robert Charette: *Most commonly used agile methodologies*, 2003, abrufbar unter http://www.cutter.com/freestuff/epmu0119.html

[4] Kent Beck: *Kommentar über XP*, 2003, abrufbar unter http://clabs.org/xpprac.htm

[5] Benjamin J. J. Voigt: *Dynamic System Development Method*, Seminararbeit, Institut für Informatik, Universität Zürich, 2004

[6] Silvan Hollenstein und Thomas Rutz: *The Crystal Family*, Seminararbeit, Institut für Informatik, Universität Zürich, 2003

[7] Michael Kauflin: *Agile vs. Klassische Entwicklungsmethoden*, Seminararbeit, Institut für Informatik, Universität Zürich, 2003

[8] Raffael Schweitzer: *Scrum – eine agile Methode zur Software Entwicklung*, Seminararbeit, Institut für Informatik, Universität Zürich, 2003

[9] Scott W. Ambler, *Answering the "Where is the Proof That Agile Methods Work" Question*, 2003, abrufbar unter http://www.agilemodeling.com/essays/proof.htm

[10] David Putman: *Workshare Technology and eXtreme Programming*, 2003, abrufbar unter http://www.objectmentor.com/processImprovement/workshareCaseStudy/xpAtWorkshare

[11] Alistair Cockburn und Laurie Williams, *The Costs and Benefits of Pair Programming*, 2003, abrufbar unter http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

[12] Carmen Systems et al.: *Resource Planning for the railway*, 2003, abrufbar unter http://www.dsdm.org/en/publications/case_studies.asp