

Fakultät für Wirtschaftswissenschaften



Diplomarbeit

Unterstützung agiler Verfahren der Softwareentwicklung durch Open Source-Projektmanagementsoftware

Diplomarbeit zur Erlangung des akademischen Grades
Diplom-Wirtschaftsinformatiker (FH)
der Hochschule Wismar

vorgelegt von Jens Vogel
geboren am 27.03.1967 in Burgstädt
Studiengang Wirtschaftsinformatik, Mat. 10 66 94

Betreuer: Prof. Dr. Herbert Neunteufel
Weiterer Gutachter: Prof. Dr. Jürgen Cleve

Chemnitz, den 17. April 2009

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Begriffsbestimmung	6
1.2.1 Verfahren	6
1.2.2 Projektmanagement	8
1.3 Zusammenfassung	9
1.3.1 Problemfelder	10
1.3.2 Ziele der Diplomarbeit	10
2 Aufgabenstellung	13
2.1 Überblick	13
2.2 Vorgehen	14
3 Stand des Wissens	15
3.1 Klassische Vorgehensmodelle	15
3.1.1 Rational Unified Process	16
3.1.2 V-Modell XT	20
3.2 Agile Verfahren	22
3.2.1 Agile Softwareentwicklung in Deutschland	23
3.2.2 Gemeinsamkeiten agiler Verfahren	25
3.3 Extreme Programming (XP)	31
3.3.1 Überblick	31
3.3.2 Werte	33
3.3.3 Aktivitäten	34
3.3.4 Praktiken	34
3.4 Scrum	38
3.4.1 Überblick	38
3.4.2 Prozessrahmen	40
3.4.3 Rollen	41
3.4.4 Artefakte	43
3.4.5 Regeln	44
3.5 APM	47
3.5.1 Überblick	47

3.5.2	Kleinprojekt	48
3.5.3	APM-Iteration	48
3.5.4	Planungsebenen	48
3.5.5	Projektphasen	49
3.6	Kritik an agilen Verfahren	52
3.6.1	Extreme Programming Refactored	52
3.6.2	Scrum	59
3.6.3	APM	62
3.7	Open Source	63
3.7.1	Open Source als kreative Kraft	63
3.7.2	Motivation der Entwickler	64
3.7.3	Open Source im Unternehmen	65
3.8	Einordnung	68
3.8.1	No Silver Bullet Yet	68
3.8.2	Kleine Teams sind agil	73
3.8.3	Das geeignete agile Verfahren	73
3.9	Zusammenfassung	74
4	Projektmanagementunterstützung agiler Verfahren	77
4.1	Überblick	77
4.2	Besteht Notwendigkeit zur Werkzeugunterstützung?	78
4.3	Anforderungen an Open Source-Projektmanagementssoftware	81
4.3.1	Erfolgsfaktoren	81
4.3.2	Best Practices	83
4.4	Zusammenfassung	88
5	Evaluation von Open Source-Projektmanagementsoftware	91
5.1	Vorgehen	91
5.2	Eingrenzung Open Source-Projektmanagementsoftware	91
5.2.1	Kurzcharakteristik ausgewählter Open Source-Projektmanagementssoftware	92
5.2.2	Entscheidungsalternativen	95
5.3	Nutzwertanalyse Open Source-Projektmanagementsoftware	96
5.3.1	Bewertungskriterien	97
5.3.2	Bewertung Anforderungsmanagement	99
5.3.3	Bewertung Release-/Iterationsplanung	103
5.3.4	Bewertung Praktiken	106
5.3.5	Bewertung Konfigurierbarkeit	109
5.3.6	Bewertung Einfache Handhabung	112
5.3.7	Bewertung Aktivität	113
5.3.8	Auswahl einer Open Source-Projektmanagementsoftware	114
5.4	Zusammenfassung	115

6 Fazit und Ausblick	117
Danksagung	vii
Eidesstattliche Erklärung	ix
Abbildungsverzeichnis	xii
Tabellenverzeichnis	xiii
Glossar	xv
Literaturverzeichnis	xviii
A Abbildungen	xxvii

Kapitel 1

Einleitung

Einen Spaten zum Blumenumtopfen einzusetzen ist so fehl am Platz wie ein Teelöffel zum Ausheben einer Baugrube. Auf das richtige Werkzeug und die richtige Methode kommt es an.

(*Bernd Oestereich*)

1.1 Motivation

In der Geschäftswelt hat die Informatik nun seit gut 4 Jahrzehnten einen festen Platz mit immer größeren Einfluss. Und doch ist die Anzahl bedingt erfolgreicher oder abgebrochener Informatikprojekte immer noch erschreckend hoch. Einen Einblick in diese Problematik bietet der Chaos-Bericht der STANDISH GROUP.¹ So untersucht die Standish Group Softwareprojekte und stellt regelmäßig fest, dass ein anforderungsgerechtes Produkt – wenn überhaupt – oft erst nach gewaltiger Überschreitung von Termin und Budget geliefert wird [Str03]. Seit 1994 wurden im Rahmen dieser Studie über 100.000 Projekte innerhalb der USA auf ihren Erfolg in Bezug auf Umfang, Zeit, Kosten und Qualität hin überprüft. Die untersuchten Projekte werden in drei Kategorien eingeteilt: „Erfolgreich“ (engl.: successful) für Projekte, die alle Anforderungen erfüllen und in Zeit und Budget liegen. Als „Mangelhaft“ (engl.: challenged) werden Projekte eingestuft, die zwar abgeschlossen wurden und im Einsatz sind, deren Entwicklung jedoch in Zeit, Budget oder Leistung nicht im vorgesehenem Umfang lag. Projekte, deren Entwicklung abgebrochen wurde oder deren Ergebnis nie zum Einsatz kam gelten als „fehlgeschlagen“ (engl.: failed). Zwar hat sich im Verlauf der Jahre 1994 bis

¹<http://www.standishgroup.com>

2006 der Anteil erfolgreicher Projekte von 16% auf 35% erhöht, jedoch verbleiben immer noch 65% nicht planmäßig durchgeführte oder abgebrochene Vorhaben [Mül08]. Abbildung 1.1 illustriert diese Entwicklung für die Chaos-Reports 1994 bis 2004, wonach 2004 die Quote erfolgreich abgeschlossener Projekte noch bei 29% lag [HL07]. Eine Verbesserung ist also durchaus zu erkennen.

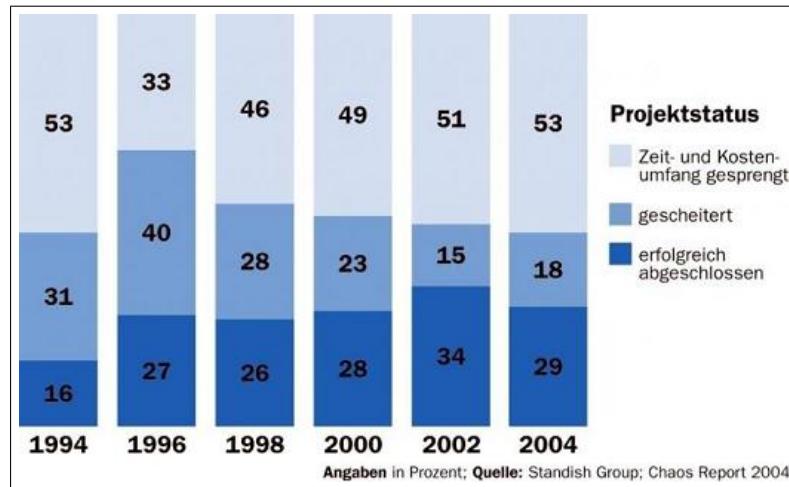


Abbildung 1.1: Chaos-Reports 1994-2004

Zurückführen lassen sich die Verbesserungen der letzten Jahre gemäß STANDISH GROUP auf bessere Software-Entwicklungswerkzeuge, besser geschulte Projektleiter, die strengere Einhaltung von optimierten Management-Prozessen und auf tendenziell kleinere beziehungsweise kürzere Informatikprojekte. Danach sind die wichtigsten Kriterien für den Projekterfolg: Einbeziehung der Benutzer, Unterstützung durch das Management und klare Anforderungen. Unvollständige Anforderungen und fehlende Einbeziehung der Benutzer sind demnach die wesentlichen Faktoren, die zum Scheitern von Projekten führen [Sta95]. Als Gründe für die Fehlschläge nennt MÜLLER [Mül08] vor allem Mängel im Projektmanagement und im Entwicklungsprozess. Weitere Studien bestätigen dieses Ergebnis. Nach einer Studie, die gemeinsam von TECHREPUBLIC und GARTNER GROUP vorgelegt wurde, scheitern ca. 40% aller IT-Projekte [Boo00].

2001 wurde auf einem Treffen von 17 Vertretern verschiedener iterativ-inkrementeller Ansätze der Begriff *Agile* geprägt. Auf diesem Treffen entstand das agile Manifest, das gewissermaßen einen gemeinsamen Nenner agiler Softwareentwicklung darstellt². Die agile Softwareentwicklung will einerseits die Vorteile und Errungenschaften klassischer Verfahren erhalten. Andererseits will sie die Begrenzungen dieser auflösen und darüber hinausgehen [Oos08a].

²<http://agilemanifesto.org/history.html>, 28.12.2008

Frage 1 Für Softwareunternehmen stellt sich somit die Frage, ob Softwareprodukte mit Hilfe agiler Verfahren Erfolg versprechender entwickeln können.

Der Softwareentwickler oder Softwarearchitekt sieht sich mit einer enorm großen Vielzahl von Techniken und Methoden konfrontiert, deren Ansätze teils diametral auseinandergehen, was die Entscheidung für oder gegen den Einsatz einer spezifischen Technik zum Risikofaktor werden lässt. Softwareentwicklung in kleinen iterativen Schritten ist nicht nur in agilen Verfahren essentieller Bestandteil. BROOKS hebt die Bedeutung der Nachverfolgung (engl.: tracking) des Projektfortschritts in Form von kleinen individuellen Meilensteinen hervor [Bro03].

Es kann dabei festgestellt werden, dass die Ansätze insbesondere an der Frage der dem Softwareentwicklungsprozess zu Grunde liegenden Anforderungen auseinander klaffen. In der einen extremen Ausprägung wird dabei von der Erwartung ausgegangen, alle Anforderungen zu Beginn vorliegen zu haben. Gerade bei großen Projekten gerät das schnell zu einer endlosen Anforderungsanalyse. Das andere Extrem – der Versuch weitgehend ohne dokumentierte Anforderungen auszukommen – ist zumindest im Hinblick auf die entstehende Softwarearchitektur fragwürdig [Sta08]. BROOKS führt aus, dass es eine der größten Herausforderungen der Softwareentwicklung ist, genau das zu erfassen, was der Kunde wirklich will [Bro87]. Und in der Tat weiß der Kunde meist nicht was er will, bevor er nicht einen ersten Entwurf ausprobiert hat. Obwohl zur Anforderungsanalyse und deren Management zahlreiche Methoden und Techniken existieren und diese hinreichend publiziert sind, stehen Projektleiter und Entwickler vor einer erdrückenden Vielfalt, so dass eine objektive Entscheidung nur schwer getroffen werden kann.

Des weiteren ist zu einigen Methoden und Verfahren ein umfangreiches Rundumwissen erforderlich. Speziell für den Rational Unified Process (RUP) und für vergleichbare klassische Ansätze gilt: „Eine Einführung des Prozesses mit allen unterstützenden Tools macht für kleine oder einmalige Projekte sicherlich wenig Sinn. Sind die Projekte größer und wird der RUP für mehrere Projekte verwendet gibt er die Möglichkeit wirtschaftlich und effektiv Software zu entwickeln.“ [F+05] OESTEREICH betont die Tatsache, dass Methoden wie Fachkonzepte, Dokumentationen, Quellcodierichtlinien, Versionsverwaltung usw. keinen nebensächlichen Selbstzweck erfüllen ([Oes98] S. 23).

Frage 2 Es ist also die Frage zu stellen, warum es angesichts der Vielzahl von Techniken, Methoden und Vorgehensmodellen in der Unternehmenspraxis dennoch große Schwierigkeiten bei der Durchführung von Softwareprojekten gibt?

Zur Frage, was agile Softwareentwicklung ist, wird auf der Homepage der AGILE ALLIANCE³ ausgeführt: Ende der 1990-er Jahre traten verschiedene neue Methoden in das öffentliche Interesse. Jede dieser Methoden war eine spezielle Kombination alter, gewandelter und neuer Ideen. Doch alle diese Methoden heben die Bedeutung der engen Zusammenarbeit zwischen dem Programmierteam und den Experten der Fachdomäne hervor: direkte Kommunikation ist effektiver als geschriebene Dokumentation; die weiter entwickelte Software wird in kurzen Zyklen ausgeliefert und schafft sofort Nutzwert für den Kunden; kleine, sich selbst organisierende Teams; und Methoden, den Code und das Team so zu gestalten, dass die unvermeidlichen Anforderungsänderungen nicht in eine Krise führen.

Die Betonung liegt auf der enormen Bedeutung der engen Zusammenarbeit zwischen dem Programmierteam und den Experten der Fachdomäne. So ist in den letzten Jahren die Komponente „Mensch“ stärker in den Fokus der Teamarbeit in Projekten – also auch Softwareprojekten – gerückt. Dies drückt sich auch in der im IT-Umfeld häufig verwendeten Bezeichnung „soft skills“ (deutsch: weiche Fähigkeiten) aus. So werden seit einiger Zeit speziell zu diesem Thema Beratung, Coaching und Seminare im IT-Umfeld angeboten, was die Bedeutung des Themas für Projektarbeit unterstreicht.⁴ Das Aspekte wie Kommunikation, Vertrauen und Teammotivation in den letzten Jahren zunehmend Bedeutung erlangt haben, zeigt der Themenschwerpunkt der OOP 2009: „Soft(ware) Skills: The key to successful projects“.⁵

Vor allem für kleine und mittlere IT-Unternehmen scheint deshalb der agile Ansatz Erfolg versprechend. Schließlich wird bei diesen Methoden die Rolle des Softwareentwicklers stärker betont. Auch werden dort typischerweise meist kleine Projekte von kleinen Projektteams realisiert. ABENDROTH führt dazu die GFK-STUDIE WIRTSCHAFTSFORSCHUNG 2000 an, wonach von insgesamt 10.568 Unternehmen der Primärbranche (Softwareentwicklung) 8.173 Unternehmen (77%) über eine Unternehmensgröße von 1-9 Mitarbeitern verfügt [Abe01]. Investitionen in entsprechende Softwaretools für klassischen Methoden sowie in die dann erforderliche Wissensaneignung werden von den Unternehmen oft gescheut. Klassische Methoden, wie der Rational Unified Process , werden zwar hervorragend durch Software unterstützt, die Lizenzkosten für diese Werkzeuge belasten den Budgetrahmen kleiner Softwareunternehmen jedoch erheblich. So liegt beispielsweise der Preis für die Floating-Lizenz der IBM-Software *Requisite Pro* bei 4.491 Euro [VW06]. Außerdem gelten Schnelligkeit, Flexibilität und ein günstiges Preis-Leistungs-Verhältnis unter wachsendem Konkurrenzdruck als Erfolgsfaktoren.

³<http://agilealliance.org/show/2>, 29.12.2008

⁴<http://www.oose.de/soft-skills/seminare.html>, 05.04.2009

⁵<http://oop2009.com>, 29.12.2008

Frage 3 Eine weitere Frage ist, wie geeignet agile Verfahren in Bezug auf die Teamgrößen sind. Dadurch, dass die Kommunikation in agilen Projekten eine wesentliche Funktion einnimmt, kann vermutet werden, dass aufgrund der zunehmenden Interkommunikationskomplexität⁶ ab etwa 8 Teammitgliedern agile Verfahren nicht mehr sinnvoll angewendet werden können.

BOEHM entwickelt in [Boe06] eine hegelianische Sicht auf den Prozess der Softwareentwicklung. Er stellt Eingangs die Hypothese auf, dass dessen Dialektik auch für die Softwareentwicklung von den 1950-er Jahren bis heute gilt. Demnach wird solange mit funktionierenden Thesen (im Kontext der Softwareentwicklung: Methoden, Prozesse, Vorgehensmodelle) gearbeitet, bis festgestellt wird, dass die Methoden und Prozesse in einigen wichtigen Punkten versagen und sich ein Ansatz entwickelt, der besser funktioniert (Antithese). Schließlich bezieht sich der so entwickelte neue Ansatz verstärkt auf die ursprüngliche These, so dass die Verbindung der positiven Aspekte beider Ansätze zu einem Dritten führt (Synthese), der die ursprünglichen Nachteile vermeidet ([Boe06] S. 12). Für die aktuelle Dekade bezeichnet BOEHM die Antithese und teilweise Synthese mit *Agilität und Wert* (engl.: Agility and Value) ([Boe06] S. 18).

Frage 4 Gibt es ein agiles Verfahren, dass – speziell in Deutschland – mit Erfolg angewendet wird, und kann dieses Verfahren in der Entwicklung von Softwaresystemen in kleinen Teams eingesetzt werden?

Nicht zuletzt motiviert mich meine eigene Erfahrung mit der Entwicklung von kleinen bis mittleren Softwaresystemen zu einer eingehenderen Beschäftigung mit dieser Problematik. Die Anwendung klassischer Verfahren ist gerade in kleinen Organisationseinheiten schwierig, steht oft in einem schlechten Verhältnis zur Projektgröße und ist auch aus Kostengesichtspunkten dem Management schwer zu vermitteln. Der leichtgewichtige Charakter agiler Verfahren erscheint daher als vernünftige Alternative. Hinzu kommt das praktische Interesse im Bereich der Open Source-Software, nach geeigneter Software zur Unterstützung des Managements agiler Verfahren zu suchen.

Frage 5 Letztlich ist zu fragen, ob ein Softwaresystem gefunden werden kann, das unbeschränkte Verwendbarkeit⁷ garantiert und somit den spezifischen Bedürfnisse des jeweiligen agilen Verfahrens angepasst werden kann, sowie den Entwicklungsprozess agiler Softwareprojekte im Sinne des Projektmanagements unterstützt und somit einen Wert an sich darstellt?

⁶ $\frac{n(n-1)}{2}$

⁷ im Sinne freier Software (Vgl. Seite 63)

Fazit

Nach wie vor scheitern in der Softwareentwicklung viele Projekte oder werden nicht innerhalb des geplanten Zeit- und Kostenumfangs abgeschlossen.

Wesentliche Faktoren für den Erfolg von Softwareprojekten sind

- Einbeziehung der Benutzer,
- Unterstützung durch das Management,
- klare Anforderungen.

Als wesentliche Faktoren für den Misserfolg sind

- Mängel im Projektmanagement und
- Mängel im Entwicklungsprozess

zu nennen.

Obwohl die Techniken, Methoden und Vorgehensweisen für die Herstellung guter Software bekannt sind, ist die Quote erfolgreich abgeschlossener Projekte nach wie vor relativ gering.

Agile Verfahren greifen die Vorteile klassischer Verfahren auf und versuchen die Nachteile, die diese mit sich bringen, zu vermeiden. Der Kommunikation der Projektbeteiligten kommt dabei eine außerordentliche Bedeutung zu.

1.2 Begriffsbestimmung

Einige wesentliche Begriffe, die in dieser Arbeit Verwendung finden, werden in der Literatur nicht immer einheitlich verwendet. In diesem Abschnitt wird deshalb eine Definition und Einordnung dieser Begriffe vorgenommen.

1.2.1 Verfahren

Für die Bearbeitung des Themas dieser Diplomarbeit wird der Begriff *Verfahren* verwendet. Die Verwendung dieses Begriffs ist erforderlich, da ansonsten keine

einheitliche begrifflich Einordnung möglich ist. Ein Verfahren bezeichnet ebenso ein Vorgehensmodell als auch eine Sammlung von Methoden.

Vorgehensmodell

Vorgehensmodelle zählen zu den Referenzmodellen. Es stellt ein Entwurfsmuster als Modell für die zu entwickelnden Sachverhalte zur Verfügung. STAHL-KNECHT/HASENKAMP definieren den Begriff folgendermaßen:

„Allgemein beschreibt jedes Vorgehensmodell die Folge aller Aktivitäten, die zur Durchführung eines Projektes erforderlich sind. Vorgehensmodelle für die Systementwicklung von Anwendungssystemen geben an, wie die Prinzipien, Verfahren und Werkzeuge der Software-Entwicklung [...] einzusetzen sind.“ [SH05]

Stärker gerichtet auf den Aspekt Entwicklungsteam lautet die Definition von LEFFINGWELL/WIDRIG etwas kompakter:

„Your team's software development process *defines who (which member of the team) is doing what (what activity is being performed [sic], when (that activity is done in relation to other activities), and how (the details and steps in the activity) in order for your team to reach its goal.*“ [LW00] S. 213

Ein *Vorgehensmodell* kann allgemein folgendermaßen charakterisiert werden:

- Wer erledigt
- was
- wann
- wie
- um das Teamziel zu erreichen?

Methode

„Eine Methode [...] beschreibt eine Vorgehensweise für einen bestimmten Anwendungsbereich. Beispielsweise kann die Beschreibung

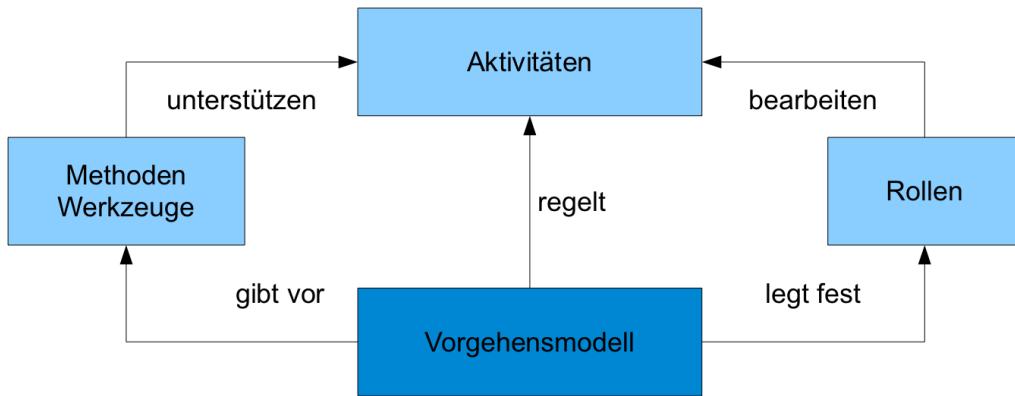


Abbildung 1.2: Allgemeine Darstellung Vorgehensmodell

von Anforderungen mit Hilfe von Anwendungsfällen als Methode bezeichnet werden. Die Beschreibung von Anforderungen mit Hilfe von Testfällen wäre eine andere Methode.“ [Oes09]

Demnach kann Extreme Programming (XP) eher als Methodensammlung denn als Vorgehensmodell bezeichnet werden.

Definition Verfahren

Der Begriff Verfahren wird in der vorliegenden Diplomarbeit definiert als:

Verfahren sind Vorgehensmodelle oder Sammlungen von Methoden, die in agil oder klassisch durchgeführten Softwareprojekten angewendet werden.

1.2.2 Projektmanagement

Nach der DIN 69901 wird unter Projektmanagement die „Gesamtheit von Führungsaufgaben, Führungsorganisation, Führungstechniken und -mitteln für die Projektabwicklung“ verstanden. Dabei ist der Begriff „Führung“ als die Steuerung der verschiedenen Aktivitäten im Projekt in Hinblick auf die übergeordneten Projektziele zu interpretieren. Die Definition nach DIN stellt dabei besonders die

Verbindung zwischen „Was ist zu tun“ mit „Wer macht das“ und „Wie wird das gemacht“ heraus.⁸

Das *Project Management Institute (PMI)*⁹ geht über obige Definition hinaus und definiert Projektmanagement als:

„Project Management is the application of knowledge, skills, tools and techniques to project activities to meet project requirements.“¹⁰

Das PMI bezieht demnach Werkzeuge und Techniken ausdrücklich in die Definition mit ein.

Im Sinne der Agilität kann Projektmanagement dreierlei bedeuten:

1. Management agiler Projekte.
2. Agiles Management von Projekten.
3. Agiles Management von agilen Projekten.

Definition Projektmanagement

In der vorliegenden Diplomarbeit wird der Begriff Projektmanagement im letzten Sinne – in Anlehnung an die PMI-Definition – definiert:

Projektmanagement ist die agile Anwendung von Wissen und Fertigkeiten, Werkzeugen und Techniken bei der Abwicklung agiler Projekte.

1.3 Zusammenfassung

Im Abschnitt 1.1 wurde auf die Motivation zu dieser Diplomarbeit eingegangen. Die dort herausgearbeiteten Fragestellungen führen schließlich zu den Problemfeldern, die im Rahmen dieser Diplomarbeit bearbeitet werden.

⁸http://www.management-knowhow.de/index.php?id=10#_index_P, 29.12.2008

⁹<http://www.pmi.org/Pages/default.aspx>

¹⁰http://de.wikipedia.org/wiki/Projektmanagement#Definitionen_Projektmanagement, 29.12.2008

1.3.1 Problemfelder

Verfahren

Das für effiziente und effektive Softwareentwicklung Methoden und Prozesse in Form von Verfahren unabdingbar sind scheint unbestritten. Jedoch sind schwergewichtige, klassische Ansätze in kleinen Projektteams schwierig zu etablieren. Auch vor dem Hintergrund kleiner bis mittlerer Projekte steht der erwartete Nutzen einem zu großen Aufwand gegenüber. Agile Prozesse, mit schneller Wertschöpfung, versprechen dieses Dilemma zu lösen. Inwieweit die Trennung zwischen klassischen und agilen Ansätzen überhaupt Gültigkeit besitzt oder ob es auch hier zu einer Synthese kommt wird im Kapitel 3 untersucht. Darüber hinaus werden einige klassische sowie agile Verfahren genauer betrachtet und bewertet.

Projektmanagement

Zu einem nicht zu vernachlässigenden Teil sind die Gründe für das vollständige oder teilweise Scheitern von Softwareprojekten im mangelhaften Projektmanagement zu suchen. Nach möglichen und sinnvollen Führungs- und Steuerungsmechanismen des Projektmanagements für die relevanten Verfahren wird im Kapitel 4 gesucht.

Werkzeugunterstützung

Der Einsatz eines geeigneten Softwaretools, mit dem das Projektmanagement wirksam unterstützt werden kann, bildet das dritte Problemfeld. Dieses wird im Abschnitt 5.3 bearbeitet und eine geeignete Open Source-Projektmanagementsoftware ermittelt.

1.3.2 Ziele der Diplomarbeit

1. Erarbeitung des aktuellen Standes zu insbesondere in Deutschland eingesetzten agilen Verfahren im Vergleich zu klassischen Verfahren.
2. Diskussion und Bewertung der identifizierten Problemfelder agiler Methoden.

3. Ableitung der Anforderungen, die sich aus den Problemfeldern ergeben.
4. Evaluation Open Source-Projektmanagementsoftware und Auswahl eines Werkzeugs, das zur Unterstützung der identifizierten Anforderungen geeignet ist und sich optimal in den agilen Entwicklungsprozess einbettet.

Kapitel 2

Aufgabenstellung

Ich bin bereit, überallhin zu gehen, wenn es nur vorwärts ist.

(David Livingstone)

2.1 Überblick

Im Kapitel 3 werden einige klassische Methoden vorgestellt, auf agile Verfahren allgemein sowie auf ausgewählte agile Verfahren vertiefend eingegangen. Dabei werden die behandelten agilen Verfahren einer kritischen Betrachtung im Kapitel 3.6 unterzogen. Der Abschnitt 3.8 fasst das bis dahin Gesagte zusammen und gibt erste Antworten auf die in der Einleitung (Abschnitt 1.1) gestellten Fragen:

Frage 1 *Können Softwareunternehmen, bei steigender Komplexität der Produkte und einem äußerst dynamischen Markt, Softwareprodukte mit Hilfe agiler Verfahren Erfolg versprechender entwickeln?*

Frage 2 *Warum gibt es angesichts der Vielzahl von Techniken, Methoden und Vorgehensmodellen in der Unternehmenspraxis dennoch große Schwierigkeiten bei der Durchführung von Softwareprojekten?*

Frage 3 *Wie geeignet sind agile Verfahren in Bezug auf die Teamgrößen? Kann die Vermutung, dass agile Verfahren ab etwa 8 Teammitgliedern nicht mehr sinnvoll angewendet werden können, bestätigt werden?*

Frage 4 *Gibt es ein agiles Vorgehen, das – speziell in Deutschland – mit Erfolg angewendet wird und kann dieses Verfahren in der Entwicklung von Softwaresystemen in kleinen Teams eingesetzt werden?*

Die Letzte in der Einleitung aufgeworfene Frage (Vgl. Seite 5) wird durch Erarbeitung der Anforderungen und Evaluation relevanter Systeme aus dem Bereich Open Source-Projektmanagementssoftware im Abschnitt 4.3 und Kapitel 5 beantwortet. Die Frage lautet:

Frage 5 *Kann ein Softwaresystem gefunden werden, das uneingeschränkte Verwendbarkeit¹ garantiert und somit den spezifischen Bedürfnisse des jeweiligen agilen Verfahrens angepasst werden kann, sowie den Entwicklungsprozess agiler Softwareprojekte im Sinne des Projektmanagements unterstützt und somit einen Wert an sich darstellt?*

2.2 Vorgehen

Untersuchung von agilen Verfahren und Klärung des aktuellen Wissensstandes im Abschnitt 3.2. Dabei wird aus Gründen der Vergleichbarkeit im Abschnitt 3.1 auch auf bedeutende klassische Verfahren eingegangen.

Situation Open Source vor dem Hintergrund des wirtschaftlichen Einsatzes in Unternehmen im Abschnitt 3.7.

Theoretische Erarbeitung der Anforderungen an eine Software zur optimalen Unterstützung des Managements agiler Projekte im Kapitel 4. Dabei wird speziell die Situation kleiner Teams berücksichtigt.

Praktische Evaluation von Systemen aus dem Bereich Open Source-Projektmanagementsoftware auf Basis der erarbeiteten Anforderungen kleiner Teams im Kapitel 5.

¹im Sinne freier Software (Vgl. Seite 63)

Kapitel 3

Stand des Wissens

Obwohl ein Vorgehensmodell geradezu nach einer Anpassung schreit, hält sich das Projekt sklavisch an das nicht angepasste Standardvorgehen.

(Tom DeMarco)

3.1 Klassische Vorgehensmodelle

Im Folgenden werden einige klassische Softwareentwicklungsmodelle beschrieben, die vor allem durch breiten Einsatz von Werkzeugen und Dokumentation gekennzeichnet sind und sich schon allein dadurch von agilen wesentlich unterscheiden. Die Darstellung verfolgt vor allem vergleichende Zwecke, d.h. die beschriebenen klassischen Modelle stellen primär eine Reflexionsfläche für die Untersuchung der agilen Methoden dar.

Auf ältere klassische Verfahren wird im Einzelnen nicht eingegangen. Diese Methoden sollten in der Praxis keine oder eine eher untergeordnete Rolle mehr spielen. Zu nennen ist hier das *Wasserfallmodell* nach [Roy70], das auf Grund seiner Steifigkeit als fehlerträchtiges und mit hohem Kostenrisiko behaftetes Modell gilt. Allerdings wurde (und wird) das Wasserfallmodell oft als rein sequentieller Prozess aufgefasst, was teilweise im Zusammenhang mit Festpreisprojekten zu sehen ist. Bei diesen Projekten baut ein Prozessschritt auf dem vorhergehenden auf – Softwaredesign erst nach abgeschlossener Anforderungsanalyse und Codierung wiederum erst danach ([Boe06] S. 15). Nach [OW08] legte David Maibor, Mitautor des US-Militär-Standards DoD-STD-2167, der sich auf die Arbeit von Winston Royce bezieht, den Grundstein für dieses Missverständnis. Maibor war demnach mit iterativ-inkrementeller Entwicklung nicht vertraut und verwendete

eine stark vereinfachte Variante des Wasserfallmodells. Das *Spiralmodell*, 1985 von [Boe88] publiziert, wurde zwar theoretisch viel beachtet, wird in der Praxis jedoch selten angewendet. Es kann als Weiterentwicklung des Wasserfallmodells verstanden werden, da es dieses um die dort fehlenden Iterationen ergänzt [Bal97].

3.1.1 Rational Unified Process

Der Rational Unified Process basiert auf der Zusammenarbeit der als die ‚drei Amigos‘ bekannten Autoren Ivar Jacobson, Grady Booch und James Rumbaugh. Deren Ansätze vereinten sich sowohl zur Unified Modeling Language (UML) als auch zum Rational Unified Process (RUP), welcher durch die Firma Rational, später unter dem Dach der IBM, weiterentwickelt wurde (siehe Abbildung A.1).

Best Practices

Der Rational Unified Process vereint als Softwareentwicklungsprozess (Software Engineering Process) folgende Best Practices [Kru99]:

Iterative Softwareentwicklung: Durch den iterativen Ansatz des Rational Unified Process können Anforderungsänderungen berücksichtigt werden. Die Integration findet nicht am Ende der Entwicklung sondern kontinuierlich statt. Dadurch werden Projektrisiken frühzeitig verringert und das Management hat die Möglichkeit, taktische Änderungen vorzunehmen. Auch die Wiederverwendung von Software wird erleichtert. Schließlich entsteht durch den iterativen Ansatz eine robustere Architektur.

Anforderungsmanagement: Das Anforderungsmanagement ermöglicht es anstehende Änderungen zu erkennen, zu organisieren und vor allem zu managen. Komplexe Projekte können besser kontrolliert werden. Die Software-Qualität steigt und dadurch wird eine höhere Kundenzufriedenheit erreicht. Verzögerungen werden reduziert und damit sinken die Projektkosten.

Komponentenbasierte Architektur: Die Steuerung des Rational Unified Process findet durch Use Cases (Anwendungsfälle) statt. Das Design ist dabei auf die Architektur gerichtet. Der Rational Unified Process bietet einen methodischen und systematischen Ansatz zum Design, zur Erstellung und zur Validierung der Softwarearchitektur.

Modellierung mit UML: Das System wird grafisch mit Diagrammen der UML beschrieben. Damit können Systemausschnitte beliebiger Tiefe in diversen Ansichten modelliert und transparent gemacht werden.

Prozess- und Produktqualität: Durch den Test-Workflow konzentriert sich der Rational Unified Process auf das Verifizieren und Messen, ob das Produkt die erforderliche Qualität erreicht hat oder nicht. Es werden entsprechende Prozess- und Produktmetriken bereitgestellt.

Änderungsmanagement: Softwareentwicklung erfordert ein hohes Maß an Flexibilität. Mit dem Rational Unified Process werden Änderungen in wichtigen Teilen des Systems verfolgt und synchronisiert. Änderungsmanagement bedeutet hierbei die Verwaltung aller Änderungen im Hinblick auf Anforderungen, Design und Implementierung sowie die entsprechenden Maßnahmen bei auftretenden Fehlern und Missverständnissen.

Der Rational Unified Process kann durch drei wesentliche Merkmale charakterisiert werden. Diese sind:

Use-Case Driven: Der Prozess verwendet Use Cases, um die Entwicklung von Beginn an voran zu treiben.

Architekturzentriert: Der Prozess sucht nach den wichtigsten Aspekten für die Architektur. Die Architektur ist die technische Sicht auf das Ganze und spiegelt die Bedürfnisse der Nutzer wieder. Die zentrale Rolle kommt den Use Cases zu (Use-Case-Sicht).

Iterativ und Inkrementell: Der Prozess zerlegt große Projekte in kleinere Projekte oder Mini-Projekte. Jedes Mini-Projekt besteht aus einer Iteration, die eine Verbesserung des Produkts ergibt. Die Iterationen werden mit Hilfe von Use Cases geplant. Dabei stellt jede Iteration einen „Wasserfall“ mit den RUP-typischen Prozessphasen dar.

Prozessstruktur

Die Abbildung 3.1 stellt den Rational Unified Process generell dar. Horizontal ist der zeitliche Verlauf und vertikal die inhaltliche Struktur dargestellt. Der Rational Unified Process besteht aus den Phasen *Inception* (engl.: Anfang; hier: Konzeption), *Elaboration* (engl.: Ausarbeitung; hier: Design/Entwurf), *Construction* (engl.: Ausführung; hier: Implementierung/Test) und *Transition* (engl.: Überleitung; hier: Übergabe), die sich über den gesamten Systemlebenszyklus wiederholen können. Jede Iteration wird mit einem Release abgeschlossen, Releases innerhalb einer Iteration sind aber ebenfalls möglich.

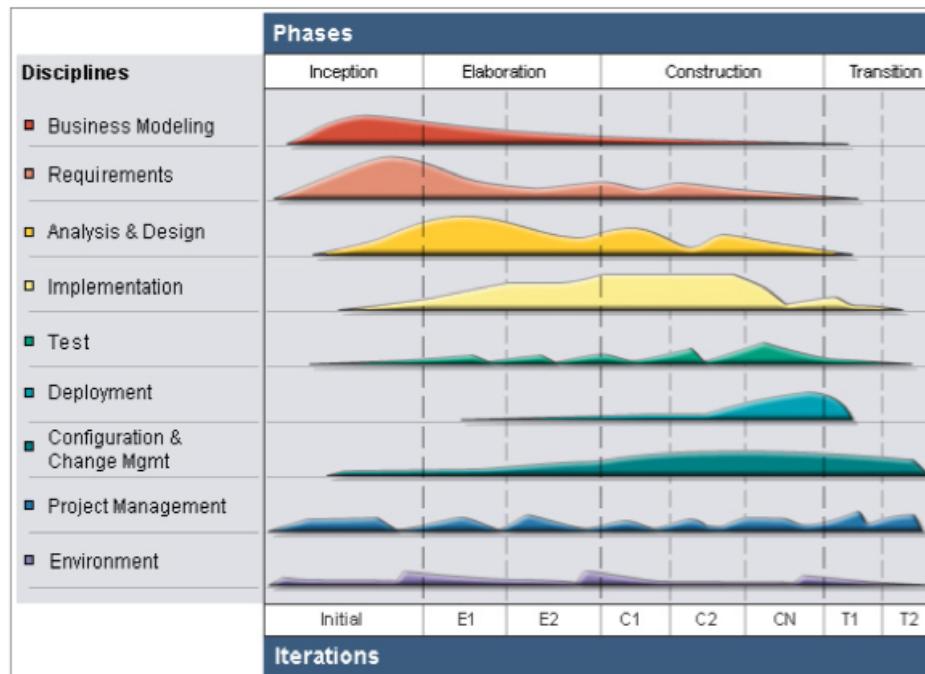


Abbildung 3.1: Rational Unified Process Prozessstruktur [PEF⁺07]

Inception Phase: Während der Inception Phase wird aus der Kernidee eine Produktvision entwickelt. In dieser Phase wird das Produkt konzipiert und die zugrunde liegenden Geschäftsvorfälle spezifiziert. Der Umfang des Projektes wird festgelegt sowie Kosten und Risiken geschätzt. Diese Phase wird als Konzeptionsphase bezeichnet.

Elaboration Phase: In dieser Phase werden die Produkteigenschaften spezifiziert und das Design der Architektur festgelegt. Dazu werden die meisten Use Cases detailliert spezifiziert. Der Fokus dieser Phase liegt darauf, zu erkennen, ob das Projekt wie konzipiert realisiert werden kann. Die notwendigen Aktivitäten und die benötigten Ressourcen werden festgelegt. Diese Phase wird als Design- bzw. Entwurfsphase bezeichnet.

Construction Phase: Die Produktkomponenten werden in dieser Phase entwickelt, getestet und in das fertige Produkt integriert. Diese Phase wird als Implementierungsphase bezeichnet.

Transition Phase: Das Produkt wird an den Benutzer übergeben. Die erreichte Qualität wird überprüft. Es schließt sich die Auslieferung, Schulung, Anpassung und Wartung an. Diese Phase wird als Übergabephase bezeichnet.

Core Workflows

Der Rational Unified Process definiert *Core Workflows*, die während des Entwicklungsprozesses eingesetzt werden. Dazu gehören: Business Modeling, Requirements, Analysis, Design, Implementation, Test und Deployment (siehe Abbildung 3.1). Die Workflows sind nicht sequentiell und werden mit unterschiedlicher Intensität in allen vier Phasen abgearbeitet.

Prozessschwergewicht

Der Rational Unified Process und seine Dokumentation erfordert eine gründliche Beschäftigung mit dem gesamten Prozess. Dazu gehört neben dem Erwerb von theoretischem Wissen ebenso formelles Training. Ein Mentor innerhalb der eigenen Organisation kann dabei hilfreich sein. Der dazugehörige Aufwand sollte nicht unterschätzt werden.

Der Rational Unified Process selbst kann dabei nicht als starrer Prozess betrachtet werden. Er muss zum Arbeitsumfeld, zu den Arbeitsgewohnheiten und zur Projektgröße passen.

Für den Prozess, der selbst frei zugänglich ist, existieren eine Reihe von Werkzeugen der Firma Rational (IBM)¹, die den Rational Unified Process ideal unterstützen. Allerdings setzt der Einsatz dieser Werkzeuge ein nicht unerhebliches Investment voraus.

Zusammenfassung

Der Rational Unified Process zählt zu den ausgereiftesten Vorgehensmodellen und kann – bei richtiger Anwendung – den Softwareentwicklungsprozess insgesamt verbessern. Auf Grund seiner Komplexität ist es jedoch schwierig, den Prozess korrekt anzuwenden:

¹<http://www-01.ibm.com/software/rational/>

”However, unless you have a real expert on-staff it is likely that you will not significantly increase your likelihood of success trying to adopt this process. The process is too complex, too difficult to learn, and too difficult to apply correctly. If you don’t have an expert, an expert who has actually delivered similar projects using this process, then either hire or rent one and plan to engage the expert for at least one year.” ([Mel02] S. 6)

3.1.2 V-Modell XT

Das V-Modell entstand ursprünglich 1997 als V-Modell 97, das unter diesem Namen keine Weiterentwicklung mehr erfuhr. Durch den verbindlichen Einsatz dieses Modells im öffentlichen Bereich verbreitete es sich auch in Unternehmen, bis hin zu KMUs.

Die Weiterentwicklung des Modells zu dessen Nachfolger *V-Modell XT* zielte im wesentlichen auf eine Verbesserung in den Bereichen Anpassung und Erweiterung, Anwendbarkeit, Skalierung. Das Modell liegt zurzeit als Version 1.3 vor². Auf die verbesserte Anpassbarkeit des Verfahrens verweist das Kürzel XT – eXtreme Tailoring. Dabei wurden aktuelle Normen und Vorschriften sowie aktuelle Technologien berücksichtigt. Im Rahmen von Entwicklungsprojekten wurde der Anwendungsbereich auf die Betrachtung des gesamten Lebenszyklus erweitert.

Das V-Modell XT ist modular aufgebaut. Die modularen Elemente werden als *Vorgehensbausteine* bezeichnet. Ein Vorgehensbaustein kapselt Rollen, Produkte und Aktivitäten. Dabei werden Produkte zu Produkt- und Aktivitäten zu Aktivitätsgruppen zusammengefasst. Ein Vorgehensbaustein wird als eine Einheit betrachtet, die eigenständig verwendet werden kann. Vorgehensbausteine können unabhängig voneinander verändert und weiterentwickelt werden.

Für die Durchführung eines Projektes sind *Projektdurchführungsstrategie* und *Entscheidungspunkte* ausschlaggebend. Die Projektdurchführungsstrategie definiert die Reihenfolge der zu erreichenden Projektfortschrittsstufen. Ein Entscheidungspunkt ist ein genau festgelegter Zeitpunkt im Projektplan, zu dem über den weiteren Projektfortgang („go“ oder „no go“) entschieden wird. Dazu wird auch festgelegt, welche *Produkte* zu diesem Zeitpunkt fertig gestellt werden sein

²<http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/1.3, 12.02.2009>

müssen. Dadurch entsteht die grundlegende Struktur des Projektverlaufs. Die Produkte sind letztlich die Projektergebnisse, auf deren Basis die detaillierte Projektplanung vorgenommen wird. Dabei ist die Verantwortung für jedes Produkt über eine Rolle definiert, so dass die dafür verantwortliche Person im Projekt feststeht. Durch definierte Anforderungen an das Produkt und beschriebene Abhängigkeiten zu anderen Produkten kann die Produktqualität überprüft werden.

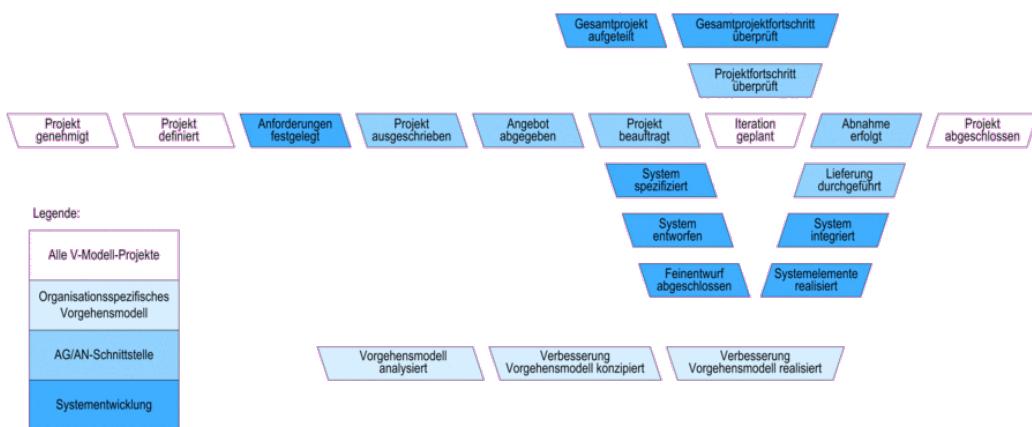


Abbildung 3.2: V-Modell XT

Die Abbildung 3.2 zeigt das V-Modell XT mit seinen möglichen Projekttypen³ [Rau06]. Durch Auswahl des Projekttyps, der anzuwendenden Vorgehensbausteine und der Durchführungstrategie mit den dazugehörigen Entscheidungspunkten wird für den konkreten Fall der Umfang des V-Modell XT festgelegt – das so genannte Tailoring. Mit Projektmerkmalen wird ein Anwendungsprofil erstellt. Daraus ergeben sich die verpflichtend zu verwendenden Vorgehensbausteine und die möglichen Projektdurchführungsstrategien. So gelangt man schrittweise zum Ausgangspunkt für die konkrete Projektplanung.

Das V-Modell XT enthält dadurch auch iterativ-inkrementelle und agile Projektdurchführungsstrategien, so dass im Rahmen dieses Modells agile Techniken innerhalb von IT-Projekten der öffentlichen Hand eingesetzt werden können ([OW08] S. 13).

Im Jahr 2005 wurde das V-Modell XT in einigen Pilotprojekten verwendet. Die Ausschreibung zu IT-WiBe, einem Softwaresystem für Wirtschaftlichkeitsbetrachtungen in der Bundesverwaltung, wurde nach V-Modell durchgeführt und innerhalb des vorgegebenen Zeit-

³siehe Legende der Abbildung 3.2

rahmens abgeschlossen. Die Software selbst wurde ebenfalls gemäß V-Modell XT entwickelt. Allerdings muss berücksichtigt werden, dass das beteiligte V-Modell-Team Erfahrung in der Anforderungserstellung mit Anwendungsfällen hatte. Außerdem waren Mitarbeiter des Teams, das für die Weiterentwicklung des Verfahrens verantwortlich ist, in das Projekt eingebunden. In einem ungeschulten Team wären vermutlich einige Schwierigkeiten zu erwarten gewesen [NR05].

3.2 Agile Verfahren

In der Softwareentwicklung bezeichnet Agilität (lat. *agilis* ‚flink, beweglich‘) einen flexibleren und schnelleren Softwareentwicklungsprozess, als das bei den klassischen Verfahren der Fall ist. Basierend auf Werten genügen einige wenige Regeln, die dafür um so disziplinierter befolgt werden (sollten), um den Softwareentwicklungsprozess zu strukturieren.

Die Ende der 1990-er Jahre entstandenen agilen Verfahren bauen einerseits auf dem iterativ-inkrementellen Ansatz älterer Modelle auf und stellen andererseits eine Gegenbewegung zu den überreglementierten und starren Ansätzen dieser Modelle dar [OW08].

In seiner Keynote (Key5) auf der OOP2009 benennt DEMARCO vier Aspekte, die die Leistungsfähigkeit jedes IT-Unternehmens ausmachen:

Leistungsfähigkeit: Die richtigen Dinge richtig machen.

Geschwindigkeit: Das oben Genannte schnell machen (schneller als die Konkurrenz).

Agilität: In der Lage zu sein, die Richtung zu ändern, wenn das Ziel sich ändert.

Unternehmenskultur: Der Klebstoff, der eine Organisation zusammen hält.

„Doch dabei kann folgendes beobachtet werden: die meisten Dinge, die wir studieren, lesen und lernen, handeln von Leistungsfähigkeit; die anderen drei Aspekte werden meist ignoriert. Bei Vorgehensmodellen und Prozessen geht es um Leistungsfähigkeit, bei Qualitätssicherung geht es um Leistungsfähigkeit und auch bei den so genannten agilen Methoden geht es mehr um Leistungsfähigkeit als um die Fähigkeit, auf veränderte Ziele einzugehen.“ (DeMarco [DeM09])

3.2.1 Agile Softwareentwicklung in Deutschland

Im Januar 2008 hat das Fachblatt *OBJEKTSPEKTRUM*⁴ gemeinsam mit *it-agile*⁵ eine Online-Umfrage zum Thema „Stellenwert agiler Softwareentwicklung in Deutschland“ durchgeführt [WR08].

Wie die Autoren ausführen, erhebt die Studie keinen streng wissenschaftlichen Anspruch, lässt aber durchaus Rückschlüsse auf den aktuellen Stand der agilen Softwareentwicklung in Deutschland zu. Es ist zwar zu vermuten, dass diejenigen, die sich mit Agilität beschäftigen, überrepräsentiert sind, aber dennoch ein breiter Durchschnitt der deutschen IT-Landschaft an der Studie teilgenommen hat.

Der Bekanntheitsgrad agiler Methoden ist demnach in Deutschland sehr hoch. Insgesamt kennen 93 % der Befragten agile Softwareentwicklung im Allgemeinen, davon planen 12 % den Einsatz, 28 % haben erste Erfahrungen gemacht und 36 % gehen bereits agil vor. Dabei ist *eXtreme Programming (XP)* zwar die bekannteste Methode, *Scrum* und *FDD* wird jedoch häufiger erfolgreich eingesetzt. Interessanterweise wenden Scrum 21 % erfolgreich an, XP jedoch nur 14 %. Die Autoren bezeichnen damit Scrum als die Methode, die am erfolgreichsten eingesetzt wird und begründen dies mit „der Tatsache, dass XP sehr mächtig ist, sich aber nur schwer einführen und durchhalten lässt“. Andere Verfahren spielen nur eine untergeordnete Rolle.

Befragt zu den agilen Techniken, die verwendet werden, sind die drei meist genannten mit 48 % „priorisierenden Produktverantwortlichen/Kunden“, 52 % „kurze Releasezyklen“ und 50 % „inkrementelle Auslieferung“.

Befragt nach der Wirkung von Agilität nennen die Teilnehmer am häufigsten⁶ „Flexibilität“ mit 89 %, „Spaß bei der Arbeit“ mit 88 % und „Lernen bei der Arbeit“ mit 87 %. Die „bessere Einhaltung von Terminen und Budget“ ist mit 55 % bzw. 49 % für viele Teilnehmer erstaunlicherweise nicht eine Folge agiler Softwareentwicklung, legen agile Methode doch gerade auf diese beiden Aspekte großen Wert.

Nach der Studie scheint agiles Vorgehen nach wie vor eine Domäne kleiner und mittler Projekte zu sein. 27 % sind der Meinung agiles Vorgehen eignet sich nicht für große Projekte, 21 % wollen oder können sich dazu nicht festlegen und 52 % meinen agiles Vorgehen funktioniert auch in großen Projekten. Obwohl mehr als die Hälfte

⁴<http://www.sigs-datacom.de/sd/publications/os/index.htm>

⁵<http://www.it-agile.de/>

⁶Mehrfachnennungen waren möglich

der Befragten positiv zu agilen Verfahren in großen Projekten stehen interpretieren die Autoren die Zahlen mit folgender Begründung⁷:

„Wenn man sich die Zahlen zur Frage ‚Glauben Sie, dass Agilität folgendes bewirkt‘ ansieht, dann wird deutlich, dass meist über 75% der Befragten an positive Wirkungen von Agilität glauben. Aber bei der Frage nach großen Projekten sind es ‚nur‘ 50%. Unsere Aussage ist daher relativ zu sehen: Im Vergleich dazu, wie positiv die Wirkungen von Agilität eingeschätzt werden, ist der prognostizierte Erfolg für große Projekte verhältnismäßig gering.“

Diese Auffassung scheint sich auch durch die typischen Teamgrößen zu bestätigen. Danach arbeiten mehr als 70 % der Teilnehmer in Projekten mit maximal zehn Entwicklern.

Auch die Frage, ob agiles Vorgehen Chaos und/oder fehlende Projektverantwortlichkeit bewirkt, haben nur 14 % der Teilnehmer mit „Ja“ beantwortet. Die Autoren begründen dies „mit der Tatsache, dass agile Prozesse zwar sehr wenige, dafür aber umso klarere Regeln vorgeben.“ Außerdem ist die Zufriedenheit der Teilnehmer mit den eigenen Projekten bei denjenigen größer, die bereits erfolgreich Agilität einsetzen.

Zusammenfassend konstatieren die Autoren, „agile Softwareentwicklung ist keine Spielwiese für risikofreudige Hacker (mehr), sondern sie wird erfolgreich in industriellen Projekten in verschiedenen Bereichen eingesetzt.“ Der Nutzen des agilen Vorgehens fällt jedoch nicht so hoch aus wie erwartet – hier sprechen die Autoren von Optimierungsbedarf.

Im Rahmen dieser Diplomarbeit können folgende Aussagen aus der vorgelegten Studie abgeleitet werden:

- Scrum ist das erfolgreichste agile Verfahren in Deutschland. XP gilt als zu komplex.
- Kleine Projekte, die von kleinen Teams bearbeitet werden, scheinen besser für agile Verfahren geeignet.
- Trotz der überwiegend positiven Bewertung werden agile Methoden nur selten vollständig eingesetzt.
- Agile Verfahren führen sich nicht selbst (automatisch) ein und durch. Hier gibt es Optimierungsbedarf.

Das hat auf das weitere Vorgehen folgende Auswirkungen:

⁷E-Mail der Autoren der zitierten Studie Arne Rook und Henning Wolf

- Die vorhandenen agilen Verfahren werden nicht alle gleichermaßen untersucht. Vielmehr konzentrieren sich die Abschnitte 3.3, 3.4 und 3.5 auf die Verfahren *XP*, *Scrum* und *APM*.
- Die in Abschnitt 1.1 getroffene Behauptung, dass der agile Ansatz vor allem für kleine und mittlere IT-Unternehmen mit entsprechenden Projektgrößen Erfolg versprechend ist, kann vorerst bestätigt werden.
- Möglicherweise liegt dies an den in agilen Verfahren geringer ausgeprägten Rahmenbedingungen und der Verwendung von Techniken aus anderen, traditionellen Verfahren (z.B. UML). Dem wird in Abschnitt 3.5 Rechnung getragen.
- Die Frage nach Optimierungsbedarf leitet zu Kapitel 4 und Abschnitt 5.3, in denen nach praktischer Unterstützung durch Projektmanagementsoftware gesucht wird.

3.2.2 Gemeinsamkeiten agiler Verfahren

Den agilen Verfahren gemeinsam ist der Versuch, den Problemen der klassischen Ansätze entgegenzuwirken. Die Probleme – insbesondere bei nicht-iterativen Prozessen – können folgendermaßen charakterisiert werden:

- Die Komplexität traditioneller Verfahren ist für kleine Projekte zu groß.
- In der Planungsphase muss großer Aufwand betrieben werden.
- Änderungen von Anforderungen während der Realisierung sind nur mit großem Aufwand zu realisieren⁸.
- Teammitglieder werden als Funktionseinheit betrachtet⁹

Die agilen Verfahren versuchen diesen Problemfeldern zu begegnen, indem:

- Software mit sehr kurze Iterationszyklen erstellt wird.
- Anforderungsänderungen zur täglichen Praxis gehören und die Änderungskosten im Laufe des Entwicklungsprozesses sogar geringer werden [Bec03].

⁸BOEHM beschreibt in [Boe81], dass die Änderungskosten im Lauf des Softwarelebenszyklus exponentiell steigen.

⁹Der sprachliche Niederschlag des „Funktionierens“ eines Mitarbeiters kann leicht durch eine Suche in einschlägigen Suchmaschinen gezeigt werden. Der Suchbegriff „Mitarbeiter funktioniert“ ergibt bei Google 647 und bei Yahoo 952 Treffer (Suchanfrage vom 05.12.2008)

- Die menschliche Komponente stärker Berücksichtigung findet.

Obwohl sich alle agilen Verfahren in den letzten 5 - 7 Jahren praktisch weiterentwickelt haben, allgemeiner sowie theoretisch fundierter geworden sind, gibt es kein agiles Standardverfahren [OW08]. Neben persönlichen Vorlieben liegen die Ursachen in der überwältigenden Vielfalt der Methoden und der konkreten Situation, in der sich das Projekt und Team befindet: „Wie schon der klassische Beraterspruch, wenn nach einer Lösung gefragt wird, sagt: ‚It depends‘“ [Pau08]. Ein weiterer Grund ist die Verbreitung bestimmter Verfahren innerhalb nationaler oder sprachlicher Grenzen. So ist der agile Ansatz *DSDM Atern* nur in Großbritannien verbreitet, in Deutschland jedoch weitgehend unbekannt [Sah08].

Auch für agile Verfahren ist bei der Einführung eines Verfahrens oder Modells eine typische Konsequenz der Versuch, mit dem Verfahren allen Situationen gerecht zu werden [Eck09]. ECKSTEIN führt weiter aus: „*Das tatsächliche Vorgehensmodell* [...] ist das Vorgehensmodell, das tatsächlich eingesetzt und gelebt wird. Typischerweise gibt es davon keine Beschreibung, da jeder glaubt, etwas anderes zu befolgen, und doch befolgen alle Projektmitarbeiter genau dieses eine tatsächlich Vorgehensmodell. [...] Letztendlich ist es gerade dieses tatsächliche Vorgehensmodell, das am Ende Software produziert.“

Das agile Manifest

Im Jahr 2001 wurde das agile Manifest¹⁰ von Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland und Dave Thomas ins Leben gerufen. Der Wortlaut des Manifestes ([OW08] S. 15):

„Wir entdecken bessere Wege zur Entwicklung von Software, indem wir Software entwickeln und anderen bei der Entwicklung helfen. Durch diese Tätigkeiten haben wir gelernt:

- Individuen und Interaktion sind wichtiger als Prozesse und Werkzeuge
- Funktionierende Software ist wichtiger als umfangreiche Dokumentation

¹⁰<http://www.agilemanifesto.org/>, 14.12.2008

- Kooperation mit Projektbetroffenen ist wichtiger als Vertragsverhandlungen
- Reaktion auf Änderungen ist wichtiger als Festhalten an einem starren Plan

Natürlich sind auch die Dinge rechts wichtig, aber im Zweifelsfall schätzen wir die linken höher ein.“

Das agile Manifest stellt den zentralen Ausgangspunkt der agilen Verfahren dar. Auf dessen Basis ist die auf Werten begründete agile Methodik überhaupt erst vorstellbar. Die einzelnen Aussagen reichen also weit in die durchaus stark differierenden agilen Ansätze hinein, so dass dazu einige Erläuterungen sinnvoll erscheinen [OW08]:

Individuen und Interaktion sind wichtiger als Prozesse und Werkzeuge. Agile Methoden können als „Menschenzentriert“ bezeichnet werden. Das Team gilt als der Erfolgsfaktor. Das wird vor allem durch die große Verantwortung, die agilen Teams übertragen wird, deutlich. Nur das Nötigste muss geplant werden. Starr reglementierte Prozesse fehlen, so dass sich die Kreativität des Teams verantwortungsvoll entfalten kann.

Funktionierende Software ist wichtiger als umfangreiche Dokumentation. Nur wirklich notwendige Dokumentation wird erstellt. Gute Kommunikation reduziert die Notwendigkeit dafür. Direktes Feedback wird durch kurze Releasezyklen möglich, so dass die Anforderungsdokumentation entsprechend kurz ausfallen kann.¹¹

Kooperation mit Projektbetroffenen ist wichtiger als Vertragsverhandlungen. Einer der agilen Grundsätze ist die enge Zusammenarbeit mit dem Kunden. Das Vertrauensverhältnis zwischen den Projektbeteiligten bestimmt nachhaltig den Entwicklungsprozess. Die zu realisierenden Features legt der Kunde verantwortungsvoll mit fest. Durch rechtzeitiges Feedback wird das Risiko, falsche Software zu entwickeln, minimiert. Die Basis ist Vertrauen, so dass die Bedeutung detaillierter Verträge sinkt.

¹¹BECK vertritt in [Bec03] den Standpunkt, dass das System im Wesentlichen durch den Code dokumentiert ist. Dass dieser Standpunkt durchaus umstritten ist, wird in *Extreme Programming Refactored: The Case Against XP* [SR03] deutlich. Darauf wird im Abschnitt 3.6.1 weiter eingegangen.

Reaktion auf Änderungen ist wichtiger als Festhalten an einem starren Plan. Auf Änderungen, die sich in Softwareprojekten meist ergeben, soll flexibel reagiert werden. Die Planung und Anforderungsdefinition erfolgt evolutionär, mit schrittweiser Verfeinerung. Die Gesamtplanung ist auf das Sicherheitsbedürfnis der Beteiligten abgestimmt.

Prinzipien agiler Verfahren

Das agile Manifest mit seinen vier Kernaussagen wird durch zwölf Prinzipien konkretisiert. Diese sind in den verschiedenen agilen Methodiken unterschiedlich ausgeprägt. HÜTTERMANN führt diese Prinzipien in [Hüt08] weiter aus und gelangt zum Teil zu äußerst prinzipiellen Aussagen (die der Autor selbst als schwarz-weiß Darstellung bezeichnet), auf die im Abschnitt 3.6 näher eingegangen wird:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. Kundenzufriedenheit ist demnach höchstes Ziel der Softwareentwicklung. Um dies zu gewährleisten, ist es notwendig, frühzeitig und kontinuierlich lauffähige Software zur Verfügung zu stellen. Der Kunde wird dadurch in die Lage versetzt, rechtzeitig Feedback zu geben und so die Entwicklung zu beeinflussen.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. In den meisten traditionellen Projekten wird versucht, die Architektur und Planung vor der Implementierung durchzuführen. Änderungsanforderungen während der Implementierungsphase sind schwierig und nur mit großen Aufwand zu verwalten und zu realisieren. Die einzelnen Phasen folgen mehr oder weniger streng aufeinander. In agilen Projekten sind diese Phase eng miteinander verknüpft und laufen fast gleichzeitig ab. Hier wird es sogar begrüßt, wenn Anforderungsänderungen während der eigentlichen Entwicklung bekannt werden.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale. Da agile Projekte sowohl *iterativ* als auch *inkrementell* arbeiten,

können lauffähige Zwischenlösungen – das Inkrement – in kurzen, regelmäßigen Zeitspannen bereitgestellt werden. In traditionellen Projekten wird Software oft erst zum vereinbarten Termin ausgeliefert.

Business people and developers work together daily throughout the project. In agilen Projekten arbeiten Entwicklerteam und Kunde sehr eng zusammen. Der Kunde gilt als Domain-Experte, der sich mit den Anforderungen auskennt. Durch die ständige Verfügbarkeit des Kunden und die enge, partnerschaftliche Zusammenarbeit werden die vagen Anforderungen zu Beginn des Projektes im Laufe der Entwicklung weiter konkretisiert. Im Gegensatz dazu wird in traditionellen Projekten das so genannte *Big Design Up Front (BDUF)*, also das vollständige Design der Lösung zu Beginn, entworfen.

Build projects round motivated individuals, give them the environment and support they [sic] need and trust them to get the job done. Die Individualität der Teammitglieder, insbesondere der Programmierer, wird in agilen Projekten hervorgehoben. Dadurch und indem diese alle Unterstützung und Werkzeuge erhalten, die sie zum Erreichen ihrer Ziele benötigen, wird von einer Motivation ausgegangen, die sich positiv auf den Projektverlauf auswirkt. In nichtagilen Projekten herrscht hingegen oft demotivierende Gleichschaltung.

The most efficient and effective method of conveying information with and within a development team is face-to-face conversation. Agile Entwicklung basiert zu einem großen Teil auf Vertrauen zwischen den Partnern. Deshalb ist es notwendig, dass Kommunikation *face-to-face* – also persönlich stattfindet.

Working software is the primary measure of progress. In agilen Projekten wird davon ausgegangen, dass Artefakte wie Spezifikationen und Dokumentationen nur Mittel zum Zweck sind. Demnach ist „die Software das Artefakt, das der Kunde in Auftrag gibt und das er so geliefert haben möchte, dass er es einsetzen kann. [...] In agilen Projekten zeigt man, was man bisher entwickelt hat, anstatt es zu erklären oder zu beschreiben.“

Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely. Die Arbeit in agilen Projekten ist gleichmäßig verteilt. Daraus resultiert ein gleichmäßiges Arbeitstempo, das sich auch darin ausdrückt, dass Überstunden vermieden werden. Jede Stunde im Projekt wird als gleich wichtig wahrgenommen. Die Belastung der Softwareentwickler ist mit der aeroben Belastung eines Ausdauersportlers vergleichbar. Permanente Überbelastung (Dauerlauf im anaeroben Bereich) führt näher an Herzinfarkt und Familienzwist.

Continuous attention to technical excellence and good design enhances agility. Werden durch die Programmierer während ihrer Arbeit an der Umsetzung von Anforderungen Fehler oder Mängel am Design festgestellt, wird dieses augenblicklich verbessert. Dazu werden kontinuierliche *Refactorings* eingesetzt. Unterstützt wird dieses Prinzip durch kollektives Eigentum am Code, testgetriebene Entwicklung und paarweises Programmieren.

Simplicity – the art of maximization the amount of work not done – is essential. Mit minimalem Aufwand ein optimales Ergebnis zu erzielen ist schon aus betriebswirtschaftlichen Erwägungen heraus erstrebenswert und gleichwohl eine Kunst (ist das Streben des Kunden doch ein genau umgekehrtes). In agilen Projekten wird versucht, für eine konkrete Aufgabenstellung die minimale Lösung zu erarbeiten.¹² Diese einfache Lösung verringert Komplexität und trägt so zum guten Design bei. Es wird also nicht versucht, zukünftige Anforderungen bereits zu Beginn der Entwicklung durch generische Lösungen zu berücksichtigen.

The best architectures, requirements and designs emerge from self-organizing teams. Indem dem Entwicklerteam in agilen Projekte große Verantwortung übertragen wird, verteilen diese die Arbeit selbstständig untereinander. Durch enge Zusammenarbeit und Kooperation wird das Design und die Architektur verbessert. Selbstorganisierende Teams entstehen nicht von selbst. Sie benötigen einen entsprechenden Handlungsrahmen, eine Umgebung, die ungestörtes Arbeiten erlaubt. Diese Bedingungen muss der Projektmanager schaffen.

¹²Was die Entwicklung von Frameworks deutlich erschweren dürfte.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. Agile Projekte werden in regelmäßigen Abständen einer so genannten *Retrospektive* unterzogen. Dies sollte mindestens nach Abschluss jeder Iteration der Fall sein. Aufgetretene Schwierigkeiten werden gemeinsam sachlich analysiert um herauszufinden, wie die Arbeit des Teams verbessert werden kann. Die Ergebnisse fließen dann in die folgende Iteration ein.

3.3 Extreme Programming (XP)

3.3.1 Überblick

Ende der 1990-er Jahre wurde *Extreme Programming (XP)* im Rahmen des *Crysler Comprehensive Compensation System (C3)* [A⁺98] im Wesentlichen von Kent Beck, Ron Jeffries und Ward Cunningham entwickelt [Hüt08]. Dieses Projekt ist deshalb so überaus bedeutend, da in diesem Projekt XP unter großer öffentlicher Beachtung angewendet wurde. Trotz Abbruch des Projekts am 01.02.2000¹³ wird das Projekt von den XP-Befürwortern als Erfolg gesehen. Von den Kritikern wird dieser Abbruch natürlich als Scheitern betrachtet.¹⁴ Auf die Kritik an XP wird im Abschnitt 3.6 eingegangen.

BECK führt zur Frage *Was ist XP?* aus:

„XP ist eine leichte, effiziente, risikoarme, flexible, kalkulierbare, exakte und vergnügliche Art und Weise der Softwareentwicklung. XP unterscheidet sich von anderen Methoden durch:

- frühzeitige, konkrete und fortwährende Feedbacks durch kurze Zyklen
- einen inkrementellen Planungsansatz, bei dem schnell ein allgemeiner Plan entwickelt wird, der über die Lebensdauer des Projekts hinweg weiterentwickelt wird
- die Fähigkeit, die Implementierung von Leistungsmerkmalen flexibel zu planen und dabei die sich ändernden geschäftlichen Anforderungen zu berücksichtigen

¹³<http://www.c2.com/cgi/wiki?ChryslerComprehensiveCompensation>, 19.12.2008

¹⁴Für eine eingehendere Beschäftigung mit dieser Thematik verweise ich auf [SR03] und die Postings im C2-Wiki, <http://www.c2.com/cgi/wiki?CthreeProjectTerminated>, 19.12.2008

- die Abhängigkeit von automatisierten Tests, die von den Programmierern und den Kunden geschrieben werden, um den Entwicklungsfortgang zu überwachen, das System weiterzuentwickeln und Mängel frühzeitig zu erkennen
- das Vertrauen darauf, dass mündliche Kommunikation, Tests und Quellcode die Struktur und den Zweck des Systems zum Ausdruck bringen
- die Abhängigkeit von einem evolutionären Designprozess, der so lange andauert, wie das System besteht
- das Vertrauen auf die Zusammenarbeit von Programmierern mit ganz gewöhnlichen Fähigkeiten
- die Abhängigkeit von Verfahren, die sowohl den kurzfristigen Instinkten der Programmierer als auch den langfristigen Interessen des Projekts entgegenkommen

XP ist eine Disziplin der Softwareentwicklung. Es ist eine Disziplin, weil es bestimmte Dinge gibt, die man tun muss, wenn man XP einsetzen will. Man kann es sich nicht aussuchen, ob man automatisierte Tests schreibt oder nicht – wenn man es nicht tut, dann ist es auch kein XP; Ende der Diskussion.“ (Beck, [Bec03] S. xvii)

Wenn auch einige der Methoden wie iterative und inkrementelle Entwicklung sowie automatisierte Tests breite Zustimmung und Anwendung fanden und finden, sind einige Aussagen darunter, die – zumindest auf den ersten Blick – in einem grundlegenden Widerspruch zu klassischen Verfahren stehen und deshalb gründlich zu untersuchen sind:

Die Fähigkeit, die Implementierung von Leistungsmerkmalen flexibel zu planen und dabei die sich ändernden geschäftlichen Anforderungen zu berücksichtigen. Noch etwas prägnanter formuliert JEFFRIES diese fantastischen Möglichkeiten (aus Kundensicht) in [Jef00] S. 33:

„We give you the ability to move very rapidly, and to change your requirements any time you need to.“

Wie es mit diesen Voraussetzungen möglich sein soll, unter realen Bedingungen ein Projekt zu einem vernünftigen und robusten Design zu verhelfen, wird durch die obigen Aussagen nicht geklärt. Die traditionelle Abfolge im Umgang mit Anforderungen – Analyse, Design, Implementierung – scheint jedoch auf den Kopf gestellt. Auch ob die

Entwicklung der Änderungskosten so verläuft, wie XP das verspricht ([Bec03] S. 23-24), bleibt abzuwarten.

Das Vertrauen darauf, das mündliche Kommunikation, Tests und Quellcode die Struktur und den Zweck des Systems zum Ausdruck bringen. Allein wenn man bedenkt, dass auch in agilen Projekten gelegentlich Entwickler das Team verlassen und neue hinzukommen, so scheint es äußerst schwierig, ohne Dokumente und Diagramme auf einem abstrakten Niveau, das Design und die Architektur der Software nur anhand von Unit-Tests und Sourcecode verständlich zu machen.

3.3.2 Werte

XP definiert die fünf Werte von XP als: *Kommunikation, Einfachheit, Feedback, Mut* und *Respekt*. In [Bec03] begründet BECK die Notwendigkeit zur Vereinbarung von Werten so:

„Die kurzfristigen Ziele von einzelnen Leuten stehen oft im Widerspruch zu den langfristigen Zielen der Gemeinschaft. Gesellschaften haben gelernt, mit diesem Problem umzugehen, indem sie gemeinsame Wertvorstellungen entwickelt haben, die durch Mythen, Rituale, Strafen und Belohnungen gestützt werden. Fehlen diese Werte, tendieren Menschen dazu, sich nur um ihre eigenen, kurzfristigen Interessen zu kümmern.“

Das *Kommunikation* eine zentrale Rolle in XP einnimmt wurde bereits im Abschnitt 3.3.1 deutlich, schließlich wird darauf vertraut „das mündliche Kommunikation [...] die Struktur und den Zweck des Systems zum Ausdruck bringen“ [Bec03]. Der Wert *Einfachheit* ist bekannt geworden durch den Slogan¹⁵ *Do The Simplest Thing That Could Possibly Work (DTSTTCPW)*, was bedeutet, nur das für die Realisierung einer Storycard¹⁶ gerade Notwendige zu tun. Das bezieht sich sowohl auf Algorithmen als auch auf mögliche zukünftige Anforderungen, die damit im Zusammenhang stehen könnten. Durch *Feedback* in einem XP-Projekt weiß das Team in welchem Zustand

¹⁵Phrasen und Abkürzungen, oft aus Wiki-Hyperlinks in Groß-Klein-Schreibung (CamelCase) hervorgegangen, sind zu einem wesentlichen Bestandteil der „XP-Kultur“ geworden ([SR03] S. 103)

¹⁶Beschreibung von Leistungsmerkmalen

sich das Projekt momentan befindet. Dazu werden insbesondere umfangreiche Tests durchgeführt. Mit *Mut* wird in XP die Fähigkeit bezeichnet, in komplizierten Situationen (wenn z.B. Fehler an der Architektur offenbar werden) neue Wege einzuschlagen und bereit zu sein, den bisher entwickelten Code wegzwerfen und von vorn zu beginnen. Als weiterer Wert gehört gegenseitiger *Respekt* dazu.¹⁷

Von diesen vier Werten leiten sich einige Grundprinzipien ab und konkretisieren dadurch die unbestimmten Werte. BECK nennt in [Bec03] die Grundprinzipien *unmittelbares Feedback*, *Einfachheit anstreben*, *inkrementelle Veränderung*, *Veränderungen wollen* und *Qualitätsarbeit* sowie einige weniger zentrale Prinzipien.

3.3.3 Aktivitäten

Aktivitäten in XP beschreiben, was zu tun ist. Werte helfen zu entscheiden, ob es richtig getan wird. BECK schließt in [Bec03] auf S. 50 folgendermaßen:

„Sie programmieren, weil Sie nichts leisten, wenn Sie nicht programmieren. Sie testen, weil Sie sonst nicht wissen, wann Sie mit dem Programmieren fertig sind. Sie hören zu, weil Sie sonst nicht wissen, was Sie programmieren und testen sollen. Und Sie entwerfen ein Design, damit Sie unendlich lange programmieren, testen und zuhören können. So sieht es aus. Dies sind die Arbeiten, die wir strukturieren müssen:

- Programmieren
- Testen
- Zuhören
- Designentwurf“

3.3.4 Praktiken

Werte und Prinzipien gemeinsam mit Praktiken machen XP zu einem anwendbaren Verfahren. Abbildung 3.3¹⁸ zeigt die in XP verwendeten Praktiken, die Kreise symbolisieren die Zusammenhänge zwischen diesen auf verschiedenen Ebenen. Praktiken sind dabei die Dinge, die

¹⁷<http://www.martinfowler.com/articles/newMethodology.html#XpextremeProgramming>, 21.12.2008

¹⁸<http://www.xprogramming.com/xpmag/whatisxp.htm>, 21.12.2008

ein Team täglich anwendet, wobei Werte die Basis dafür darstellen. Werte ohne Praktiken können nicht angewendet werden. Praktiken ohne Werte sind einfach nur Aktivitäten, die zweckfrei angewendet werden.¹⁹

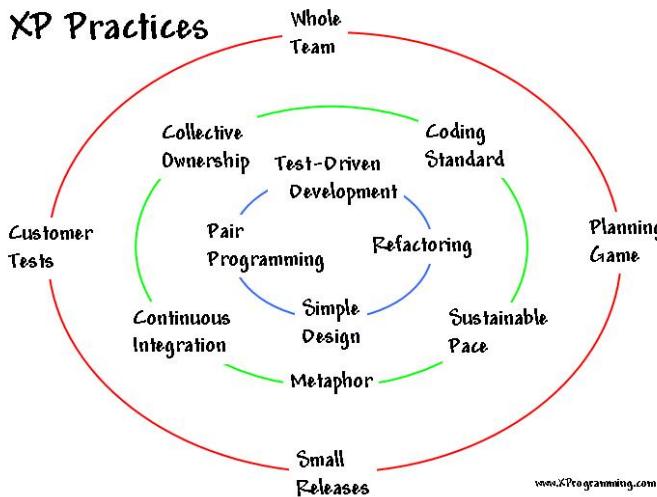


Abbildung 3.3: XP Praktiken

Testgestützte Entwicklung. XP verfolgt mehr oder weniger besessen das Konzept der testgetriebenen Entwicklung (engl.: test-driven development). Dabei werden zuerst *Komponententests* (engl.: unit tests) und danach die Komponenten selbst entwickelt. Erst wenn alle Tests erfolgreich durchlaufen werden, wird mit der Entwicklung des nächsten Leistungsmerkmals begonnen. Nach jeder Programmänderung werden sämtliche Tests erneut durchgeführt, so dass die Abalauffähigkeit der bereits implementierten Funktionen gewährleistet bleibt (*Regressionstest*). Auch die Kunden definieren Abnahmekriterien in Form von *Akzeptanztests*. Dadurch kann die Kundenzufriedenheit und der erzielte Nutzen validiert werden.

Das Planungsspiel. Mit dem Planungsspiel beginnt jede Iteration. Der Kunde oder das Kundenteam erläutert seine *User Story*, die mit dem Programmierteam diskutiert wird. Jede User Story wird

¹⁹Die in XP angewendeten Praktiken haben sich im Laufe der Jahre etwas gewandelt. Die Fundstellen mit größter Aktualität sind wahrscheinlich <http://www.xprogramming.com/xpmag/whatisxp.htm> und <http://c2.com/cgi/wiki?ExtremeProgrammingCorePractices> ([SR03]).

von den Programmierern grob abgeschätzt. In Abhängigkeit von den Aufwandsabschätzungen wird vom Kunden der Umfang der nächsten Iteration festgelegt. Die Auswahl ist am geschäftlichen Wert ausgerichtet. Die Programmierer zerlegen die User Stories in einzelne, technische Aufgaben. So werden Geschäftsprioritäten mit technischen Aufwandsschätzungen kombiniert.

Ganzes Team. Für das Projekt trägt das Team als Ganzes Verantwortung. Dabei existieren keine starren Rollenverteilungen. Natürlich sind Programmierer im Team, ebenso wie der Kunde oder das Kundenteam. Die früher verwendete Praktik *Kunde vor Ort* (engl.: *on-site-customer*) wird aktuell hier eingeordnet. Der Kunde oder ein Kundenteam gibt das Ziel jeder Iteration durch die Auswahl entsprechender User Stories vor. Der Kunde sollte ein tatsächlicher Benutzer des Systems sein und während der gesamten Arbeitszeit dem Entwicklerteam zur Verfügung stehen. An dieser Stelle sei darauf hingewiesen, dass in früheren Lesarten von XP der Kunde als eine Person aufgefasst wurde. Dieses Problem wurde mittlerweile erkannt und auf ein Team von Kunden ausgeweitet (KENT BECK in [McB02] S. xvi).

Programmierung in Paaren. Das Programmieren in Paaren ist eine Technik, die den Wissenstransfer steigern soll und ermöglicht, dass Anfänger von Spezialisten lernen können. Dabei codiert einer der Programmierer, während der andere eine eher abstrakte Sicht auf das System hat und dadurch den Überblick behält. Regelmäßig werden die Rollen getauscht und Programmiererpaare wechseln ebenfalls, so dass jeder Programmierer letztlich die gesamte Codebasis (in Teilen) bearbeitet. Durch dieses 4-Augen-Prinzip ist die Folge die Verminderung von programmtechnischen als auch fachlichen Fehlern.

Fortlaufende Integration. In XP-Projekten werden getestete Komponenten in kurzen Abständen zu einem lauffähigen Gesamtsystem zusammengefügt. Je häufiger dies geschieht umso routinierter und automatisierter wird die Integration ablaufen. Üblicherweise integrieren XP-Teams ihre Projekte mehrmals täglich. Die kurzen Integrationszyklen sind notwendig, damit die neu entwickelten Funktionen ebenfalls in kurzen Zyklen ausgeliefert werden können.

Designverbesserung (Refactoring). Stößt ein Programmiererpaar auf einen Designfehler an der Softwarekomponente, an der es gerade

arbeitet, wird dieser Defekt meist umgehend durch Refactoring beseitigt. Das gesamte Team verbessert so laufend das Design und die Architektur des Systems.

Kurze Releasezyklen. Nach relativ kurzer Zeit wird bereits ein einfaches System an den Kunden ausgeliefert. Dadurch erhält das Programmierteam wichtiges Feedback für die Weiterentwicklung. Innerhalb kurzer Zeit wird dann die nächste Version herausgebracht.

Einfaches Design. Jedes Leistungsmerkmal wird vom XP-Team auf die denkbar einfachste Weise realisiert. Es soll genau das Gewünschte erreicht werden und nicht mehr. Das Design der Software ist exakt an die momentane Funktionalität des Systems angepasst.

Metapher. Die Funktionsweise des gesamten Systems wird durch eine vom gesamten Team gemeinsam entwickelte Metapher veranschaulicht. Daran richten sich sämtliche Entwicklungsaktivitäten aus.

Gemeinsame Verantwortlichkeit. In XP wird das Team als Ganzes betrachtet. Eine Verantwortlichkeit eines Entwicklers für Teile des Systems ist nicht gegeben. Damit existiert auch kein Wissensmonopol Einzelner. Der Code gehört dem gesamten Team. Das schließt ein, dass jedes Programmiererpaar zu jeder Zeit beliebige Stellen des Codes bearbeiten kann (und soll).

Coding Standards. Die Entwickler des Teams halten sich bei der Programmierarbeit an verbindliche Festlegungen, die notwendig sind, damit jedes Programmiererpaar an jeder beliebigen Stelle der Codebasis arbeiten kann. Das bezieht sich insbesondere auf die Lesbarkeit des Codes, Vereinbarungen zu Konstanten- und Variablennamen etc.

Gleichmäßiges Entwicklungstempo. Unter zu hoher, permanenter Arbeitsbelastung leidet die Freude an der Arbeit, die Konzentration und Kreativität der Programmierer und damit die Qualität des Produkts. Deshalb sind Überstunden konsequent zu vermeiden. Diese Praktik wurde früher mit *40-Stunden-Woche* bezeichnet.

3.4 Scrum

3.4.1 Überblick

Die Ideen, die zur Schaffung von Scrum und auch zu dessen Namen führten, stammen von den japanischen Wissenschaftlern Hirotaka Takeuchi und Ikujiro Nonaka und wurden in dem 1986 erschienenen Artikel *The New New Product Development Game* entwickelt. Der Grundgedanke dieser neuartigen Produktentwicklung entstammt dem Rugby:

„In today’s fast-paced, fiercely competitive world of commercial new product development, speed and flexibility are essential. Companies are increasingly realizing that the old, sequential approach to developing new products simply won’t get the job done. Instead, companies in Japan and the United States are using a holistic method – as in rugby, the ball gets passed within the team as it moves as a unit up the field.“[TN86]

In [TOS08] wird von TAKEUCHI ET AL. festgestellt:

„The Toyota Production System (TPS) is a necessary but not sufficient condition for Toyota’s success. TPS is a radical innovation that has disrupted a huge market. However, without the ‚soft‘ factors that drive Toyota, they would still be just another auto company.

This is also true of Scrum where the Scrum framework is a necessary condition for hyperproductivity but that alone will never get you there. The team must rise to the occasion, engineering practices must be extraordinary, and management must provide an environment that removes all impediments that stand in the way. You can only get high quality by going fast in the right way and you can only go fast in the right way by working less, not more. Failure of managers to understand these paradoxes will cause Scrum to fail and often the company along with it. This is natural in a free market economy where bad companies deserve to fail and the sooner the better as then improved companies, products, and services can emerge more quickly.“

Daraus lassen sich einige grundlegende Aussagen zu Scrum ableiten:

- Das Team muss die Fähigkeit haben, Gelegenheiten wahrzunehmen und daran zu wachsen.
- Die angewandten Ingenieursmethoden müssen außerordentlich sein.
- Das Management muss alle Hindernisse aus dem Weg räumen.
- Effektivität gepaart mit Effizienz sind nicht mit Mehrarbeit zu erreichen.
- Wird das vom Management nicht verstanden, wird es nicht funktionieren.

Obwohl die von Takeuchi und Nonaka untersuchten Produktionsmethoden aus der Automobilbranche stammen, können diese Prinzipien durchaus auf die Softwareentwicklung übertragen werden, was sich vor allem im *Lean Development* Anwendung fand.

Der originale Scrum-Prozess wurde als Verfahren der Softwareentwicklung 1995 anlässlich der OOPSLA-Konferenz durch *Advanced Development Methods Inc. (ADM)* und *VMARK Software* – beides Mitglieder der Open Management Group – vorgestellt. Dem vorausgegangen waren Untersuchungen und Forschungen, an denen sich auch Wissenschaftler der DuPont Experimental Station beteiligten [ADMkA]. Als Begründer von Scrum sind Jeff Sutherland (VMARK) und Ken Schwaber (ADM) zu nennen. Später war ebenfalls Mike Beedle an der Weiterentwicklung beteiligt.

Scrum beruht auf einer Beobachtung, die durch die von ADM beauftragten DuPont-Wissenschaftler gemacht wurde: Sie unterschieden Prozesse in solche, die als *definierte Prozesse* bezeichnet und andere, die *empirische Prozesse* genannt wurden. Erstere sind dadurch charakterisiert, dass sie vorhersehbar, definiert und wiederholbar ablaufen. Letztere sind unvorhersehbar, nicht definiert und können nicht wiederholt werden. Der Softwareentwicklungsprozess nun gehört zur Kategorie der empirischen Prozesse, da nicht alle Daten, Fakten etc. bekannt sind und somit ständige Beobachtung und Steuerung notwendig macht. Die Wissenschaftler waren daher höchst erstaunt über die Tatsache, dass Softwareentwicklungsprozesse so durchgeführt werden, als ob es sich um definierte Prozesse handelte. Sie verglichen dies mit dem Ergebnis, das chemische Prozesse hervorbringen würden, wenn diese nicht vollständig verstanden würden: sie ergäben nicht vorhersagbare Resultate. ADM bestätigte letztlich diese Vermutung, da sie tatsächlich nicht vorhersagbare Ergebnisse erhielten, in Form von Software, die nicht ausgeliefert wurde oder den erwarteten Kundennutzen nicht erbrachten [ADM96].

Scrum geht also davon aus, dass Softwareentwicklung ein nicht vorhersagbarer Prozess ist. Scrum verfolgt einen iterativ, inkrementellen Ansatz und reagiert auf Änderungen im Projektumfang, Kostenrahmen und Zeitplan. Es werden Methoden zur Messung des Projektfortschritts und zum Management des Prozesses eingesetzt. Am Ende jeder Iteration kann ein bestimmter Umfang an Funktionalität ausgeliefert werden. Scrum kann wie folgt charakterisiert werden [ADM08]:

- Ein agiles Verfahren mit dem Softwareentwicklung überwacht und gesteuert werden kann.
- Ein „Mantel“ für vorhandene Praktiken der Softwareentwicklung.²⁰
- Ein teambasierter Ansatz zur iterativ, inkrementellen Entwicklung von Systemen, bei denen sich Anforderungen schnell ändern können.
- Ein Prozess, der hilft das Chaos, das aus Interessenkonflikten und unterschiedlichen Bedürfnissen entsteht, zu kontrollieren.
- Ein Weg zur Verbesserung der Kommunikation und zur Maximierung von Kooperation.
- Eine Möglichkeit die Dinge zu erkennen und zu beseitigen, die der Entwicklung von Systemen im Weg stehen.
- Ein Weg zur Maximierung der Produktivität.
- Anwendbar auf kleine Projekte bis zur Projekten, die ganze Organisationen umspannen.
- Eine Möglichkeit jedem im Team das Gefühl zu geben, gute Arbeit zu leisten.

3.4.2 Prozessrahmen

Das Kernstück in Scrum ist ein iterativ, inkrementeller Prozess. Dieser Prozess ist in Abbildung 3.4 dargestellt. Jede Iteration, die typischerweise einen Zeitraum von etwa 30 Tagen umfasst, hat ein Inkrement des Produktes zur Folge. In Scrum wird die Iteration als *Sprint* bezeichnet. Dieser Zuwachs an Funktionalität kann dem Kunden zur Verfügung gestellt werden. Im 24-Stunden-Rhythmus werden in Form des *Daily Scrum Meetings* die Anforderungen und Aktivitäten des Teams untersucht und synchronisiert. Angetrieben wird dieser Prozess durch eine Liste mit Anforderungen, dem so genannten *Product Backlog*, der die Anforderungen in Form von *User Stories* enthält.

²⁰Was Scrum geeignet für Kombinationen mit anderen Verfahren wie z.B. XP macht.

Dieser Zyklus wiederholt sich während der gesamten Entwicklungszeit des Projekts. Dabei untersucht das Team zu Beginn einer Iteration im *Sprint Planning Meeting* die durchzuführenden Arbeiten. Dabei werden die Elemente ausgewählt, die zum Ende einer Iteration in ein Inkrement potentiell auslieferbarer Funktionalität umgesetzt werden können. Diese Elemente werden im *Sprint Backlog* geführt. Während der Iteration arbeitet das Team selbstorganisierend an der Umsetzung der ausgewählten Elemente. Zum Ende der Iteration wird im *Sprint Review Meeting* das Ergebnis präsentiert, so dass das Resultat bewertet werden kann und Anpassungen in die nächste Iteration einfließen. [Sch07b]

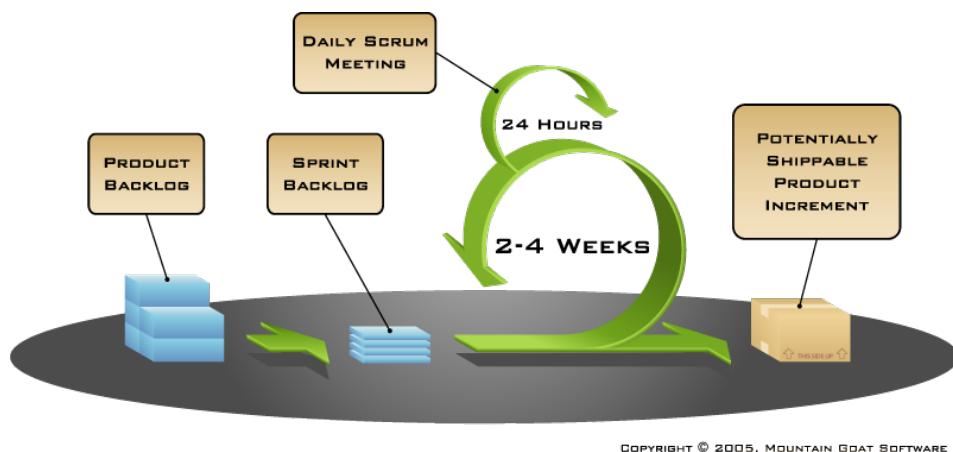


Abbildung 3.4: Prinzipielle Darstellung Scrum [Sof08]

3.4.3 Rollen

Scrum implementiert den oben beschriebenen Prozess durch drei Rollen: den *Product Owner*, das *Team* und den *ScrumMaster*. Für diese drei Rollen wird in Scrum der Begriff *Pigs* (deutsch: Schweine) verwendet. Alle anderen Personen, die irgendwie mit dem Projekt in Beziehung stehen aber nicht einer der drei genannten Rollen zugeordnet werden können, werden *Chickens* (deutsch: Hühner) genannt. Diese Bezeichnungen gehen auf einen Witz zurück, in dem ein Huhn und ein Schwein ein Restaurant unter dem Namen „Ham & Eggs“ eröffnen wollen, das Schwein aber schließlich davon absieht, da es direkt einbezogen, das Huhn jedoch nur beteiligt wäre [SB01]. So haben Chicken in Scrum nicht das Recht zur aktiven Beteiligung beispielsweise am Daily Scrum Meeting, dürfen diesem aber beiwohnen. Zu

den Chickens zählen Benutzer, Manager und alle sonstigen Stakeholder (deutsch: Interessenhalter). Deren Schnittstelle zum Scrum-Projekt ist der Product Owner.

Der Product Owner (engl.: Produkt Besitzer) ist die Person, die für die Verwaltung des Product Backlogs verantwortlich ist. Er ist gewissermaßen der Kunde, der „mit einer Stimme“ spricht²¹, er leitet also die Kundenwünsche an das Team weiter. Ebenso die Priorisierung der Elemente des Product Backlogs fallen in seinen Aufgabenbereich. Verantwortlich ist er außerdem für die Maximierung des Werts des Projekts, und somit auch für das Budget. Er ist dabei aktiv in die Produktentwicklung einbezogen, so dass für die effektive Arbeit des Product Owners darauf geachtet werden muss, dass dieser bevollmächtigt, verfügbar und qualifiziert ist [Pic08].

Der ScrumMaster ist für die Umsetzung des Scrum-Prozesses und für dessen Nutzen verantwortlich. Seine wesentliche Aufgabe besteht in der Beseitigung von Hindernissen, die das Team bei der Umsetzung der Anforderung behindern. Der ScrumMaster leitet außerdem das Daily Scrum Meeting. Die Wichtigkeit dieser Rolle macht eine Verschiebung des Fokus vom klassischen Projektleiter im Sinne der Wahrnehmung einer Führungsaufgabe, innerhalb derer Anweisungen erteilt und deren Ausführung kontrolliert werden, zur fördernden Unterstützung des Teams notwendig [Sch07b].

Das Team ist eine selbstorganisierende Gruppe von Entwicklern und Testern. Das Team wählt die Aufgaben für die nächste Iteration aus und schätzt den Arbeitsaufwand dafür ab. Es verantwortet die Umsetzung dieser Aufgaben in Softwarefunktionen. Hervorzuheben ist die Bedeutung der Selbstorganisation des Scrum Teams. Das Team ist für sein eigenes Management vollständig verantwortlich und hat die volle Autorität alles zu unternehmen, was zum Erreichen des Sprintziels notwendig ist.

²¹Einer der Gründe, der zum Scheitern des im Abschnitt 3.3.1 C3-Projekts führte, war nach Chet Hendrickson der Umstand, dass der Kunde nicht mit einer Stimme sprach (<http://www.c2.com/cgi/wiki?CthreeProjectTerminated>)

3.4.4 Artefakte

Für die Durchführung eines Scrum-Projekts sind einige verbindliche Artefakte vorgesehen.

Das Product Backlog listet die Anforderungen an das zu entwickelnde System auf. Für die Inhalte und Prioritäten ist der Product Owner verantwortlich. So wie sich das Produkt entsprechend der Rahmenbedingungen verändert wird auch das Product Backlog fortgeschrieben und erreicht nie einen Zustand der Vollständigkeit. Oder anders ausgedrückt: das Product Backlog gibt es solange das Produkt entwickelt wird. Die Elemente des Product Backlog sollten neben der Beschreibung eine Aufwandsschätzung enthalten und nach Prioritäten geordnet sein. Elemente sind sowohl Features, funktionale und nicht funktionale Anforderungen, Ergänzungen, Fehlerkorrekturen sowie technologische Fragen und Probleme. Wie in Abbildung 3.4 ersichtlich speist das Product Backlog den Sprint Backlog jeder Iteration, in der Scrum-Terminologie Sprint genannt.

Das Sprint Backlog verzeichnet die Elemente, die das Team im Laufe des Sprint Planning Meetings für den aktuellen Sprint aus dem Product Backlog ausgewählt hat. Die ausgewählten Elemente sollen im Laufe des dann folgenden Sprints in Produktfunktionalität umgesetzt werden. Nach Möglichkeit werden die Elemente weiter aufgegliedert, so dass Aufgaben entstehen, die in vier bis 16 Stunden abgeschlossen werden können. Dadurch wird auch die Aufwandsschätzung präzisiert. Jeder Aufgabe kann sowohl der Anlass oder der Urheber sowie die Verantwortlichkeit zugeordnet werden. Dadurch wird die Kommunikation der Beteiligten erheblich vereinfacht. Das wesentliche Datum ist jedoch die verbleibende Arbeit bis zur Fertigstellung für jede Aufgabe. Durch Summierung dieses Wertes über die Aufgaben des Sprints kann jederzeit der aktuelle Erledigungsstand transparent gemacht werden. Dies geschieht in Form eines *Burndown-Diagramms*.

Das Burndown-Diagramm zeigt die verbleibende Menge an Arbeitsaufwand in Korrelation zur Zeit und ist ein hervorragendes Mittel den Fortschritt des Teams zu visualisieren. Das Diagramm muss stets den aktuellen Stand wiedergeben und möglichst gut für alle Teammitglieder sichtbar sein. Abbildung 3.5 zeigt ein Burndown-Diagramm am 22. Tag eines Sprints. Durch eine Trendlinie und deren

Schnittpunkt mit der horizontalen Zeitachse kann der voraussichtliche Fertigstellungstermin zum aktuellen Zeitpunkt ermittelt werden. So ist auf einen Blick sichtbar, ob das Sprint-Ziel erreicht werden kann. Dadurch ist es sehr einfach, durch Hinzufügen oder Entfernen von Funktionalität verschiedene Varianten mit unterschiedlichem Release-Zeitpunkt zu simulieren.

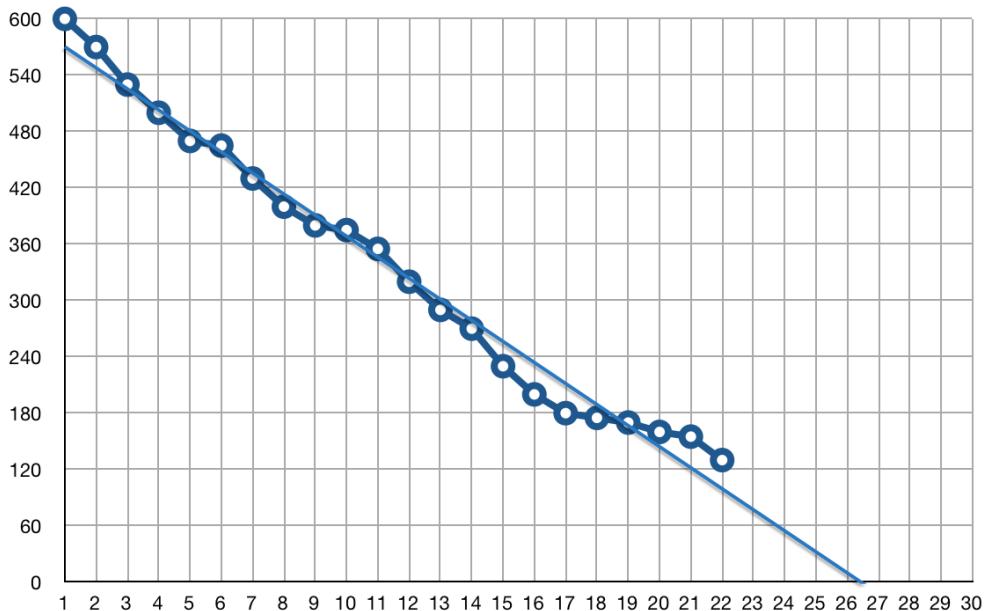


Abbildung 3.5: Burndown-Diagramm

3.4.5 Regeln

Das Sprint Planning Meeting wird am Anfang jedes Sprints abgehalten und besteht aus 2 Teilen, die jeweils 4 Stunden umfassen. Der Product Owner präsentiert im ersten Teil dem Team die Elemente des Product Backlog mit der höchsten Priorität. Gemeinsam wird erarbeitet, welche Elemente in Funktionalität umgesetzt werden. Die Verantwortung, welche Elemente schließlich ausgewählt werden, liegt beim Product Owner. Das Team übernimmt für die Menge der Elemente, die ausgewählt werden, die Verantwortung. Da für die Analyse des Product Backlog nur eine Timebox von vier Stunden zur Verfügung steht, ist die Gefahr, dass vor allem grobkörnige und unklare Elemente des Product Backlog nicht während des Sprints umgesetzt werden können, groß. Neben dem Product Owner und dem

Team nimmt der ScrumMaster am Sprint Planning Meeting teil. Gibt es keinen Product Owner übernimmt der ScrumMaster dessen Rolle. Im anschließenden zweiten Teil listet das Team die sich aus den Elementen des Product Backlog ergebenden Aufgaben im Sprint Backlog auf. Dabei agiert es völlig selbstständig. Der Product Owner steht zwar für Rückfragen zur Verfügung, unternimmt aber nichts, was einer Steuerung dieses Prozesses entsprechen würde. Als Ergebnis des zweiten Teils entsteht das Sprint Backlog, das die Gesamtverantwortung des Teams wiedergibt.

Das Daily Scrum Meeting wird jeden Tag zur möglichst gleichen Zeit am gleichen Ort durchgeführt. Im Daily Scrum Meeting wird der vergangene und der künftige Arbeitstag reflektiert, so dass der beste Zeitpunkt unmittelbar nach Arbeitsbeginn ist. Es ist essentiell, dass das gesamte Team an diesem Meeting teilnimmt oder ein Teammitglied in Vertretung den Status des abwesenden Mitglieds berichten kann. Das Meeting hat eine Timebox von 15 Minuten, setzt also den pünktlichen Beginn und Disziplin des Teams voraus. Es ist dabei durchaus empfohlen jedem zu spät kommenden Teammitglied eine wirksame Strafe z.B. in Form eines Geldbetrags (z.B. 1 EUR) aufzuerlegen [Sch07b]. Der ScrumMaster leitet das Meeting und stellt dabei jedem Teammitglied die gleichen drei Fragen:

- Was haben Sie seit dem letzten Daily Scrum Meeting fertiggestellt?
- Welche Aufgaben werden Sie bis zum nächsten Daily Scrum Meeting bearbeiten?
- Welche Hindernisse und Probleme sind aufgetreten?

Dabei soll nicht über Details diskutiert oder versucht werden, technische Fragen zu lösen. Dies kann direkt nach dem Daily Scrum Meeting geschehen. Jedes Teammitglied berichtet über seinen Status. Dadurch wird das gesamte Team synchronisiert.

Der Sprint ist der Zeitraum, in dem das Team versucht, die im Sprint Planning Meeting festgelegte Funktionalität umzusetzen. Üblicherweise hat ein Sprint eine Timebox von etwa 30 Tagen. Innerhalb dieses Zeitraums bleibt die Menge der Aufgaben überschaubar. Das ist insofern von besonderem Interesse, da das Team während dieser Zeit ungestört von äußeren Einflüssen arbeitet. Auch für das Management und andere Stakeholder ist dieser Zeitraum vertretbar, da

direkt nach Abschluss des Sprints - im Sprint Review Meeting (siehe Seite 46) - sichtbar wird, ob sich das Projekt in der gewünschten Richtung entwickelt. Während des Sprints kann das Team von außen Unterstützung und Hilfe anfordern. Ansonsten verwaltet sich das Team in dieser Zeit selbst. Die im Sprint Planning Meeting abgegebene Verpflichtung einer bestimmte Funktionalität des Product Backlog zu realisieren erfordert, dass das Product Backlog während des Sprints nicht verändert wird. Das Steuerungsinstrument während des Sprints ist das Daily Sprint Meeting, an dem alle Teammitglieder teilnehmen müsse. Der ScrumMaster hat die Möglichkeit einen Sprint abzubrechen, wenn erkannt wird, dass das Sprintziel nicht erreicht werden kann. Dazu kann der ScrumMaster vom Team oder vom Product Owner aufgefordert werden. In diesem Fall wird ein neues Sprint Planning Meeting angesetzt. Das Team hat die Möglichkeit in Abstimmung mit dem Product Owner Funktionalität aus dem Sprint Backlog zu entfernen oder hinzuzufügen. Während des Sprints muss das Team das Sprint Backlog täglich aktualisieren, um den Projektstand für alle transparent zu halten.

Im Sprint Review Meeting präsentiert das Team dem Product Owner und den Stakeholdern die im Sprint fertiggestellte Funktionalität. Dazu werden die Elemente des Product Backlog, zu denen sich das Team im Sprint Planning Meeting verpflichtet haben, durchlaufen. Im Scrum-Kontext ist komplette entwickelte und getestete Funktionalität, die potentiell ausgeliefert werden kann, fertiggestellt (engl.: done). Funktionalität, die nicht fertiggestellt ist, darf nicht vorgeführt werden. Ebenso ist eine Vorführung entlang eines „sicheren Pfades“, also das bewusste Vermeiden von instabilen Programmteilen, nicht zulässig. Dem Product Owner und den Stakeholdern darf kein Zustand vorgegaukelt werden, in dem sich die Software nicht tatsächlich befindet. Dazu gehört, dass die Funktionalität so nah wie möglich an den Bedingungen des geplanten produktiven Einsatzes präsentiert wird. Im Laufe der Vorführung können Fragen beantwortet sowie Änderungswünsche notiert werden. Die Stakeholder werden außerdem gebeten, Feedback zu geben. Dadurch wird das Product Backlog verändert, das dann die neue Grundlage im Sprint Planning Meeting darstellt. Der ScrumMaster gibt abschließend den Termin für das nächste Sprint Review Meeting bekannt.

Das Sprint Retrospective Meeting hat die Verbesserung des Prozesses zum Ziel. Innerhalb einer Timebox von drei Stunden werden den Teammitgliedern folgende Fragen gestellt:

- Was war gut im letzten Sprint?
- Was kann verbessert werden?

Am Sprint Retrospective Meeting nehmen das Team, der ScrumMaster und optional der Product Owner teil. Es handelt sich also um einen wertfreien Rückblick über den vergangenen Sprint. Die identifizierten Erkenntnisse und Ergebnisse werden entweder als nicht funktionale Anforderungen im Product Backlog oder in einer so genannten Impediment List (engl: Liste der Behinderungen) verzeichnet. Werden Retrospektiven oberflächlich durchgeführt und die gefundenen Probleme nicht bearbeitet reduziert sich automatisch die Bedeutung der Retrospektiven und das Team entwickelt das Gefühl mit den Schwierigkeiten allein gelassen zu werden.

3.5 APM

3.5.1 Überblick

Das APM-Verfahren wurde von der Firma OOSE Innovative Informatik GmbH aus Hamburg entwickelt und 2008 in [OW08] vorgestellt. OOSE ist als ein deutscher Vertreter in der Object Management Group (OMG)²² maßgeblich an der Weiterentwicklung professioneller objektorientierter Softwareentwicklung in Deutschland beteiligt.

APM ist im Kern ein konsequent timebox-getriebenes iteratives Vorgehen und stellt quasi ein Meta-Vorgehensmodell dar, welches viele Methoden und bewährte Techniken anderer agiler und auch klassischer Verfahren in einen Zusammenhang bringt. Dabei skaliert es nach Ansicht der Autoren sowohl von Kleinprojekten bis hin zu Megaprojekten mit mehr als 60 – 250 Teammitgliedern ([OW08] S. 34). Für Kleinprojekte ähnelt es am ehesten Scrum. Im Kontext dieser Diplomarbeit soll daher in diesem Abschnitt der Fokus auf die Unterschiede zu Scrum gerichtet werden. APM stützt sich allerdings insbesondere bei größeren Projekten auf eine featurebasiertes Vorgehen, auf das bei Kleinprojekten unter Umständen ganz verzichtet werden kann und deshalb hier nicht betrachtet wird.

²²http://sysml-directory.omg.org/vendor/oose_Innovative_Informatik_GmbH.html,
14.01.2009

3.5.2 Kleinprojekt

In [Oos08a] wird ein Kleinprojekt folgendermaßen definiert:

„[...] Projekte mit 1 – 8 Projektmitarbeitenden. Innerhalb eines Kleinprojektes werden zwei verschiedene Rollen unterschieden: ProjektmitarbeiterInnen und ProjektleiterIn.“

Im einfachsten Fall handelt es sich um ein Einmann- bzw. Einfrauprojekt. Sofern das Projekt aus mehreren Personen besteht, hat eine dieser Personen die Projektleitungsrolle inne, die das Projekt nach außen repräsentiert und nach innen Entscheidungsbefugnisse hat. [...]“

Die gute Eignung der genannten Teamgröße zur Projektdurchführung werden von den Autoren in der guten Führungsspanne und der auf die Kommunikation bezogene optimalen Größe (vgl. Fussnote auf Seite 4) gesehen.

3.5.3 APM-Iteration

Analog dem Sprint (Seite 45) in Scrum verwendet APM timebox-orientierte Iterationen. Innerhalb jeder Iteration werden die elementaren Aktivitäten Anforderungen definieren, Lösung konzipieren, Erfolgskriterien definieren, Lösung entwickeln, Erfolg messen und Planung aktualisieren durchlaufen. Auch hier steht am Ende einer Iteration ein messbares Ergebnis mit potentiell auslieferbarer Funktionalität. Wie in Abbildung 3.6 dargestellt nähert sich der Entwicklungsprozess von Iteration zu Iteration immer genauer dem Ziel und die Unschärfe nimmt ab. Die vollständig ausgeprägte APM-Iteration ist im Anhang in Abbildung A.2 auf Seite xxviii dargestellt.

3.5.4 Planungsebenen

Wie in Abbildung 3.7 dargestellt definiert APM so genannte Planungsebenen zur Strukturierung des Projektes.

Die Projektebene definiert auf der obersten Planungsebene die projektbezogenen Artefakte und Dokumente. Dazu gehören insbesondere der Projekt- und Releaseplan sowie Kostenaussagen. Die Aktualität wird während des gesamten Projektverlaufs sichergestellt. Auf dieser hohen Ebene wird somit das gesamte Projekt charakterisiert.

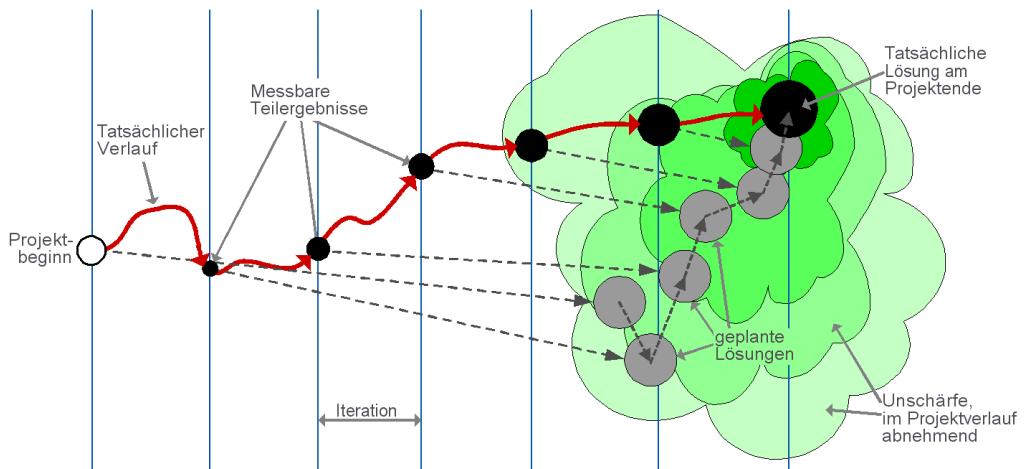


Abbildung 3.6: APM-Iterationswolkenmetapher [Oos08a]

Die Releaseebene bricht die Ergebnisse der Projektebene in feiner granulierte Bestandteile herunter und definiert diese in einem Iterationsplan. Dabei werden ebenso die Prioritäten der Anforderungen festgelegt.

Die Team- und Iterationsebene leitet aus dem übergeordneten Iterationsplan die möglichst konkreten Aufgaben für die Bearbeiter ab. Somit wird sowohl eine Grobplanung als auch eine Feinplanung vorgenommen. Durch zyklisches Review der Iterationen koppeln die einzelnen Ebenen in Form von Feedbackschleifen zurück. Die Iterationen werden als Timeboxen durchgeführt. Der Verlauf einer APM-Iteration ist in Abbildung A.2 dargestellt.

3.5.5 Projektphasen

Obwohl der Begriff Phase meist eine Assoziation zum Wasserfallmodell weckt wird in APM dieser Begriff in anderer Bedeutung verwendet. Ausgehend von der Überlegung, dass Iterationen für das Management oder den Lenkungsausschuss zu technisch, zu fein granuliert und daher schwer zu vermitteln sind, werden auf abstrakterer Ebene Phasen in Form von Meilensteinen (engl.: milestone) definiert. Damit sind grundlegende Fortschritte des Projektes als Zwischenziele festgelegt. Die in [Oos08a] zu den einzelnen Phasen angegebenen Zeiträume treffen zwar eher auf Projekte zu, die größer sind als die auf Seite 48

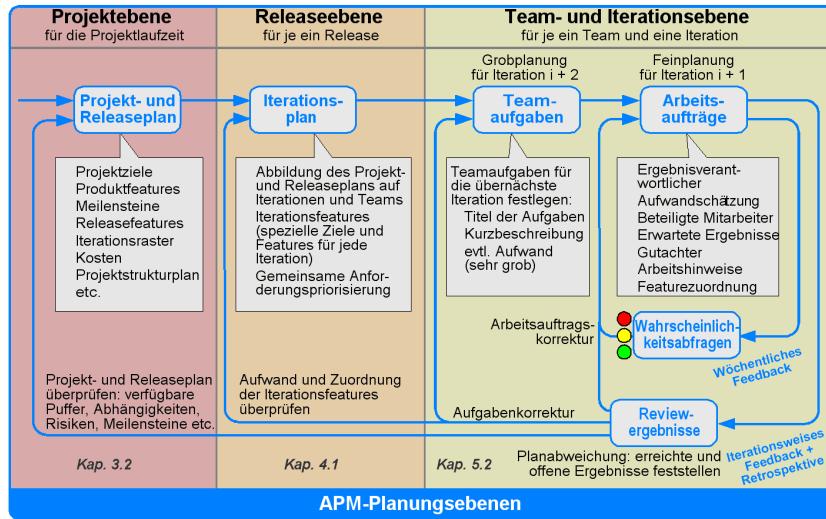


Abbildung 3.7: APM-Planungsebenen [Oos08a]

definierten Kleinprojekte, schließen aber die Anwendung der Phasen in diesen Projekten nicht aus.

Die Vorbereitungsphase dient zur Schaffung des grundlegenden Verständnisses der Projektaufgabe. In dieser Phase sollen auf einem entsprechend groben Niveau folgende wichtige Ergebnisse erreicht werden ([OW08] S. 105):

- Anforderungs- und Featureübersicht
- Grobes Klassen-/Datenmodell
- Identifizierung der wichtigsten Risiken und Chancen
- Realisierungsalternative und Rahmenbedingungen klären
- Aufwände und Termine abschätzen
- Groben Projektplan erstellen
- Abstimmung der Ergebnisse

Die typische Dauer dieser Phase liegt zwischen drei bis sechs Wochen. Am Ende der Phase sollte der Auftraggeber in der Lage sein, eine Entscheidung zu treffen.

Die Startphase soll den grundsätzlichen Lösungsansatz klarstellen, aus dem sich die fachliche und technische Architektur ableitet. Innerhalb einer Zeit von einem bis drei Monaten sollen folgende wichtige Ergebnisse entstehen ([Oos08a] S. 106):

- Identifizierung der Features, Anforderungen und von Anwendungsfällen
- Etablierung der anzuwendenden Arbeitsprozesse, Integrationsverfahren und Strukturen
- Realisierung der wichtigsten Features und Anwendungsfälle auf Grundlage der festgelegten Prioritäten
- Untersuchung von Risiken
- Fällen grundlegender fachlicher und technischer Entscheidungen
- Bereitstellung eines ersten Releases (ggf. als Prototyp)
- Festlegen der fachlichen und technischen Architektur

In der Hauptphase geht es darum, die angestrebte Lösung vollumfänglich zu realisieren und kann deshalb auch als *Kernphase* bezeichnet werden. In dieser Phase sind folgende Tätigkeiten enthalten bzw. entstehen folgende Ergebnisse ([Oos08a] S. 106):

- iterativ-inkrementelle Umsetzung der Features und Anforderungen gemäß den festgelegten Prioritäten
- Regelmäßige Integrationsbuilds
- Vorabnahmen der Software auf Basis der Integrationsbuilds
- Erstellung regelmäßiger Releases

Die Hauptphase hat eine typische Dauer von bis zu 1,5 Jahren.

Die Abschlussphase dient der Einführung der komplettierten Softwarelösung und schließt Abnahme, Freigabe und Auslieferung ein. Die Dauer dieser Phase liegt bei zwei bis acht Wochen. In dieser Phase werden folgende wichtige Ergebnisse erzielt ([Oos08a] S. 107):

- Test und Abnahme in der endgültigen Zielumgebung
- Beseitigung von Mängeln
- Erstellung von Korrekturreleases
- Inbetriebnahme
- Übergabe in die Wartungsorganisation

An die Abschlussphase schließt sich das eigentliche Projektziel – die Betriebsphase – an. Inwieweit darunter Wartung oder Weiterentwicklung des Softwaresystems verstanden werden kann hängt von den konkreten Umständen und Vereinbarungen ab.

3.6 Kritik an agilen Verfahren

Sicherlich ist generell kein Verfahren frei von Kritik. Doch vor allem *Extreme Programming (XP)* wurde in [SR03] einer substantiellen Kritik unterzogen. Das ist sowohl im „Hype“ rund um XP und das C3-Projekt (siehe Abschnitt 3.3) begründet als auch in objektiv sachlichen Argumenten. Im Folgenden ist zu betrachten, inwieweit die Kritik zutrifft und ob diese Aspekte auch die anderen agilen Verfahren berühren.

3.6.1 Extreme Programming Refactored

Wie bereits im Abschnitt 3.3.1 erwähnt sind zwei Punkte besonders interessant: *Oral Documentation* und *Embracing Change*. Auf diese beiden Aspekte und auf die Tatsache, dass die XP-Praktiken einander bedingen, wird in den folgenden Abschnitten exemplarisch eingegangen.

Oral Documentation

Üblicherweise werden Anforderungen in nicht-XP-Projekte in Form von „kugelsicheren“ Spezifikationen dokumentiert. Diese beschreibt wie sich das System aus Anwendersicht verhalten soll. Dazu ist eine gewisse Analyse des Problems erforderlich, welche das Problem hinter dem Problem zum Vorschein bringt [LW00]. Um eine Spezifikation zu erstellen, die detailliert genug ist, ein komplettes System exakt zu beschreiben, ist viel Arbeit notwendig. Der Vorteil ist das frühzeitige Erkennen von Problemen und Abhängigkeiten, deren Entdeckung zu einem späteren Zeitpunkt möglicherweise zu weit größeren Schwierigkeiten führen können. Um das Wasserfallproblem (vgl. Seite 15) zu vermeiden, ist es sinnvoll das zu entwickelnde System in Subsysteme aufzuteilen. Der Ansatz agiler Verfahren, Anforderungen erst dann zu bearbeiten, wenn diese realisiert werden sollen, ist durchaus von Wert. Schließlich können sich die Anforderungen in der Zwischenzeit ändern. Ebenso sollten Anforderungen mit hohem Risiko oder von großem geschäftlichem Wert einer intensiveren Analyse unterzogen werden. Dadurch entsteht Klarheit, sowohl für Entwickler als auch für Auftraggeber. Zahllose Projekte wurden unter Zuhilfenahme von Spezifikationen erfolgreich entwickelt. Dazu ist sicherlich Erfahrung und Wissen notwendig. Warum jedoch darauf verzichtet werden

soll kann in XP vielleicht dadurch erklärt werden, dass der Grund-satz *Embracing Change* gilt. Sind Änderungen an den Anforderun-gen zu jeder Zeit willkommen ist das Erstellen von Dokumenten dazu natürlich Zeitverschwendug. Eine Verständigung über Anforderun-gen mit dem Kunden (sign-off) und Überprüfung derselben nach der Entwicklung ist dann allerdings auch nicht möglich.

Der Verzicht auf Designdokumente, UML-Diagramme etc., verursacht spätestens bei der Weiterentwicklung von Systemen ernsthaft Schwie-rigkeiten. Das primäre XP-Ziel ist die Lieferung von Software. Die Weiterentwicklung zu ermöglichen ist erst das sekundäre Ziel. AMB-LER wies in [Amb02]²³ darauf hin, dass zum Erfüllen der Bedürfnisse der Stakeholder auch gehört, ein System zu realisieren, das robust genug für Weiterentwicklungen ist. Der Gedanke, dass XP gern zur Jobabsicherung angewendet wird, ist nicht aus der Luft gegriffen. In [SR03] (S. 168) wird Ron Jeffries folgendermaßen zitiert:

„Customer says ‚Please do these eleven things‘. Team says ‚We can do any ten this iteration, which ten do you want first?‘.

Customer says ‚I want eleven‘. Team says ‚We can do any ten this iteration, which ten do you want first?‘.

Customer says ‚You are idiots, I need eleven‘. Team says ‚Yes, we are, but we can do any ten this iteration, which ten do you want first?‘.

Customer says ‚If you won’t do [eleven], I’ll get another team.‘. Team says ‚If you can get a team that knows the system as well as we do, costs as little as we do, and that can do eleven, you should definitely get them. While you work on that, we can do any ten this iteration, which ten do you want first?‘“

Die in XP verwendeten User Stories erfüllen diese Zweck nicht. Sie sind nicht komplett, flüchtig und deren Abstraktionsebene kollidiert sehr schnell mit der simplen Tatsache, dass das präzise Erinnerungs-vermögen des Menschen eher schlecht einzuschätzen ist. Auch die Annahme, dass paarweises Programmieren und gemeinsamer Code den Mangel an Dokumentation kompensiert, ist nicht bewiesen und ist lediglich bei sehr kleinen Projekten überhaupt denkbar. Das Soft-wareprodukt ist auch nach dem Übergang in die Betriebsphase von den an der Entwicklung beteiligten Programmierern abhängig, was ein großes geschäftliches Risiko darstellt.

²³gefunden in [SR03]

Den weitgehenden Verzicht auf Dokumentation in XP-Projekten illustriert DOUG ROSENBERG in [SR03] S. 170 mit einer Frage, die ihm sein Sohn Rob stellte, nachdem er beiläufig von „Oral Documentation“ hörte: „Dad, is oral documentation oxymoronic²⁴ ... or is it just plain moronic²⁵?“

Embracing Change

In [Boe81] beschreibt BOEHM den exponentiellen Anstieg der Änderungskosten von der Analyse- bis zur Wartungsphase:

„Further, late corrections involve a much more formal change approval and control process, and a much more extensive activity to revalidate the correction. These factors combine to make the error typically 100 times more expensive to correct in the maintenance phase on large projects than in requirements phase.“

In Abbildung 3.8 ist sind die Änderungskosten grafisch dargestellt. Die unweigerliche Folge davon ist, dass jedes System irgendwann den entropischen Tod sterben muss – da die Kosten der Änderungen den geschäftlichen Nutzen übersteigen.

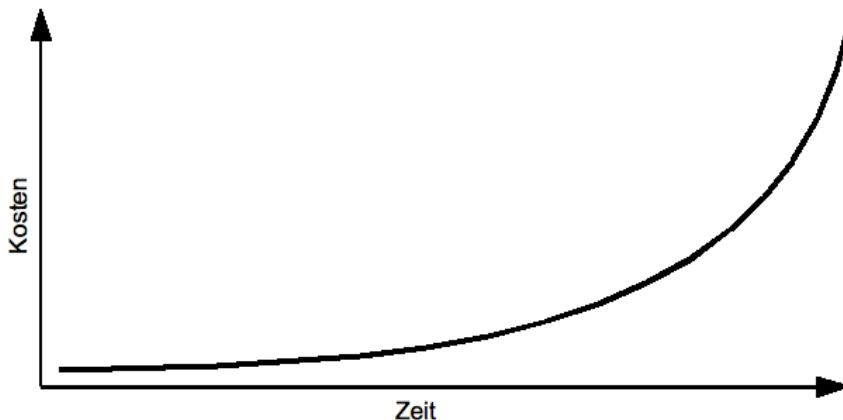


Abbildung 3.8: Änderungskosten nach Boehm

Die Behauptung von XP ist nun, dass diese Kurve abgeflacht verlaufen kann. Das beschreibt BECK in [Bec03] so:

²⁴oxymoron : Greek oxumōrón, from neuter of oxumōrós, pointedly foolish : oxus, sharp; see oxygen + mōrós, foolish, dull

²⁵moron : Origin: 1905–10, Americanism; Greek mōrón, neut. of mōrós foolish, dull

„Was wäre, wenn die Kosten von Änderungen mit der Zeit nicht exponentiell anstiegen, sondern sehr viel langsamer wüchsen und schließlich eine Asymptote erreichten?“

Die Kurve, die Beck den Informatikprofessoren von morgen hypothetisch empfiehlt, ist in Abbildung 3.9 dargestellt.



Abbildung 3.9: Änderungskosten nach Beck

Der starke Kostenanstieg zu Beginn eines Projektes ist damit zu erklären, dass XP – zumindest von einigen Vertretern – ein Projekt vom ersten Tag so betrachtet, als wäre es in der Wartungsphase. Die erfreuliche Aussicht, in Zukunft Änderungen ohne erhebliche Kosten realisieren zu können, muss jedoch als unbestätigt angesehen werden. Bisher veröffentlichte Daten zeigen, dass diese Abflachung für größere Projekte nicht stattfand ([Boe06], S. 19). Für Boehm jedenfalls ist dies die kostenintensivste Phase. Die Abbildung 3.10 zeigt zudem schematisch, dass die Kosten in genau die Projektphase verschoben werden, die Boehm als die geeignetste für Änderungen bezeichnet. Ob die Kosten dann im weiteren Projektverlauf nicht doch exponentiell ansteigen, kann durchaus mit einer „Was wäre“ -Spekulation a la Beck formuliert werden.

Frühzeitige, häufige Lieferungen gewährleisten schnell Feedback des Kunden. Dieser Umstand ist sicher eine Tatsache und positiv zu bewerten. Ob jedoch eine sehr schnell geliefertes erstes Release erheblichen Nutzen für den Kunden erbringt ist nicht sicher. Spätestens dann, wenn frühzeitige und häufige Releases in den produktiven Einsatz beim Kunden gelangen, ist die Wahrscheinlichkeit, das Projekt nie fertig zu stellen, sehr hoch. Schließlich werden meist wirkliche Daten in der wirklichen Welt verarbeitet. Das hat zur Folge, dass

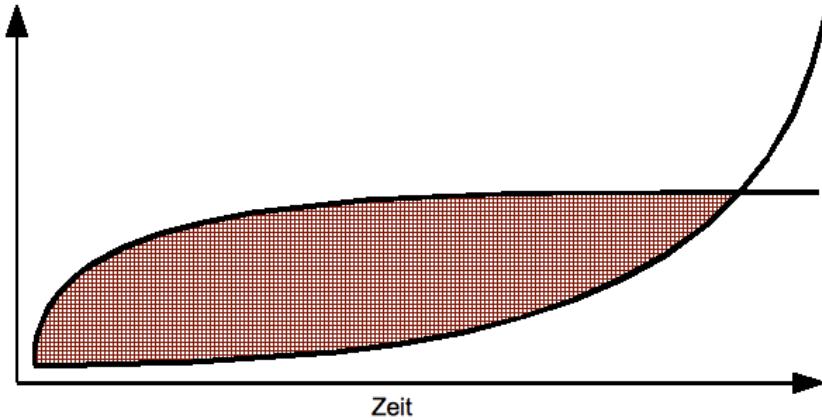


Abbildung 3.10: Änderungskostenverlagerung durch XP in die Projektstartphase

bei Änderungen am Design und der Architektur umfangreiche Arbeiten wie beispielsweise Datenkonvertierungen etc. durchgeführt werden müssen. Mit der nötigen Sorgfalt, da es sich um produktive Daten handelt. An diesem Punkt kann eine anfängliche Begeisterung des Kunden sehr schnell in das komplette Gegenteil umschlagen. Der Vorteil eines grundsätzlich besser verstandenen Problems durch ein hinreichendes Big Design Up Front (Vgl. Seite 29), das eine etwas längere Phase bis zum ersten Release mit sich bringt, kann erheblich größer sein. Außerdem kann in dieser frühen Phase Kundenfeedback auch mit Prototypen eingeholt werden. Die Chancen, die kurze Feedback-Zyklen und die Anwendung agiler Techniken sind in Abbildung 3.11 dargestellt. AMBLER belässt es bei der Kostenkurve nach Boehm (Vgl. Abbildung 3.8), zeigt aber auf, dass agile Techniken bereits in der frühen Projektphase wirksam sind.

Alles oder Nichts

Eines der Probleme, die sich beim geplanten Einsatz von XP einstellen, ist die Tatsache, dass die einzelnen Praktiken für sich genommen wahrscheinlich nicht besonders gut funktionieren werden. Darauf weist BECK in [Bec03] mehrfach hin. Die Bereitschaft, insbesondere in existierenden Projekten, XP mit allen Methoden einzuführen dürfte in der Regel nicht allzu hoch sein, ist damit doch ein hoher Aufwand und hohes Risiko verbunden. Scheitern jedoch XP-Projekte kann meist eine Praktik gefunden werden, die nicht vollständig ange-

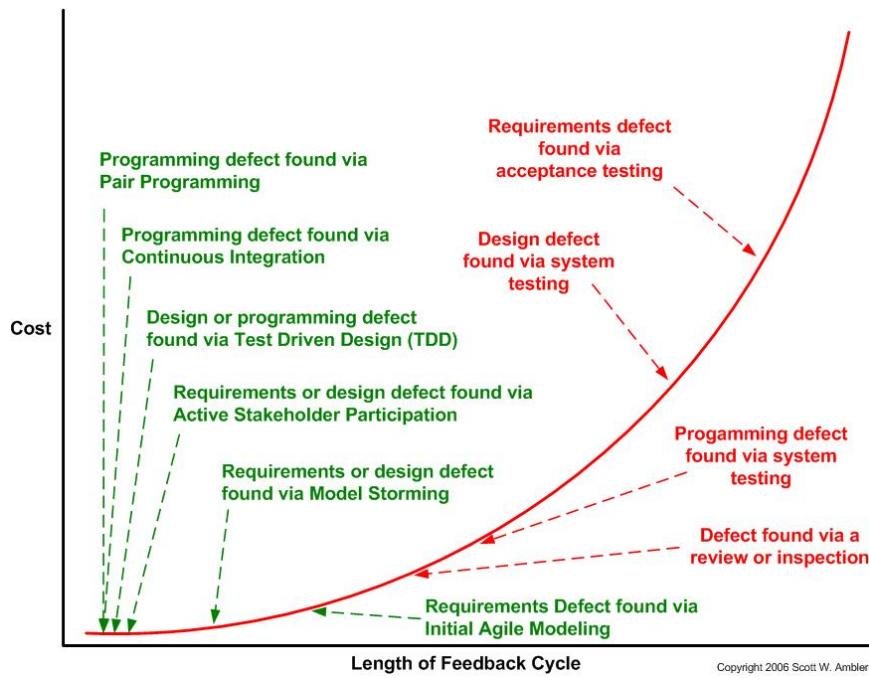


Abbildung 3.11: Anwendung agiler Techniken auf die Änderungskostenkurve [Amb08b]

wendet wurde, so dass sich XP gewissermaßen resistent gegen Kritik erweist.

Insofern ist XP ein symbiotischer Prozess, in dem sämtliche XP-Praktiken angewendet werden – oder es ist nicht XP. Das erfordert ausnehmend hohe Disziplin, vor allem bei der Einführung und macht diese schwierig. In Abbildung 3.3 ist die Abhängigkeit der einzelnen XP-Praktiken zu erkennen. Die Theorie in XP besagt, dass die Schwächen der einzelnen Praktiken durch die Anwendung der anderen Praktiken aufgehoben werden und sich so gegenseitig stützen. Fehlt nun aber eine dieser Praktiken ist der Kreislauf unterbrochen und die Anwendung anderer Praktiken schlägt fehl. Da wiederum einige Praktiken in der Praxis schwierig über den gesamten Projektverlauf durchzuhalten sind, ist die Gefahr, dass das Projekt dadurch in Schwierigkeiten gerät, inherent. Das sind besonders die Praktiken *Ganzes Team* (früher *Kunde vor Ort*) (vgl. Seite 36) und *Programmierung in Paaren* (vgl. Seite 36), was verteilte Teams und Fernarbeitsplätze nicht möglich machen würde, sowie *Designverbesserung* durch andauerndes Refactoring (vgl. Seite 36). Die Schwierigkeit, durch Refactoring das Design zu verbessern, erfordert außerdem entsprechend gut aus-

geprägte Fähigkeiten gutes von schlechtem Design zu unterscheiden. Deutlich wird dies durch folgendes Zitat:

„James Grenning mentioned that he has seen that when refactoring is neglected, it does not take long for the team to feel the pain. Once they feel the pain for themselves, they take it more seriously. Also a general lack of design skills makes refactoring difficult. If the team does not recognize good and bad design, refactoring does not get you anywhere.“²⁶

Fazit

XP wird nur dann funktionieren, wenn alle Praktiken eingesetzt werden. Das ist in der Praxis jedoch kaum möglich. Außer das Team besteht aus einer Reihe von Spitzentwicklern, von Programmierausten im positiven Sinne, die Praktiken wie Refactoring im Schlaf beherrschen. In der realen Welt ist das aber selten der Fall. Durch die Absolutheit der Erfordernisse, die XP postuliert, kann XP nicht angepasst werden und passt nur für eine relativ kleine Gruppe von Organisationsstrukturen. Dadurch kann vermutet werden, dass XP im Kern ein instabiler Prozess ist, auch wenn der Ansatz einer empirischen Prozesssteuerung bei weitgehend unbekannten Anforderungen richtig ist. Stabilität, also die Fähigkeit eines Softwareentwicklungsprozesses, nicht von einzelnen Komponenten existentiell abzuhängen, ist aber eine grundlegende Forderung an einen Softwareentwicklungsprozess.

Der extrem ausgeprägte Verzicht auf Dokumentation oder die Einschränkung auf „mündliche Überlieferung“ erweist sich in der Praxis als ebenso schwer durchführbar. In Kombination mit der Möglichkeit, jederzeit Änderungen der Anforderungen zu formulieren, ist das, was gemeinhin als Spezifikation einer Software bezeichnet wird, schlicht nicht vorhanden. Dadurch werden jedoch Instrumentarien verschenkt, die vor allem für Geldgeber, Stakeholder und das Management von großem Wert sind und helfen, das Projekt transparent zu machen.

Erwähnt sei an dieser Stelle, dass einige der XP-Praktiken von hohem Wert sind und einen festen Stellenwert in der Softwareentwicklung haben. Dazu gehören vor allem testgetriebene Entwicklung, kontinuierliche Integration und Coding Standards. Das wird auch durch die

²⁶<http://fc-md.umd.edu/projects/Agile/SecondeWorkshop/summary2ndeWorksh.htm>, 19.01.2009

Erweiterung von Scrum mit XP-Praktiken in xP@Scrum [Cha] und die Erweiterung des Rational Unified Process durch ein XP-Plug-In [IBM04] deutlich. Scrum stellt damit einen Projektmanagement-Wrapper bereit, der die XP-Praktiken ermöglicht und deren Einsatz nicht bedingungslos als Ganzes fordert. Dadurch erweist sich Scrum agiler als XP: *Individuals and interactions over processes and tools.* COHN schließt in seinem Posting in [Coh07b] wie folgt:

„I find true XP to be a small target off in the distance. If a team can aim at that and hit the bull’s eye, wonderful. If not, however, they are likely hacking (e.g., refactoring without any automated testing or TDD). Scrum is a big bull’s eye that on its own brings big improvements simply through the additional focus and the timeboxed iterations. That’s a good starting point for then adding the XP practices.“

Für eine Vertiefung zu diesem Thema sei auf [SR03] verwiesen.

3.6.2 Scrum

Prozessunterstützung

Anders als XP ist Scrum keine in sich geschlossene Sammlung von Praktiken und keine Methodologie. Scrum stellt vielmehr einen Projektmanagement-Wrapper bereit, der auf die konkreten Projektumstände zugeschnitten werden kann. Obwohl Scrum diverse Prinzipien und Regeln deutlich formuliert und fordert, ist es in der Durchführung bei weitem nicht so dogmatisch wie XP. Scrum gewährleistet dadurch eines systematisches Vorgehen während des Projektverlaufs. Der inkrementell-iterative Charakter von Scrum bezieht sich dabei automatisch auf das Projekt im technischen Sinne, als auch auf das Projektverfahren, das durch Retrospektiven kontinuierlich verbessert werden kann. Scrum unterstützt dadurch methodisch den *Kontinuierlichen Verbesserungsprozess (KVP)* des Teams. Im KVP versuchen Teams konkrete Verbesserungsvorschläge zu erarbeiten, die aus der Analyse und Bewertung des Arbeitsbereichs hervorgehen und daraus Massnahmen abgeleitet werden, die umgesetzt und geprüft werden [WW01]. In der Literatur sind ebenso Bezüge zur japanischen Industrie zu finden und ist mit dem japanischen *Kaizen* vergleichbar. Der Ursprung von Scrum ist ebenfalls in einer Untersuchung der japanischen Unternehmenskultur zu finden (vgl. Seite 38). So ist es möglich, die ISO 9001 mit Scrum zu implementieren [Sch08].

Kombination mit XP-Praktiken

Aus technischer Sicht ist es sinnvoll Scrum mit XP-Praktiken (die für sich genommen teilweise klassische Ingenieurs-Disziplinen sind) anzureichern. Dazu existieren mehrere Ansätze. Mike Beedle kombiniert Scrum in einzelnen Teams mindestens mit den Praktiken Continuous Integration, Testing (Unit, Spot, Regression, Acceptance), Release Management, Spontaneous Cooperation and Collaboration sowie den Standards: Coding, Documentation, Database zu *Enterprise Agile Process (EAP)* (früher *XBreed*) [Bee]. Wie Abbildung 3.12 zeigt, werden mit *XP@Scrum*, dem Ansatz von *Advanced Development Methods, Inc (ADM)*, die XP-Praktiken Simple Design, Testing, Refactoring, Pair Programming, Collective Ownership, Continuous Integration und Coding Standards angewendet.

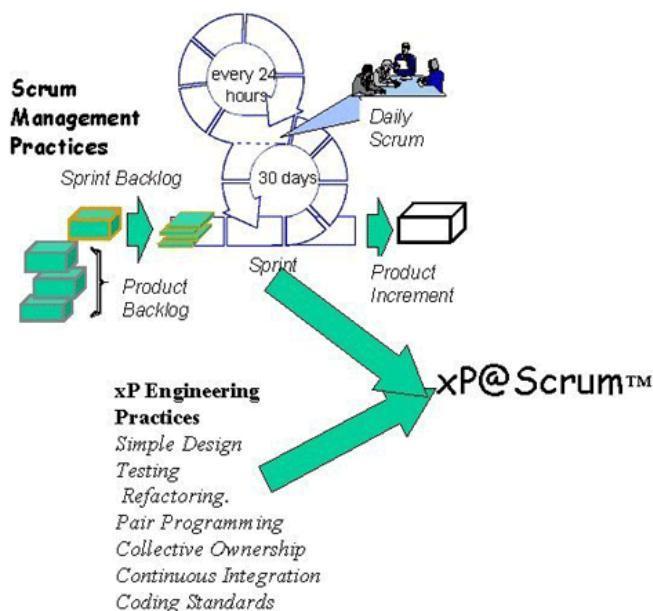


Abbildung 3.12: XP@Scrum [Cha]

Ausgehend von den im Abschnitt 3.3.4 erläuterten XP-Praktiken (Vgl. Seite 35) sowie den weiter oben genannten Ansätzen Scrum mit XP zu kombinieren, ergibt sich die Darstellung in Tabelle 3.1²⁷.

Es kann angenommen werden, dass der Einsatz der nachfolgenden XP-Praktiken zu den Erfolgsfaktoren bei Softwareentwicklung gene-

²⁷die Spalte *Scrum* steht für die Entsprechung der XP-Praktik originär in Scrum

Tabelle 3.1: Einsatz von XP-Praktiken mit Scrum

XP-Praktik	EAP	XP@Scrum	Scrum
Whole Team	–	–	Team
Planning Game	–	–	Sprint Planning Meeting
Small Releases	–	–	Product Backlog
Customer Tests	Acceptance Test	–	–
Collective Ownership	–	Ja	–
Coding Standards	Ja	Ja	–
Sustainable Pace	–	–	–
Metaphor	–	–	–
Continuous Integration	Ja	Ja	–
Test-Driven-Development	Ja	Ja	–
Refactoring	–	Ja	–
Simple Design	–	Ja	–
Pair Programming	–	Ja	–

rell und speziell beim Einsatz von Scrum in Softwareentwicklungsprojekten gehören:

- Continuous Integration
- Testing
- Coding Standards

Das bedeutet aber nicht, dass ausschließlich diese Praktiken Erfolg versprechend sind. Für sich genommen, ausgewogen und wohl überlegt eingesetzt, stellen auch die anderen XP-Praktiken einen praktischen Wert dar. So sollte beispielsweise Refactoring heutzutage zum Standardwerkzeug eines Softwareentwicklungsteams gehören.

Fazit

Eine ausführlich belegte Untersuchung oder grundlegend kritische Be trachtung des Verfahrens wie zu XP in [SR03] wurde zu Scrum nicht gefunden. Ebenso liegt bisher keine ausreichend breite Feldstudie vor, die es zulässt, die Einsatzfähigkeit von Scrum ausreichend zu beurteilen. Da jedoch Scrum – wie in Abschnitt 3.2.1 dargelegt – als die in Deutschland am erfolgreichsten eingesetzte Methode gilt, können dem Verfahren generell gute Erfolgsaussichten unterstellt werden.

Dafür spricht die Tatsache, dass Scrum kein dogmatisches Vorgehen fordert und als Projekt-Framework oder Projektmanagement-Wrapper ohnehin nicht den Projekterfolg garantiert. Scrum bietet aber eine fundierte Unterstützung für die agile Softwareentwicklung muss aber nicht darauf beschränkt werden. Es ist ebenso gut vorstellbar, Scrum bei Organisationsprojekten wie z.B. Beteiligung an Fachmessen einzusetzen.

Scrum, in Kombination mit den Praktiken *Continuous Integration*, *Testing* und *Coding Standards* eingesetzt, können als Erfolgsfaktoren bei der agilen Entwicklung von Softwareprojekten angesehen werden.

Bezüglich der Skalierbarkeit von Projekten (Projektgröße und/oder Teamgröße) bietet Scrum einen gut dokumentierten Ansatz, der als *Scrum of Scrums* bezeichnet werden kann [Coh07a]. Die Teams werden zu je 7 Teammitgliedern strukturiert und funktionieren jedes für sich wie in Abschnitt 3.4 beschrieben. Synchronisiert werden die Teams und deren Arbeit durch Meetings auf übergeordneter Ebene, eben dem Scrum of Scrums [Nar07]. Der Einsatz von Scrum in großen Projekte, z. B. Adwords bei Google [Sut06] (und der offensichtliche Erfolg dieser Projekte), lassen den Schluss zu, dass sich Scrum in großen Projekten bewährt.

3.6.3 APM

Scrum für Softwareentwickler in Deutschland

Im Kern handelt es sich bei APM um ein Verfahren, das große Ähnlichkeiten zu Scrum aufweist. Dabei nimmt APM besonders Rücksicht auf Aspekte der Projektdurchführung in Deutschland. Das zeigt sich an der Terminologie (Scrum-Sprint entspricht der Iteration, Product Backlog heißt Iterationsplan usw.) und an der klassischen Rollenverteilung (z.B. Vorhandensein des Projektleiters).

Das hier vorgestellte APM-Verfahren kann im Kontext dieser Diplomarbeit als Ergänzung und Kombination agiler Verfahren durch Methoden, Konzepte und Terminologie klassischer Verfahren interpretiert werden. Die Vorteile, die iterativ-inkrementelles Vorgehen agiler Prozesse bieten, werden mit traditionellen Mustern der Softwareentwicklung verbunden. Letztere sind vor allem im Denken des Managements, also der Entscheidungsträger, stark verwurzelt, und werden im APM-Verfahren entsprechend gewürdigt.

Dabei ist APM wesentlich stärker im Sinne eines Vorgehensmodells zu verstehen, da es sowohl die Methodik als auch die unterstützenden Prozesse reflektiert. Ein Schwerpunkt ist die detaillierte Projektplanung, bei der mit Phasen und Meilensteinen gearbeitet wird. Besonders hervorzuheben ist die Tatsache, dass APM sowohl für kleine Projekte als auch für Megaprojekte geeignet ist. Der Skalierbarkeit von Projekten wird damit fundierter Rechnung getragen als bei anderen agilen Verfahren. Im Vergleich dazu ist die Fähigkeit, Projekte zu skalieren, bei XP mit sehr großen Risiken behaftet und bei Scrum zwar gut dokumentiert, aber ob die Übertragung des Verfahrens auf höhere Strukturebenen reibungslos funktioniert ist nicht zwangsläufig gegeben.

Fazit

Die wesentlichen Merkmale, die APM auszeichnen und von den anderen agilen Verfahren unterscheiden, können kurz wie folgt charakterisiert werden:

- Gute agile Basis in Scrum
- Ausführlich dokumentiert
- Praxiserprob mit Fokus auf dem Management
- Auf den deutschen Markt bezogen
- Gute Fähigkeit zu Skalieren

3.7 Open Source

3.7.1 Open Source als kreative Kraft

Zwei wesentliche Aspekte bestimmten die Entwicklung des Software-Engineering in den 1990-er Jahren wesentlich mit. STALLMAN rief die *Free Software Foundation*²⁸ und die *GNU General Public License* ins Leben [Sta02]. Das führte zu einer stärkeren Wahrnehmung der Open Source Bewegung und schuf die Voraussetzungen zur freien Nutzung so überaus erfolgreicher Softwareprogramme wie den GCC C-Compiler und den emacs-Editor. Weitere wesentliche Meilensteine waren TORVALDS Linux, RAYMONDS Essay „Die Kathedrale und der Basar“ [Ray99] und Softwareprogramme wie Apache, TCL, Python,

²⁸<http://www.fsf.org>

Perl und Mozilla ([Boe06] S. 19). Denn nur freie Software, die der definitionsgemäßen Verwendung nach STALLMAN²⁹ genügt, kann so modifiziert werden, dass nicht das Softwaresystem den Prozess bestimmt, sondern das Softwaresystem sich an den Prozess anpassen lässt. Der Kerngedanke freier Software ist frei verfügbarer Quellcode. Der Quellcode freier Software steht jedem, der die Software verwendet, zur freien Verfügung. „Frei“ in diesem Zusammenhang bedeutet Freiheit (nicht notwendigerweise Nullkosten). Freier Quellcode ist öffentlich, frei zugänglich und niemandes Eigentum. Für Richard Stallman schließt diese Freiheit das Recht ein, die Software zu jedem beliebigen Zweck auszuführen, zu verstehen, wie die Software funktioniert und den eigenen Bedürfnissen anzupassen, Kopien weiter zu verbreiten, das Programm zu verbessern und die Öffentlichkeit von diesen Verbesserungen profitieren zu lassen.[Web04]

3.7.2 Motivation der Entwickler

Eine überaus interessante Frage ist die nach den Beweggründen, die Softwareentwickler veranlasst, aus freien Stücken Software zu entwickeln, ohne eine entsprechende finanzielle Gegenleistung. Der Softwareentwickler, gedacht als *homo oeconomicus*, der eigeninteressiert und rational handelt und auf die Maximierung seines eigenen Nutzens bedacht ist [Fra04], scheint nicht geeignet, sich an der gemeinnützigen Entwicklung von Software zu beteiligen. Andere Modelle in den Wirtschafts- und Sozialwissenschaften, wie der *homo sociologicus* oder der *homo culturalis* bieten dafür jedoch hinreichende Ansätze.

In [Gra04] werden die Beweggründe freier Entwickler – basierend auf Forschungsergebnissen des sozialwissenschaftlichen Projekts „Kulturaum Internet“ so zusammengefasst:

- „[...]“
- intellektuelle Herausforderung und Spiel,
 - Kreativität und Stolz auf etwas selbst Geschaffenes,
 - etwas verwirklichen, das den eigenen Ansprüchen an Stil und Qualität genügt,
 - ein sozialer Kontakt mit Leuten, die dieselben Ideen und Interessen teilen,
 - Ruhm,
 - Förderung der kollektiven Identität.“

²⁹<http://www.gnu.org/philosophy/free-sw.de.html>, 12.02.2009

Diese Charakteristik eines Softwareentwicklers passt augenscheinlich hervorragend zu den Formulierungen des agilen Manifests (Vgl. Seite 26). Besonders die Aussagen 1 und 3 spiegeln sich in den Beweggründen der Open Source-Entwickler wieder:

- Individuen und Interaktion sind wichtiger als Prozesse und Werkzeuge
- Kooperation mit Projektbetroffenen ist wichtiger als Vertragsverhandlungen

Das lässt den Schluss zu, dass Open Source und Agilität prinzipiell gut miteinander vereinbar sind. Ob agile Verfahren in vielen Open Source-Projekten angewendet werden kann an dieser Stelle nicht näher untersucht werden. Das eben Gesagte spricht dafür. Der Fakt, dass Open Source-Software regelmäßig von verteilten Teams entwickelt wird, spricht jedoch gegen agile Grundsätze, wie beispielsweise *Ganzes Team* (Vgl. Seite 36). Andererseits sind Projekte, in denen keine Kunden einer spezifischen Fachdomäne vorhandenen sind (z.B. Tools und Frameworks für IT-Professionals) oder Open Source-Fachanwendungen, bei denen das Kernteam aus den angestellten Entwicklern einer Software-Firma besteht, sehr gut für agile Verfahren geeignet.

3.7.3 Open Source im Unternehmen

Für jedes Unternehmen stellen sich bei der Auswahl einer Software diverse betriebswirtschaftlich relevante Fragen. Zu berücksichtigen sind nach [HOT⁺08] unter anderen die Faktoren

- Kosten,
- Qualität,
- Flexibilität,
- Aktivität,
- Sicherheit,
- Rechtssicherheit,

die im Folgenden kurz betrachtet werden.

Kosten für den Lizenerwerb entstehen beim Einsatz von Open Source-Software nicht. Das bringt einen erheblichen Kostenvorteil mit sich. Dennoch sind bei Open Source-Software Folgekosten, die beispielsweise durch Schulungen und Hardwareanpassungen entstehen,

zu berücksichtigen. Die Gesamtkosten werden meist durch Bestimmung der Total Cost of Ownership (TCO) bestimmt.

Qualität von Software ist ein wichtiges, aber schwer überprüfbares Kriterium. Die Qualität zeigt sich in der Stabilität einer Software während der Laufzeit. Bei Open Source-Software greift das Prinzip der Beteiligung vieler Entwickler und Tester, was das Auffinden von Fehlern frühzeitig ermöglicht und sich in häufigen Releases widerspiegelt. Die Fehlerbereinigung in proprietärer Software ist meist mit hohem Ressourceneinsatz verbunden, was häufige Releases eher verzögert. Außerdem wirkt sich das Bekanntwerden von Fehlern in proprietärer Software negativ auf die Vertriebserfolge aus, so dass häufig der Versuch unternommen wird, Fehler zu verschweigen. Das grundlegende Problem – Instabilität einer Software durch enthaltene Fehler – liegt jedoch sowohl bei Open Source- als auch proprietärer Software vor.

Flexibilität ist durch die Verfügbarkeit des Quellcodes bei Open Source-Software prinzipiell gegeben. Den Quellcode weiter zu entwickeln, setzt allerdings auch die nötigen Ressourcen und Kompetenzen voraus. Prinzipiell existieren dadurch mehr Freiheiten als bei proprietärer Software.

Aktivität eines Softwareprojekts ist die Konstanz, mit der ein Projekt weiterentwickelt wird. Bei proprietärer Software kann davon ausgegangen werden, dass die Software kontinuierlich weiterentwickelt wird. Jedoch besteht auch hier das Risiko, z.B. durch Firmenverkäufe, dass Projekte eingestellt werden, was kostspielige Migrationen nach sich zieht. Bei Open Source-Projekten ist das Risiko durch die freiwillige Teilnahme am Projekt latent gegeben. Zu beobachten ist jedoch, dass zunehmend kommerziell orientierte Unternehmen Open Source-Software bereitstellen und ihr Geschäftsmodell auf die damit verbundenen Dienstleistungen gründen.

Sicherheit einer Software ist bei Offenheit des Quellcodes prinzipiell überprüfbar. Die Wahrscheinlichkeit, dass eine Open Source-Software keine Sicherheitslücken aufweist, wächst mit der Verbreitung, die ein Projekt hat. Sicherheitslücken in proprietärer Software zu finden ist so nicht möglich. Dadurch können allerdings auch vorhandene Lücken z.B. von Schadsoftware schlechter genutzt werden.

Rechtssicherheit ist ein gern angeführtes Argument gegen den Einsatz von Open Source-Software. Da kein „Hersteller“ der Software vorhanden ist – so die häufig anzutreffende Auffassung – ist auch keine Gewährleistung und Haftung möglich. Nach deutschem Recht ist jedoch ein genereller Haftungsausschluss nicht möglich. Gleichwohl versuchen sowohl Open Source-Lizenzen als auch Hersteller von proprietärer Software die Haftung einzuschränken. Das deutsche Recht stuft Open Source-Software als Schenkung ein. Das hat zur Folge, dass der Schenker nur dann haftet, wenn ihm Fehler bekannt sind und er diese vorsätzlich verschwiegen hat. Für Dienstleistungen, wie beispielsweise Installation und Wartung, existieren keine Unterschiede. Open Source- und proprietäre Software werden hier nicht unterschieden.

Fazit

Die Entscheidung Open Source-Software im Unternehmen einzusetzen hängt von vielen Faktoren ab. Die Betrachtung einiger wesentlicher Aspekte soll den Umstand ins Bewusstsein rufen, dass Open Source-Software nicht bedenkenlos eingesetzt werden kann. Insofern handelt es sich um keine erschöpfende Betrachtung dieses Themas. Wie sich jedoch zeigte, sind die meisten Aspekte sowohl für Open Source- als auch für proprietäre Software von gleich großem Interesse. Daraus folgt, dass sich vor allem kleine Unternehmen die Kostenvorteile, die der Einsatz von Open Source-Software bietet, nutzbar machen sollten. Zu erwähnen ist außerdem, dass die noch ausstehende Entscheidung zu Softwarepatenten in Europa ein generelles Risiko für Open Source-Software darstellt.

Die Frage, ob Open Source-Software in unternehmenskritischen Bereichen mittlerweile als etabliert angesehen werden kann, ist klar mit ja zu beantworten. Die Fachzeitschrift OBJEKTSPEKTRUM notiert in [OBJ09] eine vom Marktforschungsinstitut Survey Interactive zu diesem Thema durchgeführte Studie, die dies deutlich zeigt: „Graude in Deutschland und Frankreich gelten Open Source-Lösungen als kostengünstiger als kommerzielle Software und zeichnen sich hier nach Meinung der Befragten durch eine große Flexibilität hinsichtlich Erweiterungsmöglichkeiten aus. Die Studie zeigt, dass Unternehmen Open Source nicht als Hype auffassen, sondern den Ansatz als zukunftsfähig betrachten.“

3.8 Einordnung

An dieser Stelle soll versucht werden, auf die in Abschnitt 1.1 aufgeworfenen Fragen mit den Ergebnissen der vorangegangenen Untersuchung schlüssige Antworten zu geben.

3.8.1 No Silver Bullet Yet

Unabhängig vom verwendeten Verfahren – ob klassisch oder agil – birgt hohe Komplexität ein inhärentes Risiko in sich. Streng klassische Verfahren beruhen auf dem Ansatz eines vorhersagbaren, definierten Entwicklungsprozesses. Das stellt sich jedoch bei steigender Komplexität als Irrtum heraus. Der Mangel an Flexibilität führt dadurch zu unvorhersagbaren Ergebnissen, so dass bei steigender Komplexität die Wahrscheinlichkeit des Erfolgs rapide abnimmt, wie Abbildung 3.13 zeigt [Sut98].

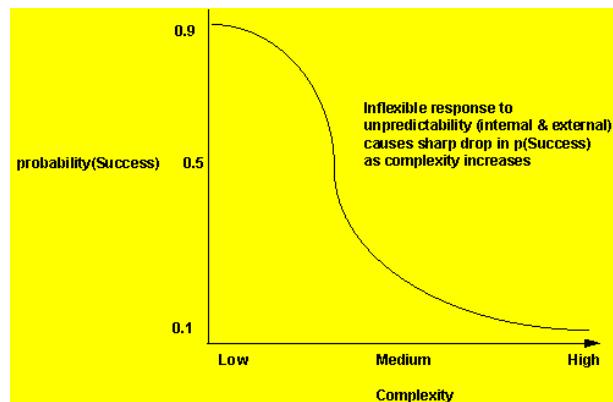


Abbildung 3.13: Fallender Erfolg bei steigender Komplexität mit klassischen Verfahren [Sut98]

Dem Problem wachsender Komplexität wird mit agilen Verfahren durch maximale Flexibilität begegnet. Als Metapher dient der evolutionäre Prozess, der bei biologischen Spezies als „punctuated equilibrium“ (deutsch: Punktualismus) [EG72] bezeichnet wird, wonach die evolutionäre Anpassungen der Arten nicht sanft und gleichförmig, sondern punktuell stattfinden. Wie Abbildung 3.14 zeigt, macht beispielsweise Scrum sich dieses Phänomen zu eigen [Sut98a]. Auf nicht oder nur schwer vorhersagbare Ereignisse kann ein agiles Verfahren punktuell reagieren und dadurch die Wahrscheinlichkeit des Erfolgs erhöhen.

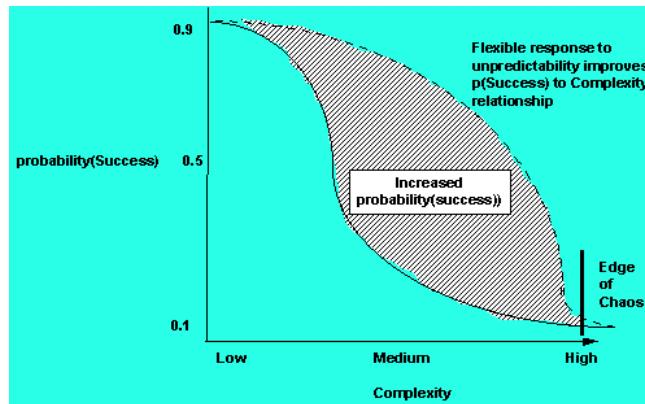


Abbildung 3.14: Wachsender Erfolg bei steigender Komplexität mit flexiblen (agilen) Verfahren [Sut98]

Abbildung 3.15 zeigt die Erfolgsquoten von agilen und traditionellen Softwareentwicklungsprojekten, wie sie eine im Jahre 2007 durch AMBLER durchgeführte internationale Studie erbracht hat [Amb07]. Zur im darauf folgenden Jahre durchgeführten Studie ([Amb08a]) bemerkt er weiter: „The DDJ 2008 Agile Adoption Survey showed that people's experience with agile software development was very positive, see Figure 3 [siehe Abbildung A.3], and that adopting agile strategies appears to be very low-risk in practice (few organizations seem to be running into serious problems, and most are clearly benefitting).“

Eine durch die Firma OOSE GMBH zusammen mit dem PMI³⁰ und der GPM³¹ durchgeführte aktuelle Studie zeigt in Abbildung 3.16 ein vergleichbares Ergebnis für Deutschland:

„Die drei am häufigsten genannten Verfahren erfolgreicher Projekte waren Scrum, APM und — etwas abgeschlagen — XP.“

Der RUP wurde etwa gleich häufig als Verfahren erfolgreicher wie erfolgloser Projekte genannt.

Das V-Modell XT (nicht aber V-Modell 97) wird etwa doppelt so oft von erfolgreichen wie von erfolglosen Projekten angewendet, kann also mit Scrum und APM mithalten.

Ein genauerer Blick in die Studie zeigt denn auch, dass es weniger die grundsätzlich Wahl des Verfahrens ist, als vielmehr die Anwendung bestimmter einzelner Techniken (die

³⁰Project Management Institute, <http://www.pmi.org>

³¹GPM Deutsche Gesellschaft für Projektmanagement e. V., <http://www.gpm-ipma.de>

bei Scrum und APM allerdings auch zentrale Eigenschaften sind).“ ([Oos09])

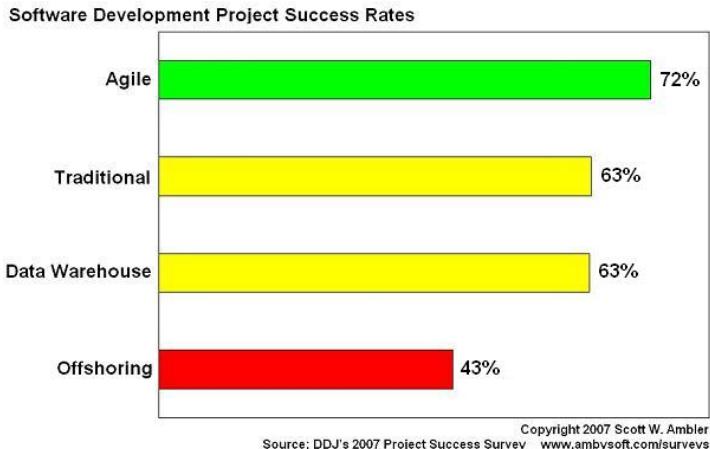


Abbildung 3.15: Erfolgsquoten von Softwareentwicklungsprojekten [Amb07]

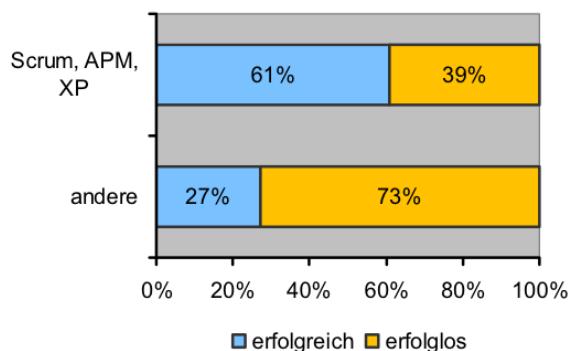


Abbildung 3.16: Erfolgsfaktor Vorgehensweise [Oos09]

Nach einer im Jahr 2008 von der Fachhochschule St. Gallen zum Thema Requirements-Engineering durchgeführten Studie [Hut09] ist die häufigste Fehlerquelle bei der Erhebung von Anforderungen sprachliche Differenzen. Kommunikation ist ein Schlüsselement agiler Verfahren und bietet damit gute Voraussetzungen, sprachlichen Differenzen entgegen zu wirken.

Antwort auf Frage 1 Zu der Frage, ob Unternehmen Softwareprodukte mit Hilfe agiler Verfahren Erfolg versprechender entwickeln können, lässt sich an dieser Stelle wie folgt beantworten: Der Einsatz agiler Verfahren garantiert nicht den Erfolg von Softwareentwick-

lungsprojekten. Allerdings kann festgestellt werden, dass agile Verfahren zum überwiegenden Teil erfolgreich durchgeführt werden und geeignet sind, steigender Komplexität und Unsicherheiten in Softwareentwicklungsprojekten besser begegnen zu können, als das bei klassischen Verfahren der Fall ist.

BOEHM identifiziert in [Boe06] zeitlose Prinzipien für jedes Jahrzehnt der Softwareentwicklung, die bei der Reflektion über den aktuellen Stand wichtig sind. Die Anwendung agiler Verfahren ist keine Erfindung der Neuzeit, sondern vielmehr eine Konsequenz aus den Erfahrungen von vielen Jahren Softwareentwicklung. Einige wichtige dieser teilweise seit den 1950-er Jahren geltende Prinzipien sollen hier genannt werden:

1. Für die Anforderungen der realen Welt sind rigoros angewendete sequentielle Prozesse zu vermeiden.
2. Nicht nur Informatik und Mathematik, sondern auch Verhaltenswissenschaften, Ökonomie und Veraltungswissenschaften sind Teil der Definition von „Engineering“.
3. Entwerfe Prototypen. Diese bergen wenig Risiko und bringen meist großen Nutzen.
4. Vermeide Last-Minute und Übernacht-Lösungen. Diese funktionieren meist nicht.
5. Eliminiere Fehler frühzeitig, am besten durch Analyse der Ursachen.
6. Vermeide Top-Down-Entwicklung. Neu hinzukommende Anforderungen und schnelle Änderungen machen diesen Ansatz unbrauchbar.
7. Zur Produktivitätssteigerung gibt es viele Möglichkeiten. Prototyping, Wiederverwendung und Prozessverbesserung gehören dazu.
8. Software muss dem Menschen nützen. Das ist der andere Teil der Definition von „Engineering“.
9. Sorge für Zufriedenheit der Stakeholder.
10. Sei kritisch gegenüber Hypes und Slogan's. Das YAGNI-Prinzip (engl.: You Aren't Gonna Need It) trifft nicht immer zu.
11. Sei skeptisch: Einheitslösungen und einen Königsweg („Silver Bullet“) gibt es meist nicht.

Agile Verfahren greifen diese Erfahrungen auf und stellen durch deren Grundsätze und Praktiken einen Softwareentwicklungsprozess bereit,

der den aktuellen Anforderungen gerecht wird. Die agilen Antworten auf oben genannte Prinzipien sind folgende (Vgl. Seite 28):

1. Anwendung eines iterativ-inkrementellen Ansatzes.
2. Menschen als handelnde Wesen, Kommunikation und Kooperation sind von zentraler Bedeutung.
3. Kurze Lieferzyklen gewährleisten frühes Feedback. Durch Prototypen können Kunden ihre Anforderungen besser formulieren.
4. Die Arbeit in agilen Projekten wird gleichmäßig verteilt (sustainable path).
5. Anwendung eines kontinuierlichen Refactoring.
6. Änderungen von Anforderungen werden im Projektverlauf erwartet. Durch Praktiken wie beispielsweise konsequentes Testen wird dies möglich.
7. Zur Verbesserung des Prozesses werden regelmäßig Retrospektiven durchgeführt.
8. Aktive Einbeziehung des Kunden in den Entwicklungsprozess.
9. Kontinuierliche frühzeitige Lieferung von Software, mit Nutzen für den Kunden.
10. Agile Verfahren passen nicht auf alle Projekte und Organisationsformen. Dogmatismus ist gefährlich (Vgl. Seite 52).
11. Ein agiles Verfahren muss an die Projektgegebenheiten angepasst werden können (Vgl. Seiten 73 und 68).

Die beiden letztgenannten Punkte zeigen jedoch, dass agile Verfahren nicht unkritisch übernommen und angewendet werden können. Ein Projekterfolg stellt sich nicht automatisch ein, indem ein bestimmtes Verfahren zur Anwendung kommt.

Antwort auf Frage 2 *Das Kernproblem bei der Entwicklung von Software bleibt von der Verwendung irgendeines konkreten Verfahrens weitgehend unberührt: die inhärente Komplexität von Software. Ein Königsweg („Silver Bullet“) stellt insofern auch die Anwendung agiler Verfahren nicht dar. Dies gilt auch trotz der Erkenntnis, dass agile Verfahren mit guten Erfolgssäussichten eingesetzt werden können. Hinzu kommt die Komplexität, die durch die Vielzahl von Verfahren, Methoden und Praktiken entsteht. Ebenso existieren keine Maßstäbe an die Professionalität von Softwareentwicklern. Die Frage, warum nach wie vor große Schwierigkeiten bei der Durchführung von Softwareprojekten existieren, kann durch die bisherige Untersuchung nicht*

beantwortet werden. Es kann nur vermutet werden, dass die jeweiligen Kenntnisse und Fähigkeiten oft den konkreten Projektsituatio-nen nicht gemäß sind und eine Reihe der oben genannten Prinzipien nicht beachtet werden. DENERT sagt dazu: „[...] Ich glaube, dass wir weitgehend wissen, wie man gute Software macht. Die Methoden, die Techniken, die Vorgehensweisen sind bekannt, aber wir sind in der Praxis der Unternehmen und Projekte nicht fähig, sie konsequent umzusetzen.“ [Den08]

3.8.2 Kleine Teams sind agil

Bedingt durch die große Bedeutung, die die Kommunikation der Mitglieder von agilen Teams hat, ist die Bestimmung der idealen Teamgröße entscheidend. Abbildung 3.17 zeigt den Produktivitätsindex bei verschiedenen Teamgrößen und dass dieser bei Teams mit mehr als acht Mitgliedern signifikant fällt [Sut98b].

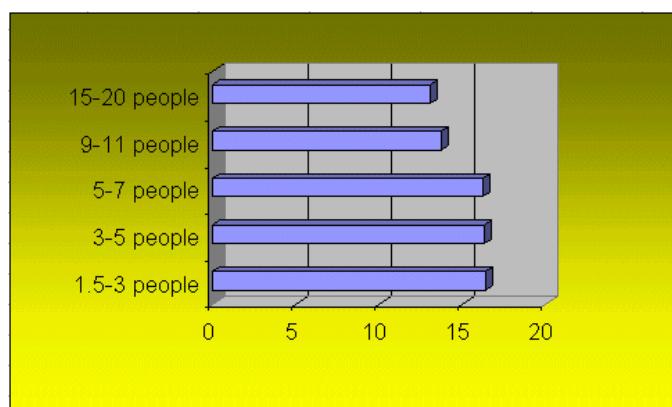


Abbildung 3.17: Produktivität in Abhängigkeit von der Teamgröße [Sut98b]

Antwort auf Frage 3 Die Frage nach der Größe von Teams kann durch die vorliegende Untersuchung klar beantwortet werden. Die maximale Teamgröße liegt demnach bei 7 Teammitgliedern. Das wird sowohl von den Befürwortern als auch den Kritikern agiler Verfahren so gesehen.

3.8.3 Das geeignete agile Verfahren

Kein Verfahren – auch kein agiles – kann für sich in Anspruch nehmen, dass es in jeder erdenklichen Situation zum Erfolg führt. Der Einsatz

eines speziellen Verfahrens hängt vielmehr von der Aufgabenstellung und den konkreten Rahmenbedingungen des Projektes ab. „Wer sich *dogmatisch* an einen Prozess wie Scrum oder XP hält, lebt definitiv keinen agilen Prozess.“ [Eck09] Wesentlich dabei ist die Akzeptanz, die ein Verfahren sowohl im Team als auch im Management hat. Wird ein Verfahren nicht unterstützt kann es nicht funktionieren. Das gilt für agile Verfahren um so mehr, als diese Individuen und Kommunikation über Prozesse und Werkzeuge gestellt werden (Vgl. Seite 26). Unter Berücksichtigung dieser Anmerkung scheint Scrum den Bedürfnissen am besten gerecht zu werden.

Antwort auf Frage 4 *Für die agile Softwareentwicklung innerhalb kleiner Teams speziell in Deutschland ist Scrum – vor allem in Kombination mit einigen XP-Praktiken – besonders gut geeignet. Darauf wies bereits die in Abschnitt 3.2.1 zitierte Studie hin und konnte durch die Untersuchung im Abschnitt 3.4 und 3.6.2 bestätigt werden.*

3.9 Zusammenfassung

Im vorangegangenen Abschnitt wurde der Stand des Wissens im Kontext dieser Arbeit wiedergegeben. Dabei wurde versucht, auf die in Deutschland relevanten agilen Verfahren so vertiefend einzugehen, dass der Leser ein realistisches Bild von den Chancen und Risiken agilen Vorgehens bekommt. Durch die bisherige Untersuchung konnten bereits einige der eingangs aufgeworfenen Fragen weitgehend beantwortet werden.

Die Beantwortung der in Abschnitt 1.1 aufgeworfenen Frage nach einem geeigneten Softwaresystem (Vgl. Seite 5) ist an dieser Stelle noch nicht möglich. Aus dem bisher Gesagten ergibt sich jedoch, dass eine Open Source-Projektmanagementsoftware folgende Aspekte möglichst gut berücksichtigen sollte:

- Unterstützung des iterativ-inkrementellen Entwicklungsprozesses (Vgl. Seite 28)
- Fähigkeit zur Anpassung an das verwendete Verfahren – insbesondere Scrum (Vgl. Abschnitt 3.4) – und die Projektgegebenheiten
- Integration der Praktiken *Continuous Integration, Testing* und *Coding Standards* (Vgl. Abschnitt 3.6.2, Tabelle 3.1)

- Einfachheit der Software, da die beteiligten Personen und nicht das Werkzeug oder der Prozess im Vordergrund stehen (Vgl. Seite 26)
- Aktivität des Open Source-Projekts (Vgl. Abschnitt 3.7.3)

Im folgenden Kapitel 4 werden die Anforderungen untersucht, die eine Open Source-Projektmanagementsoftware erfüllen sollte, damit agile Verfahren möglichst optimal unterstützt werden. Verschiedene Vertreter aus dem Bereich Open Source-Projektmanagementsoftware werden kurz charakterisiert und deren Erfüllungsgrad der definierten Anforderung untersucht.

Kapitel 4

Projektmanagementunterstützung agiler Verfahren

Der Projektmanager ist davon überzeugt, dass der durchschnittliche künftige Fortschritt den durchschnittlichen zurückliegenden Fortschritt übertrifft.

(Tom DeMarco)

4.1 Überblick

Projektmanagement ist gekennzeichnet durch zwei Aspekte. Einerseits die Führung und Steuerung von Projekten durch handelnde Personen, meist den Projektleiter. Andererseits durch den Einsatz von Werkzeugen und Techniken.

In den folgenden Abschnitten wird der Aspekt *Werkzeuge und Techniken* in Form von Open Source-Projektmanagementssoftware betrachtet. Für eine Vertiefung des Aspekts der *Projektführung und Projektsteuerung* sei an dieser Stelle auf [Lit07], [Sch07a] und [DeM98]. Die Komplexität des Themas soll lediglich durch einen Witz illustriert werden (siehe Abbildung 4.1):

„A tourist walked into a pet shop and was looking at the animals on display. While he was there, another customer walked in and said to the shopkeeper, ‚I’ll have a C monkey please.‘ The shopkeeper nodded, went over to a cage at the side of the shop and took out a monkey. He fitted a collar and leash, handed it to the customer, saying, ‚That’ll be £5,000.‘

The customer paid and walked out with his monkey.

Startled, the tourist went over to the shopkeeper and said, „That was a very expensive monkey. Most of them are only a few hundred pounds. Why did it cost so much?“ The shopkeeper answered, „Ah, that monkey can program in C - very fast, tight code, no bugs, well worth the money.“

The tourist looked at a monkey in another cage. „Hey, that one's even more expensive! £10,000! What does it do?“

„Oh, that one's a C++ monkey; it can manage object-oriented programming, Visual C++, even some Java. All the really useful stuff,“ said the shopkeeper.

The tourist looked around for a little longer and saw a third monkey in a cage of its own. The price tag around its neck read £50,000. The tourist gasped to the shopkeeper, „That one costs more than all the others put together! What on earth does it do?“

The shopkeeper replied, „Well, I haven't actually seen it do anything, but it says it's a project manager.“ (Nick Jenkins [Jen06])



Abbildung 4.1: Projektmanager?

4.2 Besteht Notwendigkeit zur Werkzeugunterstützung?

Ausgehend von Satz 1 des agilen Manifest (Vgl. Seite 26), wonach Individuen und Interaktion wichtiger sind als Prozesse und Werkzeuge,

steht die Suche nach einem geeigneten Werkzeug nicht im Vordergrund. In der Tat bauen agile Verfahren wesentlich stärker als die klassischen Vertreter auf Individuen, Kreativität und Teamgeist. Das schlägt sich besonders deutlich in einigen XP-Praktiken, wie Ganzes Team (Vgl. Seite 36) und Programmierung in Paaren (Vgl. Seite 36) oder dem Daily Scrum Meeting (Vgl. Seite 45) nieder.

Weshalb ein Werkzeug überhaupt notwendig ist wird in [DS08] folgendermaßen begründet:

„[...] it is essential to have at your disposal a tool that enables requirements gathering, iteration planning, progress tracking and reporting. You can't rely on memory for requirements gathering. You can't rely on the universal perception for iteration planning and you definitely can't rely on telepathy for progress tracking and reporting. You need a tool that will do the job with minimum effort and minimum side effects.“

Demnach ist das einfachste Werkzeug die Kombination von Flipchart, Whiteboard, Index Cards und Stift.

Den Vorteilen (flexibel, einfach zu handhaben, preiswert), die diese einfachen Werkzeuge bieten, stehen naturgemäß einige Nachteile gegenüber:

- nicht anwendbar in verteilten Teams
- keine Möglichkeit der Berichterstellung
- Änderungen sind aufwändig

An gleicher Stelle wird RON JEFFRIES zitiert:

„I think that people and how they interact on a project are the most important thing, and I think that they need to create a way of working – a process – that works best for them. Because their interactions are critical to project success, I suggest that teams begin the work with an approach that will bring them together as people, not one that will let them remain apart, communicating electronically.“

Zum Aspekt *verteilte Teams* bemerkt AMBLER: „There is a significant risk premium on team distribution, [...] If your team is distributed, get good tooling.“ [Amb08a].

Welche weiteren Aspekte sind es, die den Einsatz von Software sinnvoll oder gar notwendig machen? Wie im Abschnitt 3.6.1 ist der weitgehende Verzicht auf schriftliche Dokumentation problematisch. Aus

der Projekterfahrung des Autors ist eine hinreichende Dokumentation vor allem bei langfristigen Projekten oder Projekten, die sich in der Wartungsphase befinden, unumgänglich. Die Transparenz eines Projekts wird durch Software auf lange Sicht ebenso verbessert, was die Einarbeitung neuer Teammitglieder einfacher gestaltet. Die aus XP bekannte Praktik der gemeinsamen Verantwortlichkeit (Vgl. Seite 37) ist auf Grund der Spezialisierung (Programmierer, Tester, etc.) in der Praxis oft nicht gegeben. Mit Hilfe eines Software-Werkzeugs kann eine Wissensbasis entstehen, die implizites Wissen in explizites Wissen überführt.

Hinzu kommen die vielfältigen Möglichkeiten, die insbesondere Webbasierte Werkzeuge bieten:

- Festlegen von Prioritäten der Backlog-Elemente
- Release-Planung
- Kontrolle des Projektfortschritts
- Versionskontrolle
- Kontinuierliche Integration
- Testunterstützung
- Fehlermanagement (Bug Tracking)
- User Help Desk

Fazit

Der Vorteil der direkten Kommunikation, die ein *Team vor Ort* bietet, kann am besten mit einfachen Werkzeugen (White Board, Index Cards, Stift) genutzt werden. Einige Aspekte sprechen jedoch auch in dieser Situation für den Einsatz von Software-Werkzeugen. Webbasierte Software sollte dann verwendet werden, wenn:

- das Team verteilt arbeitet,
- Auswertungen und Berichte erforderlich sind,
- Wissen expliziert werden soll,
- die technischen Möglichkeiten einer Softwarelösung von Nutzen sind.

Die Vorteile flexibler manueller Techniken können ohne weiteres genutzt werden, indem die Arbeitsergebnisse fotografiert und als Dateien in das Software-Werkzeug integriert werden.

Analog zur Feststellung, dass es kein Verfahren gibt, das auf jede Projekt situation passt (Vgl. Seite 73), kann auch hier vermutet werden,

dass es kein Werkzeug gibt, das allen Situationen gleichermaßen gut gerecht wird. Insofern ist die Flexibilität, die ein Werkzeug bietet, ein entscheidendes Kriterium für dessen erfolgreichen Einsatz.

4.3 Anforderungen an Open Source-Projektmanagementsoftware

4.3.1 Erfolgsfaktoren

Im Abschnitt 3.9 wurden bereits wesentliche Aspekte, die eine Open Source-Projektmanagementsoftware berücksichtigen sollte, genannt. Diese decken sich weitgehend mit den Erfolgsfaktoren, die die Firma *VersionOne*¹ in einem Kriterienkatalog [Ver08], der Hilfe bei der Auswahl einer geeigneten Projektmanagementsoftware bietet, aufführt.

Dabei muss angemerkt werden, dass es sich dabei um ein Vermarktungsinstrument für deren eigenes Produkt handelt. Da die Projektmanagementsoftware *VersionOne* keine Open Source-Projektmanagementsoftware ist wird dieses Produkt nicht genauer betrachtet, obwohl die Software für einzelne Teams mit bis zu 10 Entwicklern kostenlos zum Download zur Verfügung steht². In [Ver08] wird auf diverse Erfolgsfaktoren und Best Practices eingegangen, die im Folgenden bei der Bestimmung der Anforderungen an Open Source-Projektmanagementsoftware helfen sollen.

Demnach sind bei der Auswahl einer geeigneten Projektmanagementsoftware sechs Erfolgsfaktoren zu berücksichtigen:

- Iterativ-inkrementelle, Feature-getriebene Entwicklung
- Management Software-Lebenszyklus
- Funktionsübergreifende Teams
- Flexible Konfiguration
- Einfachheit
- Skalierbarkeit

Die genannten Erfolgsfaktoren werden im Folgenden kurz charakterisiert.

¹<http://www.versionone.com>

²<http://www.versionone.com/VersionOneTeam.asp>, 28.02.2009

Iterativ-inkrementelle, Feature-getriebene Entwicklung

Agile Planung eines Softwareprodukts muss auf der Basis von Releases und Iterationen möglich sein. Die Planung und Prioritäten müssen leicht zu ändern sein. Als primärer Planungsansatz werden Features in Form von User Stories (Vgl. Seite 40) und Anforderungen verwendet.

Management Software-Lebenszyklus

In agilen Softwareprojekten hängen einzelne Phasen sehr eng zusammen. Die Planung des Projekts, der Features und die detaillierte Test- und Aufgabenplanung sowie das Management dieser Bestandteile müssen nachvollziehbar sein und jederzeit sichtbar gemacht werden können.

Funktionsübergreifende Teams

Die Bedürfnisse und Interessen verschiedenster Mitglieder des Teams müssen berücksichtigt werden. Sowohl der Kunde, das Produkt- und Projektmanagement, die Programmierer, Tester und weitere Stakeholder verbessern durch die Verwendung eines einheitlichen Systems die Zusammenarbeit und das gemeinsame Projektverständnis.

Flexible Konfiguration

So wie kein Projekt einem anderen gleicht, sind auch Organisationssstrukturen einzigartig. Die Prozesse, Terminologie, Iterationsplanung und das Berichtswesen werden auf eine bestimmte eigene Art und Weise erledigt. Die Flexibilität einer Lösung im Hinblick auf das eingesetzte Verfahren (Vgl. Abschnitt 3.2) ist dabei ebenso zu berücksichtigen. Diesen spezifischen Bedürfnissen sollte Rechnung getragen werden.

Einfachheit

Einem Team ein einfaches und klar gegliedertes Planungs-, Nachverfolgungs- und Berichtssystem bereitzustellen ist wichtig für den breiten Einsatz in der Entwicklung. Am wichtigsten dabei ist: ein Werkzeug sollte niemals versuchen, die täglichen Meetings, persönliche

Kommunikation, regelmäßige Reviews und Retrospektiven zu ersetzen. Ein Werkzeug kann nur so gut wie das Verfahren und dessen praktische Anwendung durch Personen sein.

Skalierbarkeit

Ein agiles Werkzeug sollte gut mit wachsenden Projektstrukturen, Tausenden von Features, Fehlerbeschreibungen, Aufgaben und Tests zurechtkommen.

4.3.2 Best Practices

Auf Basis der in Abschnitt 4.3.1 definierten Erfolgsfaktoren für den Einsatz von Open Source-Projektmanagementsoftware werden nachfolgend die *Best Practices* abgeleitet und zugeordnet. Im Idealfall sollte eine Open Source-Projektmanagementsoftware diese vollständig abdecken. Dabei wird besonders die Situation kleiner Teams und Projekte berücksichtigt, so dass Skalierbarkeit und Multi-Projektmanagement eine untergeordnete Rolle spielen. Im Sinne des besseren Verständnisses wurde bei einigen Best Practices die fachüblichen englischen Bezeichnungen angefügt.

Iterativ-inkrementelle, Feature-getriebene Entwicklung

Diese Klasse der Anforderungen an eine Projektmanagementsoftware repräsentiert die Kernfunktionalität, die diese Werkzeuge bereitstellen sollten.

Anforderungsmanagement: Das Management der Anforderungen in Form von User Stories, Features und Tasks, ist bedeutend für den inkrementell-iterativen Entwicklungsprozess in agilen Verfahren (Vgl. Seite 74) und ist außerdem geeignet, den Problemen, die im Zusammenhang mit dem weitgehenden Dokumentationsverzicht in XP (Vgl. Seite 52) auftreten, wirksam zu begegnen.

User Stories erfüllen folgende Funktionen [Coh04]:

1. Sie werden für die Planung und als Erinnerung verwendet.
2. Sie dienen als Gesprächsgrundlage, um die Details der User Story herauszuarbeiten.

3. Sie werden als Testgrundlage verwendet, um zu bestimmen, ob eine User Story erfüllt ist.

Dabei spielt natürlich der Umfang eine wesentliche Rolle. So ist es oft erforderlich, User Stories in Tasks aufzuteilen, die ihrerseits konkrete, oft rein technische Aufgaben repräsentieren. Thematisch aufeinander bezogene User Stories sollten andererseits zu sog. Themes oder Epics zusammengefasst werden können. Dies ist insbesondere dann der Fall, wenn die Backlog-Priorität relativ gering ist, das Element also noch weit vom Realisierungszeitpunkt entfernt ist.

Release-/Iterationsplanung: Die Planung der für agile Verfahren typischen Iterationen (z.B. Sprint's in Scrum; Vgl. Seite 43) sollte flexibel und einfach möglich sein. Die Vergabe von Prioritäten für die einzelnen Elemente (Backlogs prioritization) ist dabei wichtiges Kriterium. Die Aufwandsschätzung und Kapazitätsplanung sollte ebenso unterstützt werden. Die Planung von Releases ist analog zu betrachten.

Release- und Iterationsplanung erfüllt drei Funktionen [Coh08]:

1. Ist hilfreich bei der Entscheidung des Produktverantwortlichen und des Teams, welcher Umfang an Funktionalität bis zu welchem Zeitpunkt umzusetzen ist. Frühe Releases tragen zu einem frühzeitigen Return-on-Investment bei.
2. Synchronisiert die Erwartungen an den Funktionsumfang mit anderen strategischen Unternehmensaktivitäten bezogen auf den Zeitrahmen für das Release.
3. Fungiert als Wegweiser für das Team, um nicht endlos von Iteration zu Iteration fortzuschreiten.

Projektfortschritt: Die Nachvollziehbarkeit des Projektfortschritts ist notwendig, um schnell und frühzeitig auf Probleme im Projektverlauf reagieren zu können. Außerdem hat das Management meist einen entsprechenden Informationsbedarf. In agilen Verfahren – speziell Scrum – wird der Projektfortschritt als noch zu erbringender Restaufwand grafisch dargestellt (Burndown Chart, Abbildung 3.5, Seite 44).

Verwaltung von Abhängigkeiten (Dependency Tracking): Mit klassischen Projektplanungswerkzeugen wird häufig viel Arbeit in die

Verwaltung von Abhängigkeiten investiert. Durch die auf ausgiebiger Kommunikation beruhenden kurzen Iterationen wird die Bedeutung von Abhängigkeiten zwischen einzelnen Elementen verringert. In [DS08] wird Joel Spolsky dazu wie folgt zitiert: „The trouble with Microsoft Project is that it assumes that you want to spend a lot of time worrying about dependencies... I've found that with software, the dependencies are so obvious that it's just not worth the effort to formally keep track of them.“ Eine einfach zu handhabende Verwaltung von Abhängigkeiten kann jedoch die Zusammenhänge, die in Softwareprojekt zweifelsohne existieren, transparenter und nachvollziehbarer machen. Insofern kann obiger Aussage nicht zugestimmt werden.

Management Software-Lebenszyklus

Die einzelnen Phasen des Software-Lebenszyklus greifen bei Anwendung agiler Verfahren eng ineinander. Da eine Software per se einen Lebenszyklus hat, ist die Bedeutung des Managements desselben systemimmanent.

Aufwandsschätzung/-verfolgung: Die Abschätzung der Aufwände (Estimation) ist in agilen Verfahren von zentraler Bedeutung. Sie bildet die Grundlage für die Kontrolle des Projektfortschritts. Die Erfassung und Verfolgung der tatsächlich geleisteten Aufwände (Effort Tracking) ist demgegenüber zwar untergeordnet, ist aber für die Abrechnung von Kundenprojekten auf Aufwandsbasis bedeutsam.

Praktiken: Auf den Einsatz von Praktiken als wichtige Erfolgsfaktoren, die zumeist im Zusammenhang mit XP genannt werden, wurde bereits weiter oben ausführlich eingegangen (Tabelle 3.1, Seite 61). Diese Praktiken werden üblicherweise in den regulären Entwicklungsprozess (z.B. Komponententests mit JUnit, Akzeptanztests mit Fit, Continuous Integration mit CruiseControl) und den dort vorhandenen Werkzeugen eingebettet³.

Durch *Versionskontrolle* Release-Stände festzuhalten und Änderungen am Code nachvollziehen zu können sind ebenfalls wichtige Aspekte in der Softwareentwicklung. Das gilt insbesondere bei gemeinsamer Verantwortlichkeit für den Code (Vgl. Seite 37). Softwareentwicklung von verteilten Teams ist ohne Versionskontrolle undenkbar.

³Für eine Vertiefung zum Thema sei auf [Hüt08] verwiesen.

Eine Projektmanagementssoftware sollte geeignete Schnittstellen z.B. zur Integration der Praktiken

- Testing
- Continuous Integration
- Coding Standards
- Versionskontrolle

in Form von Referenzen, Ergebnisreports oder Statusinformationen bereitstellen.

Fehlerverfolgung (Bugtracking): Obwohl agile Verfahren im Vergleich zu traditionellen Ansätzen für sich in Anspruch nehmen, die Qualität des Softwareprodukts zu steigern, ist die Fehlerfreiheit von Software nur in seltenen Fällen gegeben (siehe Diagramm *Quality* in Abbildung A.3). Eine integrierte Fehlerverfolgung ist vor allem in der Wartungsphase einer Software von großer Wichtigkeit, da schnelle Fehlerbeseitigung die Kundenzufriedenheit maßgeblich beeinflusst.

Nachvollziehbarkeit (Traceability): Alle wesentlichen Bearbeitungen innerhalb eines Projekts sollte eine Projektmanagementssoftware nachvollziehbar bereitstellen.

Funktionsübergreifende Teams

In einem Entwicklungsteam, in dem neben Entwicklern auch Tester, Experten der Fachdomäne, Projektmanager und weitere Rollen vertreten sind, muss eine Projektmanagementssoftware den Anforderungen dieser Gruppen gerecht werden.

Rollenprofile: Ein Werkzeug sollte diesem Umstand Rechnung tragen, indem entsprechende Profile für unterschiedliche Rollen existieren. Profilbezogen sollten spezifische Sichtweisen auf das Projekt (z.B. Management-Dashboard) realisiert werden.

Zufriedenheit (Team/Stakeholder): Die Akzeptanz einer Projektmanagementssoftware und damit die Zufriedenheit der Projektbeteiligten hängt wesentlich von der Erfüllung derer spezifischer Anforderungen ab. Eine Projektmanagementssoftware sollte von allen Projektbeteiligten gleichermaßen gut nutzbar sein.

Flexible Konfiguration

Dieser Komplex bezeichnet die Fähigkeit eines Systems, durch einfache Anpassung von Eigenschaften das Aussehen, den Funktionsumfang und die Funktionsweise gezielt auszuprägen.

Konfigurierbarkeit (Customizing): Der Möglichkeit, ein Werkzeug anzupassen, kommt deshalb entsprechend große Bedeutung zu, da das in der konkreten Projektsituation vorliegende *tatsächliche* Verfahren unterstützt werden soll und nicht umgekehrt – Anpassung des Verfahrens an das verwendete Werkzeug (Vgl. Seite 25).

Sicherheit (Security, Permissions): Ein komplexes Sicherheitssystem zur Erteilung diverser Rechte scheint aus Sicht agiler Verfahren nicht notwendig. Aus zwei Gründen ist dies jedoch zu erwägen: Erstens werden für verteilte arbeitende Teams web-basierte Lösungen eingesetzt, so dass der Zugriff darauf eingeschränkt werden muss. Zweitens können ebensolche Lösungen zu verschiedenen Zwecken (Entwicklung, Help-Desk, Wissensdatenbank etc.) eingesetzt werden, so dass eine differenzierte Vergabe von Zugriffsrechten sinnvoll ist.

Einfachheit

Eine Projektmanagementsoftware erfüllt keinen Selbstzweck. Sie sollte das Team in der täglichen agilen Entwicklungsarbeit unterstützen und keinen nennenswerten zusätzlichen Aufwand mit sich bringen.

Berichtswesen, Listen (Reporting): Den unterschiedlichen Benutzergruppen sollte eine Projektmanagementsoftware genau die Informationen bereitstellen, die diese benötigen. Projektleiter und das Management interessieren sich für den Projektfortschritt. Entwickler müssen auf Task-Listen zugreifen. Für Tester sind Testfälle interessant. Das Reporting sollte diese unterschiedlichen Aspekte berücksichtigen.

Einfache Handhabung (Online Hilfe): Die Handhabung einer Projektmanagementsoftware sollte schnell erlernt werden können. Eine übersichtliche und klar gegliederte Oberfläche ist dafür Voraussetzung. Bekannte und bewährte Konzepte (z.B. Wiki), die ebenso agil

wie der Entwicklungsprozess sind, helfen dabei. Funktionen, die selten benötigt werden, sowie überladene Werkzeuge erschweren die Handhabung deutlich. Eine Projektmanagementssoftware sollte über eine hinreichende Dokumentation (Online-Hilfe) verfügen.

Benachrichtigungen: Insbesondere bei verteilten Teams ist die systemgestützte Kommunikation von großer Bedeutung. Wichtige Informationen sollten ohne deren Interaktion zu den Empfängern gelangen. Eine Projektmanagementssoftware sollte gezielte Benachrichtigungen per E-Mail und den Bezug von RSS-Feeds ermöglichen.

Import/Export: Der Import von Daten aus anderen Systemen sollte möglich sein. Damit ist die Migration von Daten aus anderen Anwendungen oder Datenbeständen möglich. Der Export von Daten ist für die Weiterverarbeitung interessant.

Suchfunktion: Die Suche über den gesamten Datenbestand erleichtert den schnellen Zugriff auf benötigte Informationen. Das ist vor allem dann wichtig, wenn die Informationen nicht unstrukturiert – wie z.B. in einem Wiki – vorliegen.

Skalierbarkeit

Eine Projektmanagementssoftware sollte auch bei steigendem Datenvolumen performant auf Anfragen reagieren.

Linearer Ressourcenbedarf: Der Ressourcenbedarf einer Projektmanagementssoftware sollte linear zur Last wachsen. Doppelte Last sollte doppelten Ressourcenbedarf nach sich ziehen.

Systemwechsel: Eine Projektmanagementssoftware sollte problemlos auf leistungsfähigere Systeme übertragen werden können (vertikale Skalierbarkeit).

4.4 Zusammenfassung

Im Abschnitt 4.2 wurde der Frage nachgegangen, ob und in welchem Umfang Unterstützung durch Projektmanagementsoftware erforder-

lich ist. Dabei wurde festgestellt, dass bei der Anwendung agiler Verfahren kein Werkzeug die direkte Kommunikation der Projektbeteiligten ersetzen kann und darf. Es existieren jedoch eine Reihe von Gründen, weshalb Softwareunterstützung sinnvoll ist.

Erfolgsfaktoren für die Realisierung agiler Projekte wurden in Abschnitt 4.3.1 definiert. Daraus abgeleitet wurden in Abschnitt 4.3.2 die Anforderungen beschrieben, die eine Projektmanagementsoftware erfüllen sollte, um agile Projekte erfolgreich durchführen zu können.

Im Kapitel 5 werden nun relevante Systeme einer Evaluation unterzogen.

Kapitel 5

Evaluation von Open Source-Projektmanagementsoftware

Ein Mensch, der von jedem Ding den Preis und von keinem den Wert kennt.

(Oscar Wilde über Zyniker)

5.1 Vorgehen

Im Abschnitt 5.2.1 wird eine Kurzcharakteristik ausgewählter Open Source-Projektmanagementsoftware gegeben. Eine Auswahl relevanter Werkzeuge für den produktiven Einsatz wird in Tabelle 5.1 auf Seite 95 getroffen.

Mit Hilfe einer Nutzwertanalyse wird anschließend im Abschnitt 5.3 das Werkzeug ermittelt, das für agile Entwicklung den höchsten Nutzwert besitzt. Als Kriterien für die Nutzwertanalyse dienen die Erfolgsfaktoren aus Abschnitt 4.3.1. Die Anforderungen aus Abschnitt 4.3.2 werden zur Bewertung des jeweiligen Werkzeugs herangezogen.

5.2 Eingrenzung Open Source-Projektmanagementsoftware

Neben kommerziellen Produkten gibt es eine große Zahl von Open Source-Software, die sich in irgendeiner Weise mit Projektmanagement befasst. Die Grenzen zwischen den Anwendungsspektren sind fließend. Da für viele ScrumMaster und Product Owner Tabellenkalkulationen zu den beliebtesten Planungswerkzeugen gehören (Vgl.

[DS08], S. 3), kann auch OpenOffice zu dieser Kategorie gezählt werden.

Für die im Rahmen dieser Diplomarbeit durchgeführte Evaluation werden jedoch nur Werkzeuge betrachtet, die den agilen Entwicklungsprozess direkt unterstützen. Außerdem müssen die Werkzeuge produktiv eingesetzt werden können. Projekte, deren Entwicklung offensichtlich eingestellt wurde oder deren Funktionsumfang einen produktiven Einsatz nicht zulässt werden nicht untersucht.

5.2.1 Kurzcharakteristik ausgewählter Open Source-Projektmanagementsoftware

In den folgenden Abschnitten werden einige Open Source-Projektmanagementsoftware kurz charakterisiert, um einen groben Überblick über den Einsatzbereich, die Verwendbarkeit und Aktivität der jeweiligen Projekte zu geben. Die Anzahl der verfügbaren Werkzeuge ist vergleichbar mit der Vielfalt der Verfahren, Techniken und Methoden. Für die folgende Betrachtung wurde die Auswahl auf die in [Tar09] genannten Werkzeuge beschränkt. Da Agilo als Plugin auf der Projektmanagementsoftware Trac aufsetzt, wird diese ebenfalls betrachtet.

Die Aufstellung erhebt nicht den Anspruch vollständig zu sein und umfasst nur jene Werkzeuge, die über einen für die agile Entwicklung benötigten Funktionsumfang verfügen.

Agilefant

Mit Agilefant werden nach Aussage auf der Projekt-Homepage¹ die Aspekte der täglichen Arbeit, der langfristigen Produkt- und Release-Planung und des Projektmanagements agiler Projekte zusammengefasst. Ein *Backlog* wird zur Organisation der Entwicklungsarbeit mit Aufwandsschätzungen, Prioritäten und Verantwortlichkeiten eingesetzt. Dabei wird über Themes (deutsch: Themen) die Gruppierung des Backlog erreicht. Themes können zukünftigen Projekten und Iterationen zugeordnet werden. Ein Überblick der Produkt- und Projektentwicklung kann dabei schnell gewonnen werden. Für Projekte und Iterationen wird der Fortschritt mit Burndown-Diagrammen und -Metriken gemessen. Der Projektstatus wird mit den Ampelfarben

¹<http://www.agilefant.org>

(rot/gelb/grün) dargestellt. Teammitglieder werden als Ressourcen betrachtet und können einzelnen Projekten und Teams zugeordnet werden. Die Ressourcenverwaltung gestattet die Arbeitsbelastung jedes Teammitgliedes zu ermitteln. Diverse parallel laufende Projekte werden im *Development portfolio management* zusammengefasst. Die tägliche Arbeit, auch in unterschiedlichen Projekten und Iterationen kann pro Benutzer sichtbar gemacht werden. Schließlich wird in einem *Timesheet* die aufgewandte Arbeit verzeichnet. Agilefant steht unter der MIT License². Das Projekt wird aktiv entwickelt, die letzte Version wurde im März 2009 veröffentlicht.

XPlanner

Nach Aussage auf der Projekt-Homepage³ ist XPlanner ein Planungs- und Nachverfolgungswerkzeug für Teams, die Extreme Programming (XP) anwenden. Der Planungsprozess in XPlanner besteht aus der Auswahl der zu realisierenden Features (in Form von User Stories) durch den Kunden und deren Zuordnung zu Iterationen von etwa ein bis drei Wochen Dauer. Das Entwicklungsteam schätzt den benötigten Aufwand und teilt umfangreichere User Stories in Aufgaben auf. Dabei wird die Entwicklungsgeschwindigkeit des Teams in der vorangegangenen Iteration vergleichend herangezogen, so dass bei Überplanung des Teams der Umfang der ausgewählten User Stories gemeinsam mit dem Kunden neu verhandelt werden kann. XPlanner zeichnet sich durch einfache Handhabung aus und unterstützt neben XP ebenso Scrum. Es bietet, auch im Vergleich mit anderen Open Source-Werkzeugen, viele Features und Reports. Nachteilig ist jedoch der aufwändige Installationsprozess. Viele Open Source-Projekte verwenden XPlanner für die eigene Planung (Hibernate, JUnit, Log4J, Struts). Das Projekt steht unter der LGPL⁴.

Icescrum 2

Das Tool macht ausgiebigen Gebrauch von AJAX. IceScrum⁵ bietet einige interessante Features, die die Arbeit beschleunigen können. Das Tool implementiert Planning Poker, Burn Down Chart und bietet gute Drag-and-Drop-Techniken. Allerdings ist die Benutzbarkeit nicht

²<http://www.agilefant.org/wiki/display/AEF/License>, 29.01.2009

³<http://xplanner.org>

⁴<http://www.opensource.org/licenses/lgpl-2.1.php>, 29.01.2009

⁵<http://icescrum.org>

zufrieden stellend, da wesentliche Funktionen nur schwer zu erreichen sind, die Oberfläche insgesamt einen unfertigen Eindruck hinterlässt und das Tool über eine mangelnde Dokumentation verfügt. Das Projekt steht unter der GPL. Icescrum wird aktiv entwickelt, die letzte Version ist vom Februar 2009.

Trac

Trac⁶ ist in seiner Basisversion die Kombination aus einem Wiki und einem Ticket-System. Außerdem bietet das Tool gute Nachvollziehbarkeit der Aktivitäten (Timeline), integrierte Anbindung an das Versionskontrollsysteem Subversion sowie eine Meilensteinverwaltung. Mit letzterer können auch Timeboxen verwaltet werden. Die gute Konfigurierbarkeit des Werkzeugs und vor allem die Möglichkeit, das System durch eine Vielzahl von Plugins zu erweitern, zählen zu den Stärken des Werkzeugs. Damit stellt Trac das Tool mit dem breitesten Anwendungsspektrum dar, das durch Einsatz ausgewählter Plugins spezifisch für den agilen Entwicklungsprozess ausgeprägt werden kann. Trac ist lizenziert unter einer modifizierten BSD-Lizenz⁷. Das Projekt wird aktiv entwickelt, die aktuelle Version wurde im Februar 2009 veröffentlicht.

Der Bekanntheitsgrad von Trac ist vergleichsweise groß. In [Sut07] diskutiert Jeff Sutherland Trac als mögliches Tool für agile Projekte. Dabei hebt er die Features Wiki, Subversion-Anbindung und Burn Down Chart hervor. In [Hüt08] empfiehlt der Autor Trac als Werkzeug zur Projektkommunikation. Das Tool wird ebenso in einschlägigen Fachartikeln besprochen ([Huh07], [Göt06]).

Agilo

Agilo⁸ basiert auf Trac (siehe Seite 94) und ist als Trac-Plugin und als virtuelle Maschine (VMware) verfügbar. Agilo erweitert Trac um Features, die spezifisch für die agile Entwicklung sind. Agilo unterstützt User Stories, Requirements und die Trac-typischen Tickets, die als Tasks betrachtet werden können. Agilo bietet außerdem individuelle Backlogs, Burn Down Chart, Teamverwaltung, Kalkulation der verfügbaren Kapazitäten und zeichnet sich – wie Trac – durch eine

⁶<http://trac.edgewall.org/>

⁷<http://trac.edgewall.org/wiki/TracLicense>, 03.03.2009

⁸<http://www.agile42.com>

enorme Konfigurierbarkeit aus. Agilo wird nach der Apache Software License 2.0 verbreitet. Das Projekt wird aktiv entwickelt, neue Versionen werden monatlich veröffentlicht. Das Projekt steht unter der Apache License, 2.0⁹. Die aktuelle Version wurde im Februar 2009 veröffentlicht.

FireScrum

FireScrum¹⁰ unterstützt agiles Management von Scrum-Teams. Der Fokus der Entwickler liegt auf der Unterstützung von Teams, die nicht täglich an einem Ort zusammenarbeiten. Im wesentlichen soll der Projektstatus transparent gemacht werden. FireScrum bietet dem Team die Verwaltung eines Product Backlog (Vgl. Seite 43), eines Sprint Backlog (Vgl. Seite 43) und einer Aufgabenverwaltung in Form eines Task Board. Das Projekt steht unter der GPL-Lizenz¹¹. Die Dokumentation von FireScrum beschränkt sich auf Screencasts. Laut Homepage des Projekts ist FireScrum noch in der Entwicklung und die geplante Funktionalität ist nicht vollständig implementiert. Auf Grund des aktuellen Entwicklungsstandes ist der Wert von FireScrum für den praktischen Einsatz von verteilten Scrum-Teams zurzeit noch fraglich.

5.2.2 Entscheidungsalternativen

Den im Abschnitt 5.2.1 charakterisierten Lösungen im Bereich Open Source-Projektmanagementsoftware kommt eine teilweise stark zu differenzierende Bedeutung zu. Auf diesen Umstand wird in Tabelle 5.1 eingegangen, indem die Breite der zu untersuchenden Systeme auf diejenigen eingeschränkt wird, die momentan von praktischer Bedeutung ist. Die Daten zu Status, Aktivität und Rating wurden der Plattform *Sourceforge*¹² bzw. – bei Agilo – *Freshmeat*¹³ entnommen (Stand 08.03.2009). Als Kriterium wird bei der Bewertung der Projektstatus herangezogen. Die Daten zu Aktivität, Rating und Treffer dienen an dieser Stelle lediglich der Information, um die Bedeutung der jeweiligen Werkzeugs zu illustrieren.

Die in Tabelle 5.1 verwendeten Spalten haben folgende Bedeutung:

⁹<http://www.apache.org/licenses/LICENSE-2.0.html>, 03.03.2009

¹⁰<http://www.firescrum.com/>

¹¹<http://www.gnu.org/licenses/gpl.html>, 29.01.2009

¹²<http://www.sourceforge.net>

¹³<http://www.freshmeat.net/>

Status Projektstatus, in dem sich das Projekt befindet. Nur Projekte im Status *Production/Stable* oder *Mature* werden im weiteren betrachtet.

Aktivität Der für das Projekt angegebene Aktivitätsindex auf Sourcforge. Der Aktivitätsindex spiegelt die momentane Projektaktivität bezogen auf die letzte Woche wieder.

Rating Der Rating-Index auf Freshmeat entspricht einer Benutzerbewertung auf einer bis 10 reichenden Skala.

Treffer Anzahl der gefundenen Suchergebnisse einer Google-Anfrage, durchgeführt am 08.03.2009. Suchbegriff ist der jeweilige Projektname eingeschlossen in doppelte Hochkommata ("’’), der Suchzeitraum ist eingeschränkt auf das letzte Jahr.

NWA Die für die anschließende Nutzwertanalyse in Frage kommenden Werkzeuge sind mit „ja“ gekennzeichnet.

Tabelle 5.1: Relevanz von Open Source-Projektmanagementsoftware

System	Status	Aktivität	Rating	Treffer	NWA
XPlanner	Production/Stable	96,19 %	7,98	9.350	ja
Agilefant	Production/Stable	99,26 %	–	267.000	ja
Icescrum	Alpha	82,93 %	–	433	nein
Trac	Production/Stable	Unranked	8,84	329.000	ja
Agilo	Mature	–	8,62	5.780	ja
FireScrum	Planning	85,98 %	–	101	nein

Als Entscheidungsalternativen für die Nutzwertanalyse wurden die Werkzeuge *XPlanner*, *Agilefant*, *Trac* und *Agilo* ausgewählt.

5.3 Nutzwertanalyse Open Source-Projektmanagementsoftware

Die Zielsetzung einer Nutzwertanalyse wird in [Nik02] so formuliert: „Bei der Nutzwertanalyse handelt es sich um eine Methode, die verschiedene Entscheidungsalternativen vergleicht. Sie ist besonders geeignet, wenn ‚weiche‘, also in Geldwert oder Zahlen nicht darstellbare Kriterien als Entscheidungsgrundlage vorliegen. Das Ergebnis der [...] Analyse ist für jede der Alternativen eine Zahl, die den Nutzwert darstellt. Die „beste“ Alternative erhält dabei den höchsten Wert.“

Das Ergebnis der Nutzwertanalyse hängt wesentlich von der Vergleichbarkeit der Entscheidungsalternativen und der Auswahl der Kriterien sowie deren Gewichtung ab. Es handelt sich insofern nicht um ein objektives Verfahren.

Die Entscheidungsalternativen wurden bereits in Tabelle 5.1 festgelegt. In den folgenden Abschnitten werden diese Alternativen bewertet. Die Kriterien dafür werden zuvor in Abschnitt 5.3.1 definiert.

5.3.1 Bewertungskriterien

Für die im Rahmen der Nutzwertanalyse auszuwählende Entscheidungsalternative werden in Tabelle 5.2 die im Abschnitt 4.3.2 identifizierten Anforderungen mit einer Wichtigkeit versehen.

Tabelle 5.2: Auswahl der Bewertungskriterien

	Kriterien	Wichtigkeit
1	Anforderungsmanagement	sehr hoch
2	Release-/Iterationsplanung	sehr hoch
3	Projektfortschritt	hoch
4	Verwaltung von Abhängigkeiten	hoch
5	Aufwandsschätzung/-verfolgung	hoch
6	Praktiken	sehr hoch
7	Fehlerverfolgung	hoch
8	Nachvollziehbarkeit	hoch
9	Rollenprofile	mittel
10	Zufriedenheit	mittel
11	Konfigurierbarkeit	sehr hoch
12	Sicherheit	mittel
13	Berichtswesen, Listen	hoch
14	Einfache Handhabung	sehr hoch
15	Benachrichtigungen	mittel
16	Import/Export	mittel
17	Suchfunktion	mittel
18	Linearer Ressourcenbedarf	mittel
19	Systemwechsel	mittel
20	Aktivität	sehr hoch

Als weiteres Kriterium kommt die Aktivität der Open Source-Projektmanagementsoftware hinzu, da aktive Projektentwicklung bzw. stabile Software eine Voraussetzung für den produktiven Einsatzes

einer entsprechenden Open Source-Projektmanagementsoftware darstellt (Vgl. Seite 65).

Die Bewertung der Kriterien wurde unter Bezug auf die Ergebnisse des Abschnitts 3.9 vorgenommen. Für die Nutzwertanalyse werden nur die Kriterien mit sehr hoher Wichtigkeit herangezogen. Diese Kriterien sind für die Auswahl einer Open Source-Projektmanagementsoftware aus den Entscheidungsalternativen im Rahmen dieser Diplomarbeit relevant.

Natürlich können in konkreten Projektsituationen andere Präferenzen existieren, die eine andere Auswahl der Anforderungen und eine Anpassung der Nutzwertanalyse erforderlich machen. So kann beispielsweise die Aktivität eines Projektes von wesentlich geringerem Interesse sein, wenn die verwendete Projektmanagementsoftware über große Stabilität verfügt und/oder Kapazitäten vorhanden sind, die verwendete Software selbst anzupassen und weiterzuentwickeln.

Für die Nutzwertanalyse werden aus Tabelle 5.2 die Bewertungskriterien

- Anforderungsmanagement
- Release-/Iterationsplanung
- Praktiken
- Konfigurierbarkeit
- Einfache Handhabung
- Aktivität

ausgewählt.

Ermitteln der Gewichtungsfaktoren

Die Gewichtungsfaktoren der einzelnen Kriterien werden in Tabelle 5.3 zueinander in Bezug gebracht.

Anforderungsmanagement und Konfigurierbarkeit rangieren in der Gewichtung an vorderer Position. Anforderungsmanagement ist das Kernthema bei der Durchführung agiler Verfahren. Flexible Konfiguration ist notwendig, um das Werkzeug an das *tatsächlich* verwendete Verfahren anzupassen.

Die Integration der verwendeten Praktiken steht nicht im Mittelpunkt einer Open Source-Projektmanagementsoftware – eine Integration ist gleichwohl wünschenswert.

Die Legende in Tabelle 5.4 erläutert den Bewertungsindex, mit dem das Zeilenkriterium zum Spaltenkriterium in Beziehung gesetzt wird.

Tabelle 5.3: Gewichtungsfaktoren ermitteln

Kriterien	1	2	3	4	5	6	Gewicht	Faktor
1 Anforderungsmanagement	2	2	1	1	2		8	0,267
2 Release-/Iterationsplanung	0		2	1	1	1	5	0,167
3 Praktiken	0	0		0	1	1	2	0,067
4 Konfigurierbarkeit	1	1	2		2	1	7	0,233
5 Einfache Handhabung	1	1	1	0		2	5	0,167
6 Aktivität	0	1	1	1	0		3	0,100
						Summe	30	1

Tabelle 5.4: Legende Ermittlung Gewichtungsfaktoren

- | | |
|---|---|
| 2 | Zeilenkriterium wichtiger als Spaltenkriterium |
| 1 | Zeilenkriterium und Spaltenkriterium gleich wichtig |
| 0 | Zeilenkriterium unwichtiger als Spaltenkriterium |

Skala der Zielerfüllungsfaktoren

Die zumeist subjektive Einschätzung der Erfüllungsgrade der einzelnen Kriterien durch die jeweiligen Werkzeuge ergeben die Bewertungen je Kriterium und Werkzeug in den Tabellen 5.6 bis 5.11, die schließlich in Tabelle 5.12 zusammengefasst werden und den Nutzwert der jeweiligen Open Source-Projektmanagementsoftware ergibt.

Für die Einschätzung der Entscheidungsalternativen wird in Tabelle 5.5 eine Skala aufgestellt. Mit deren Hilfe ist es möglich, die Kriterien für jedes Werkzeug hinreichend fein abgestuft zu bewerten. Je besser ein Werkzeug die Kriterien erfüllen kann umso größer ist der jeweilige Zielerfüllungsfaktor. Nach dieser Bewertungsskala wird im Folgenden vorgegangen.

5.3.2 Bewertung Anforderungsmanagement

Die folgenden Abschnitte kommentieren die Eignung der einzelnen Werkzeuge für das Anforderungsmanagement. Die Bewertung wird in Tabelle 5.6 zusammengefasst.

XPlanner erfasst User Stories und Tasks in einer Wiki-Syntax. Dadurch ist auch eine Verknüpfung zu externen Inhalten (als URL)

Tabelle 5.5: Skala Zielerfüllungsfaktoren

Skala	0-2 schlecht	3-5 mittel	6-8 gut
1 Anforderungsmanagement	User Stories, Features und Tasks können transparent und nachvollziehbar erfasst, priorisiert und geändert werden.		
2 Release-/Iterationsplanung	Planung und Auswertung von Iterationen, Zuordnung der Funktionalität zu Timeboxen oder Meilensteinen.		
3 Praktiken	Integration von Praktiken der Softwareentwicklung (Testing, Continuous Integration, Coding Standards, Versionskontrolle).		
4 Konfigurierbarkeit	Flexible Anpassung des Funktionsumfangs, der Abläufe und des Formalisierungsgrades.		
5 Einfache Handhabung	Geringe Lernkurve zu überwinden, gute Dokumentation (Online-Hilfe), keine oder geringe Systemvoraussetzungen.		
6 Aktivität (Releases in den letzten 12 Monaten)	0	1 – 4	> 4

möglich. Eine Priorität kann vergeben werden. Zu jeder User Story können Tasks angelegt werden, die die eigentlichen Aufgaben repräsentieren. In Tasks ist es möglich, die Aufwandsschätzung als ideale Stunden sowie die tatsächlich geleisteten Stunden zu erfassen. Den Tasks werden auch die Informationen über den Entwickler zugeordnet. Sowohl User Stories als auch Tasks verfügen über eine Anzeige des Fortschritts. User Stories gelten als abgeschlossen, wenn alle zugehörigen Tasks bearbeitet wurden. Das Werkzeug verfolgt einen einfachen, klar strukturierten Ansatz. Abbildung 5.1 zeigt eine User Story mit vier zugeordneten Tasks.

Agilefant ordnet die User Stories Iterationen zu. Die Erfassung erfolgt auch hier in einer Wiki-Syntax. Priorität und Status sowie die Verantwortlichkeit können zugeordnet werden. Die Schätzung des Aufwands und die Erfassung der geleisteten Arbeit ist analog zu XPlanner. Außerdem ist die Zuweisung mehrerer Themes und eines Iterationsziels möglich. Der Fortschritt kann ebenfalls schnell abgelesen werden. Sehr praktisch ist die Möglichkeit, die Eigenschaften

The screenshot shows the XPlanner interface. At the top, there's a navigation bar with links for Top, Project, and Iteration. Below that is a tree view of the project structure: Project 1 > Sprint 2 > Story 1. The main area displays 'Story: Story 1 [id=397]' with a progress bar at 43.0%. Below the story title is a 'Story 1 description' section. Underneath that are details for the story: Priority (1), Estimated Hours (43.0 (4.0)), Actual Hours (43.0), Remaining Hours (0.0), Tracker (Luke), Last Update (2008-07-26 18:38), Disposition (Planned), and Status (Implemented). At the bottom of the story card is a table showing four tasks:

Actions	ID	Task Name	Type	Progress	Acc.	Ori.	Est.	Rem.	Disp.	Type
	430	Task 1.1	Debt	<div style="width: 40%;">4.0</div>	1	4.0	4.0	0.0	Planned	Debt
	608	Task 1.2	Feature	<div style="width: 90%;">9.0</div>	S	8.0	9.0	0.0	Discovered	Feature
	609	Task 1.3	Feature	<div style="width: 16.0%;">16.0</div>	M	16.0	16.0	0.0	Discovered	Feature
	610	Task 1.4	Feature	<div style="width: 14.0%;">14.0</div>	J	12.0	14.0	0.0	Discovered	Feature

At the very bottom of the screen are links for Edit, Delete, Move/Continue, Create Task, Export, History, and Print.

Abbildung 5.1: XPlanner Story-Task-Beziehung [Glo08]

mehrerer Backlog Items in einem Schritt ändern zu können. In Abbildung 5.2 wird die Erfassung eines Backlog Items gezeigt.

The screenshot shows the Agilefant application's backlog item editor. The top bar has tabs for Edit, Progress, and Spent Effort. The main area is titled 'BLI:9' and contains a rich text editor with a toolbar. The text in the editor reads: 'As a support agent, I'd like to search for e-mail-addresses to find the customers file if an e-mail is sent in.' Below the editor are several input fields and dropdown menus:

- Reference ID: BLI:9
- Name: (empty)
- Description: (empty)
- Created by: demo on 2009-03-21
- Original estimate: 5h (reset)
- Effort left: 2h (e.g. "2h 30min" or "2.5")
- State: Started
- Backlog: Sprint 1
- Iteration goal: (none)
- Priority: undefined
- Responsibles: jt
- Themes: Algo 2

At the bottom right are Save and Save & Close buttons, and standard window control buttons (Delete, Cancel, Close).

Abbildung 5.2: Backlog Item in Agilefant

Trac (und Agilo) arbeiten beide auf Basis von Tickets, die User Stories und Tasks repräsentieren. Auch hier wird mit einer Wiki-Syntax gearbeitet. Die Erfassung umfangreicher User Stories und Epics ist ebenso im integrierten Wiki möglich. Die in Trac durchgängig gegebene Möglichkeit, einzelne Inhalte im System miteinander zu verlinken (z.B. Wiki-Seiten mit Tickets und umgekehrt), schafft Nachvollziehbarkeit, Übersicht und Transparenz. Trac bietet einen generischen Ansatz, da das Werkzeug nicht speziell für agile Entwicklung ent-

worfen wurde. Die Darstellung des Fortschritts ist auf der Ebene der User Stories und Tasks nur durch Vergleich der Schätzung mit der verbleibenden Zeit möglich. Um in Trac Schätzungen verarbeiten zu können, wird allerdings ein entsprechendes Plugin benötigt¹⁴. Besonders hervorzuheben ist die Möglichkeit, beliebige Dateien an Tickets und Wiki-Seiten anzuhängen. Dadurch können Informationen konsistent zur Verfügung gestellt werden. Abbildung 5.3 zeigt ein Ticket mit einigen plugin- bzw. benutzerdefinierten Feldern.

The screenshot shows a Trac ticket interface for 'Ticket #71 (new User Story)'. The ticket details are as follows:

Reverse search e-mail to name		Opened 15 seconds ago	Last modified 3 seconds ago
Reported by:	anonymous	Owned by:	Andreas
Priority:	major	Milestone:	Sprint 1
Component:	Sonstiges	Keywords:	
Cc:		Termin:	
Test Case Result:	pass	verfügbar ab:	
Estimated Number of Hours:	0		
Billable?:	yes	Total Hours:	0
Description			
As a support agent, I'd like to search for e-mail-addresses to find the customers file if an e-mail is sent in. Reply			

Abbildung 5.3: Ticket in Trac

Agilo spezialisiert diesen generischen Ansatz und erweitert Trac um Scrum-typische Features. Es wird dabei eine hierarchische Struktur verwendet, in der Requirements, User Stories, Tasks und Bugs unterschieden werden. Als einziges Werkzeug im Vergleich bietet Agilo ausgereifere Mechanismen zur Schätzung und Aufwandsdarstellung. Neben der zeitbasierten Erfassung können für Requirements *Business Value Points* vergeben werden. User Stories verwenden *User Story Points*, die auf Fibonacci-Zahlen ([Coh08], S. 52) beruhen. Außerdem verfügen sie über eine Möglichkeit, mit der Eigenschaft *Story Priority* die Priorität gemäß der Kundenzufriedenheit nach dem Kano-Modell festzulegen ([Coh08], S. 112). Abbildung 5.4 zeigt eine User Story mit Referenzen auf Requirement und Tasks.

¹⁴<http://trac-hacks.org/search?q=estimation&noquickjump=1&wiki=on> 28.03.2009

User Story #7 (new)

Improve the integrated help system

Opened 5 days ago
Last modified 102 minutes ago

Reported by:	admin	Owned by:	
Keywords:	n.a.	User Story Points:	8
Sprint:	Sprint 4	Story Priority:	Mandatory
Total Remaining Time:	28.0h	Estimated Remaining Time:	n.a.

Description

As a user of the tool I would like to have an integrated help system that allows me to get information contextualized to the part of the tool in which I am working, so that I will spend less time searching for answers Reply

References

Referenced by:
← Requirement (#6): Need to increase usability

References:
→ Task (#12): represent result (Remaining Time: 6)
→ Task (#10): Searchbox (Owner: antonio Remaining Time: 10)
→ Task (#11): Search Base (Owner: lucas Remaining Time: 12)

Abbildung 5.4: User Story in Agilo

Fazit

Die Bewertung der Eignung der Werkzeuge ist in Tabelle 5.6 dargestellt. Agilefant zeichnet sich durch einen sauber strukturierten Ansatz aus, der gute praktische Unterstützung in der agilen Projektarbeit verspricht. Vorteilhaft ist bei Agilo das integrierte Wiki, die Möglichkeit Dateien an Tickets und Wiki-Seiten zu binden und die Möglichkeit Inhalte des Systems nahezu beliebig zu verlinken.

Tabelle 5.6: Bewertung Anforderungsmanagement

XPlanner	5
Agilefant	7
Trac	5
Agilo	8

5.3.3 Bewertung Release-/Iterationsplanung

Die folgenden Abschnitte kommentieren die Eignung der einzelnen Werkzeuge für die Release- und Iterationsplanung. Die Bewertung wird in Tabelle 5.7 zusammengefasst.

XPlanner ermöglicht keine Release-Planung. Als Planungsmöglichkeit können Iterationen geplant und ihnen User Stories zugeordnet

werden. Pseudo-Iterationen können als Ersatz für Releases und Backlogs verwendet werden¹⁵. Abbildung 5.5 veranschaulicht die einfach gehaltene Organisationsstruktur in XPlanner (Projekt → Iterationen → User Stories → Tasks). Zur Darstellung des Iterationsfortschritts bietet XPlanner diverse Diagramme an (siehe Abbildung A.4 auf Seite xxx).

Actions	ID	Iteration	Start Date	End Date	Days Wrk.	Stories
	375	Sprint 1	2008-06-18	2008-07-29	1,0	2
	485	Sprint 2	2008-08-13	2008-08-26	0,0	1

2 items found, displaying all items.1

Current Iteration

[Edit](#) | [Delete](#) | [Create Iteration](#) | [People](#) | [Export](#) | [History](#) | [Print](#)

Notes:

user: Luke [Logout](#)

Abbildung 5.5: XPlanner Projekt-Iterations-Beziehung [Glo08]

Agilefant verfügt über eine sehr ausgereifte Release-Planung. Diese werden als Projekte definiert. Die verwendete Struktur (Produkt → Projekte → Iterationen → Backlog Items → Todo's) ist in Abbildung 5.6 dargestellt (ohne Backlog Items und Todo's; eine Ansicht eines Sprint's ist in Abbildung A.5 auf Seite xxxi dargestellt). Der jeweilige Iterationsfortschritt wird als Burndown-Diagramm dargestellt.

Abbildung 5.6: Agilefant Produkt-Projekt-Iterations-Beziehung

¹⁵<http://www.xplanner.org/planning.html>, 16.03.2009

Trac plant Releases und Iterationen basierend auf einer Roadmap. In dieser werden Meilensteine angelegt, welche die Releases und Iterationen repräsentieren. Diese können allgemein als Backlogs aufgefasst werden. User Stories können in Form von Tickets diesen zugeordnet werden. Die zugrunde liegende Struktur ist dadurch relativ einfach (Milestone → Ticket (User Story/Feature/Task/Bug/...)). Abbildung 5.7 zeigt einen Meilenstein in der Roadmap. Die Möglichkeit, Trac durch Plugins zu erweitern, kann auch für die Iterationsplanung genutzt werden. Mit *agile-trac*¹⁶ steht ein Plugin dafür zur Verfügung. Für die Darstellung des Fortschritts einer Iteration steht mit dem *Scrum Burndown Plugin*¹⁷ ebenfalls eine Erweiterung bereit.

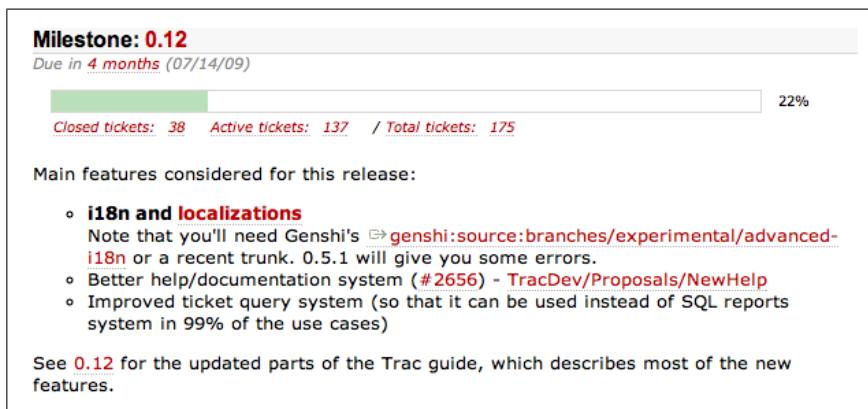


Abbildung 5.7: Meilenstein in Trac

Agilo unterstützt die Verwendung von Backlogs (Product Backlog, Sprint Backlogs und weitere, z.B. Bug Backlog, Impediment Backlog) zur Iterationsplanung. Ähnlich wie in Trac werden Meilensteine zur Release-Planung verwendet und in einer Roadmap dargestellt. Abbildung 5.8 zeigt eine solche Roadmap. Diesen Meilensteinen können ein oder mehrere Sprint's zugeordnet werden. Agilo bietet damit eine für agile Verfahren – vor allem für Scrum – wesentlich besser geeignete Struktur als Trac (ohne die dort möglichen Erweiterungen durch entsprechende Plugins). Die Struktur (Meilenstein → Sprint → Requirements → User Stories → Task's) von Agilo ist hervorragend für die Planung agiler Projekte geeignet. Für die Anzeige des Iterationsfortschritts steht für jeden Sprint eine umfangreiche Ansicht zur Verfügung. Abbildung A.7 (Seite xxxii) zeigt einen Sprint-Backlog mit Burndown-Diagramm.

¹⁶<http://www.agile-trac.org/>, 21.03.2009

¹⁷<http://trac-hacks.org/wiki/ScrumBurndownPlugin>, 21.03.2009

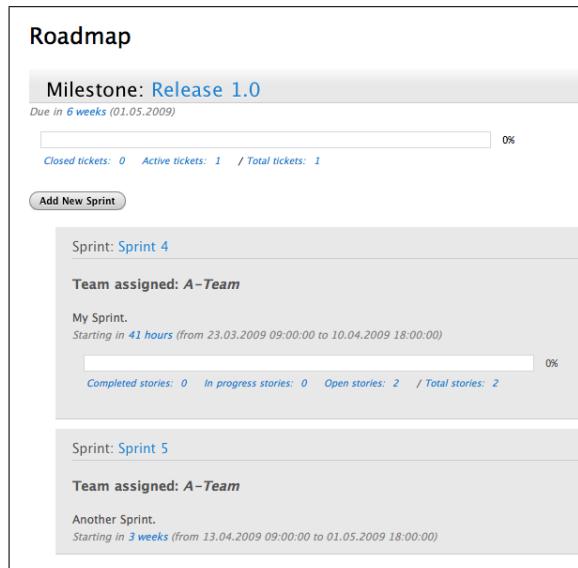


Abbildung 5.8: Roadmap in Agilo

Fazit

Tabelle 5.7 fasst die Bewertung der Werkzeuge zur Release- und Iterationsplanung zusammen. Die Planungsmöglichkeiten in Agilefant wirken ausgereift und praxisnah. Besonders die Möglichkeit Produkte und Projekte (Releases) zu verwalten ist günstig. Bei Agilo und Trac wäre dies wünschenswert.

Tabelle 5.7: Bewertung Release-/Iterationsplanung

XPlanner	3
Agilefant	7
Trac	5
Agilo	6

5.3.4 Bewertung Praktiken

Da in agile Projekten die eingesetzten Praktiken einen entscheidenden Erfolgsfaktor darstellen (Vgl. Seite 60), ist interessant, ob die Praktiken

- Testing
- Continuous Integration

- Coding Standards
- Versionskontrolle

in die Werkzeuge integriert werden können oder diese anderweitige Unterstützung bieten? Tabelle 5.8 fasst die Bewertung zusammen.

XPlanner und Agilefant bieten keine Unterstützung für die oben genannten Praktiken und integrieren diese nicht. Lediglich *Akzeptanztests* können von Stories abgeleitet und Testszenarios in Tasks oder Backlog Items spezifiziert werden.

Trac und Agilo bieten eine zum Teil sehr weitreichende Integration von Praktiken. Anmerkungen zu den einzelnen Praktiken folgen in den nächsten Abschnitten. Da Agilo auf Trac aufbaut gelten die Ausführungen für beide Werkzeuge.

Testing als Integration eines Testing-Frameworks ist in Trac nicht vorgesehen, obwohl die Integration von Fit seit geraumer Zeit diskutiert wird¹⁸. Für Akzeptanztests steht das *TestCaseManagement-Plugin* zur Verfügung¹⁹. Damit können Testfälle in einfachen XML-Dateien geschrieben werden. Testsuites, ebenfalls als XML-Dateien, fassen diese zusammen. Die Testdateien werden im Versionskontrollsystem Subversion (siehe weiter unten) gespeichert. Mit Hilfe des Plugins werden dann aus den Testfällen (oder ganzen Testsuites) Tickets generiert. Dieses Vorgehen sichert zumindest die Wiederverwendbarkeit der Testfälle.

Continuous Integration ist mit dem Plugin *Bitten*²⁰ möglich. Bitten integriert sich in die Trac-Oberfläche wie Abbildung 5.9 zeigt.

Coding Standards bezeichnen einen Satz von Regeln und Vereinbarungen, die innerhalb eines Projektes als verbindlich gelten. Die Definition dieser Regeln ist mit Trac und Agilo leicht zu erreichen, da beide Werkzeuge über ein integriertes Wiki verfügen. So kann sehr einfach eine Wiki-Seite (z.B. *CodingStandards*) eingerichtet werden²¹.

¹⁸<http://trac.edgewall.org/wiki/FIT>, 21.03.2009

¹⁹<http://trac-hacks.org/wiki/TestCaseManagementPlugin>, 23.03.2009

²⁰<http://bitten.edgewall.org/>, 23.03.2009

²¹Beispiel einer CodingStandards-Definition auf einer Trac-Projektseite: <http://trac-seagullproject.org/wiki/Standards/CodingStandards>, 06.04.2009

The screenshot shows the 'Build Status' page in Trac. It displays two main sections: '0.11-stable' and 'Trunk'. Each section has a 'Latest builds' table with four rows. The '0.11-stable' section shows builds for 'darwin-py23', 'linux-py24', 'linux-py25', and 'linux-py26'. The 'Trunk' section shows builds for 'linux-py24', 'linux-py25', and 'linux-py26'. Each build row contains the build number, date, time, environment (e.g., mac, bento), and a status indicator (Success). A checkbox for 'Show deactivated configurations' and an 'Update' button are located at the top right of the '0.11-stable' section.

Build Status				
0.11-stable				
Builds of the stable branch.				
Latest builds				
[7956] by jonas 03/18/09 21:26:49 <i>0.11-stable: Moving branch back into dev mode</i>	darwin-py23 03/21/09 19:21:22 mac (83.219.114.65) Darwin 9.6.0 / i386 Success	linux-py24 03/18/09 21:30:54 bento (88.198.140.132) Linux 2.6.27-11-server / i686 Success	linux-py25 03/18/09 21:35:39 bento (88.198.140.132) Linux 2.6.27-11-server / i686 Success	linux-py26 03/18/09 21:36:46 bento (88.198.140.132) Linux 2.6.27-11-server / i686 Success
<input type="checkbox"/> Show deactivated configurations	<input type="button" value="Update"/>			
Trunk				
Builds of the main development branch.				
Latest builds				
[7953] by jonas 03/18/09 20:21:53 <i>trunk: Backported [7952] from 0.11-stable (#4934)</i>	linux-py24 03/18/09 20:30:01 bento (88.198.140.132) Linux 2.6.27-11-server / i686 Success	linux-py25 03/18/09 20:46:16 bento (88.198.140.132) Linux 2.6.27-11-server / i686 Success	linux-py26 03/18/09 20:47:34 bento (88.198.140.132) Linux 2.6.27-11-server / i686 Success	

Abbildung 5.9: Build Status in Trac

In dieser Seite werden sämtliche Vereinbarungen erfasst, so dass neuen Teammitglieder die Adaption der Regeln leicht fallen sollte. Eine Integration von zusätzlichen Werkzeugen, die automatisch den Code gegen die Coding Standards prüfen, ist jedoch nicht möglich.

Versionskontrolle integrieren beide Werkzeuge nahtlos. Als Versionskontrollsystem wird Subversion (SVN)²² verwendet. Wie Abbildung 5.10 zeigt, kann auf die Versionsstände des Quellcodes in allen Modulen zugegriffen werden. Auf die Versionsstände kann einfach verlinkt werden. Ebenso ist es möglich, in den Notizen, die beim einchecken angegeben werden können, auf Elemente in Trac (Tickets, Wiki-Seiten) zu verlinken. Mit Hilfe eines post-commit Hook-Scripts können beim erfolgreichen einchecken direkt Aktionen auf den Trac-Elementen ausgeführt werden²³. So stehen innerhalb der Oberfläche alle Information aus der Versionskontrolle zur Verfügung. Auch Versionsunterschiede können angezeigt werden (siehe Abbildung A.8 auf Seite xxxiii). Nachteilig ist jedoch, dass die Subversion-Installation auf der gleichen Hardware wie das Trac-System installiert sein muss.

²²<http://subversion.tigris.org/>

²³<http://www.aptgetupdate.de/2007/11/07/trac-subversion-hook-script-benutzen/>, 21.03.2009



Abbildung 5.10: Versionskontrolle in Trac

Fazit

Die Integration ist mit Trac und Agilo prinzipiell möglich und bei Continuous Integration mit Bitten und Versionskontrolle mit Subversion sehr gut gelungen. XPlanner und Agilefant bieten in diesem Zusammenhang nichts nennenswertes. Die Bewertung fasst Tabelle 5.8 zusammen.

Tabelle 5.8: Bewertung Praktiken

XPlanner	1
Agilefant	1
Trac	6
Agilo	6

5.3.5 Bewertung Konfigurierbarkeit

Die folgenden Abschnitte kommentieren die Möglichkeit, die einzelnen Werkzeuge flexibel zu konfigurieren. Interessant ist ebenfalls, ob der Funktionsumfang der Werkzeuge einfach erweitert werden kann. Die Bewertung wird in Tabelle 5.9 zusammengefasst.

XPlanner bietet einige Möglichkeiten der Anpassung, die allerdings nicht über die Oberfläche möglich sind. Das Aussehen kann über eine CSS-Datei modifiziert werden. Externe Wiki-Seiten können angebunden werden. Die Anpassung des Login-Mechanismus ist möglich. Diagramme können modifiziert werden. Über ein SOAP-Interface kann XPlanner in Fremdanwendungen eingebunden werden.

Agilefant beschränkt die Anpassung auf die Möglichkeit die Ausgabe der Aufwände ein- oder auszuschalten sowie Schwellwerte für das Lastdiagramm der Teammitglieder zu beeinflussen.

Trac bietet seit Version 0.11 ein integriertes Administrationsmodul, mit dem umfangreiche Anpassungen vorgenommen werden können. Abbildung 5.11 zeigt das Administrationsmodul. In früheren Versionen musste dazu ein Plugin installiert oder die Konfiguration in einer Konfigurationsdatei vorgenommen werden. Mit dem Administrationsmodul können sämtliche Eigenschaften der Tickets (z.B. Priorität, Typen, etc.) beeinflusst werden. Über eine Vielzahl von Plugins kann die Funktionalität des Werkzeugs an die spezifischen Projektbedürfnisse angepasst werden²⁴. Für Version 0.11 stehen aktuell 204 Plugins zum Download bereit (Stand: 23.03.2009). Die Installation erfolgt zumeist über Python-Eggs²⁵. Außerdem ist die Integration von Trac in Fremdsysteme möglich. So kann Trac beispielsweise über den Mylyn/SOC/2006/Trac Connector in Eclipse eingebunden werden²⁶.

The screenshot shows the 'Administration' section of the Trac interface. On the left, there is a sidebar with various links: General, Basic Settings, Logging, Permissions, Plugins, Blog, Settings, Scrum Burndown, Plugin, Settings, Ticket System, Components, and Milestones (which is currently selected). The main content area is titled 'Manage Milestones'. It contains a table with two rows:

Name	Due	Completed	Default
Sprint 1	01/18/09	01/18/09 12:03:36	<input checked="" type="radio"/>
Sprint 2	02/18/09		<input type="radio"/>

Below the table are two buttons: 'Remove selected items' and 'Apply changes'. A note at the bottom states: 'You can remove all items from this list to completely hide this field from the user interface.'

Abbildung 5.11: Administrationsmodul in Trac

Wie Abbildung 5.12 zeigt, kann die generische Ticketstruktur gut zur Erstellung angepasster Reports genutzt werden. Diese lassen sich sehr einfach durch Verwendung eines Makros²⁷ in beliebige Wiki-Seiten einbinden.

²⁴<http://trac-hacks.org/>, 23.03.2009

²⁵<http://trac.edgewall.org/wiki/TracPlugins#InstallingaTracPlugin>, 23.03.2009

²⁶http://wiki.eclipse.org/index.php/Mylyn_Trac_Connector, 23.03.2009

²⁷<http://trac.edgewall.org/wiki/TicketQuery>, 24.03.2009

The screenshot shows a 'Custom Query' interface with the following details:

- Filters:**
 - Milestone: is 0.11.5
 - Priority: is highest
 - or Priority: high
 - Status: assigned, closed, new, reopened
 - Add filter (button)
- Columns:**
 - Group results by (dropdown), descending checked
 - Show under each result: Description checkbox
 - Max items per page: 100
 - Update button
- Results Table:**

Ticket	Summary	Milestone	Priority	Status	Owner	Type
#2672	Trac is trying to diff binary content	0.11.5	high	new	rblank	defect
#6348	Catch database exceptions in a backend neutral way	0.11.5	high	new	cboos	defect
- Note:** See [TracQuery](#) for help on using queries.

Abbildung 5.12: Benutzerdefinierter Ticket-Report in Trac

Agilo erweitert auch hier die Fähigkeiten von Trac um Agilo- und Scrum-spezifische Konfigurationsmöglichkeiten. Abbildung 5.13 zeigt die Möglichkeit, Sprints zu konfigurieren.

The screenshot shows the 'Administration' module with the 'Sprints' tab selected. The 'Manage Sprints' section displays the following data:

Name / Milestone	Start	End	Duration (working days)
Sprint 1 milestone1	22.12.2008 09:00:00	16.01.2009 18:00:00	20
Sprint 2 milestone1	19.01.2009 09:00:00	13.02.2009 18:00:00	20
Sprint 4 Release 1.0	23.03.2009 09:00:00	10.04.2009 18:00:00	15
Sprint 5 Release 1.0	13.04.2009 09:00:00	01.05.2009 18:00:00	15

To the right, there is a form for adding a new sprint:

- Name:
- Milestone: Release 1.0
- Start date (DD.MM.YYYY):
- End date (DD.MM.YYYY):
- OR
- Duration (working days):
- Add button

Abbildung 5.13: Administrationsmodul in Agilo

Fazit

Die Möglichkeit der Konfiguration des Werkzeugs ist in Trac und Agilo mit einem integrierten Modul gegeben. Besonders die Erweiterbarkeit durch Plugins ist bei Trac und Agilo hervorzuheben. Durch eine offene Architektur und eine sehr aktive Community kann der Funktionsumfang umfangreich erweitert werden. XPlanner und Agilefant bieten keine vergleichbaren Möglichkeiten.

Tabelle 5.9: Bewertung Konfigurierbarkeit

XPlanner	2
Agilefant	1
Trac	7
Agilo	7

5.3.6 Bewertung Einfache Handhabung

Die folgenden Abschnitte beinhalten Anmerkungen zur Handhabung der Werkzeuge. Die Bewertung wird in Tabelle 5.10 zusammengefasst.

XPlanner verfügt über eine zweckmäßige Oberfläche, die sich an den Bedürfnissen von XP-Teams orientiert aber auch Scrum-Features (Burndown Chart) bietet. Das Werkzeug verfügt über gute Internationalisierung, die in den anderen Werkzeugen so nicht gegeben ist. Die Dokumentation des Systems beschränkt sich auf Frequently Asked Questions.

Agilefant ist speziell auf Scrum ausgerichtet und die Bedienung erfordert etwas Einarbeitung. Die Oberfläche wirkt dennoch kompakt und klar gegliedert. Positiv ist die Möglichkeit, mehrere Projekte parallel zu verwalten, was mit *Trac* und *Agilo* nicht möglich ist. Ebenso positiv ist die grafische Darstellung der Product Roadmap²⁸, in der die wesentlichen Elemente im zeitlichen Verlauf dargestellt sind. Negativ ist die ausschließliche Unterstützung von Firefox als Frontend. Eine Benutzerdokumentation im herkömmlichen Sinne ist nicht vorhanden, was die Einarbeitung deutlich erschwert.

Trac verfolgt einen generischen Ansatz, der sich prinzipiell auch für andere Projektarten wie z.B. Organisationsprojekte nutzen lässt. Für die Anpassung des Systems für agile Verfahren stehen eine Vielzahl von Plugins bereit. Trac verfügt über eine übersichtliche, einfach zu benutzende Oberfläche. Trac integriert eine umfangreiche Online-Hilfe in das Wiki des Systems, so dass Zugriffe auf systemspezifische Informationen direkt zur Verfügung stehen. Negativ ist die Beschränkung auf ein Projekt. Um weitere Projekte zu managen muss

²⁸Verwendung findet *Timeline*, ein Widget zur Visualisierung von Daten (<http://www.simile-widgets.org/timeline/>), 25.03.2009

eine weitere Umgebung aufgesetzt werden, was allerdings mit dem Programm `trac-admin` auf der Kommandozeile einfach möglich ist.

Agilo Die Spezialisierung von *Agilo* auf Scrum schränkt die Verwendungsmöglichkeiten für andere agile Verfahren etwas ein. Die Oberfläche wirkt aufgeräumt und klar gegliedert. Die Handhabung ist einfach und weitgehend intuitiv. Die von Trac gewohnte Online-Hilfe wird um Agilo- und Scrum-spezifische Punkte ergänzt. Obwohl die Online-Hilfe derzeit noch im Aufbau ist bietet sie bereits solide Basisinformationen, die eine Einarbeitung in kurzer Zeit möglich macht.

Fazit

Die Handhabung ist bei allen Werkzeugen vergleichsweise einfach. Die Basiskonzepte bei Trac und Agilo – Wiki und Ticket – sind schlicht und leicht zu adaptieren. Agilefant verfügt über ein klares Konzept der Bedienung. Die einfache Handhabung bei Trac leidet beim Einsatz von (vielen) Plugins.

Tabelle 5.10: Bewertung Einfache Handhabung

XPlanner	5
Agilefant	6
Trac	4
Agilo	7

5.3.7 Bewertung Aktivität

Die folgenden Abschnitte vermerken die Anzahl der Releases innerhalb des letzten Jahres. Die Bewertung wird in Tabelle 5.11 zusammengefasst.

XPlanner veröffentlichte die letzte *XPlanner*-Version – 0.7b7 – am 23.05.2006²⁹. Ob der aktive Entwicklungsprozess fortgesetzt wird ist offen, die letzte Version (0.7b7) auf Sourceforge datiert auf Mai 2006³⁰. Andererseits sind Code-Aktivitäten auf der Seite http://sourceforge.net/project/showfiles.php?group_id=49017&package_id=42226, 09.03.2009

²⁹http://sourceforge.net/project/showfiles.php?group_id=49017&package_id=42226, 09.03.2009

³⁰http://sourceforge.net/project/showfiles.php?group_id=49017, 09.03.2009

fisheye.codehaus.org/changelog/xplanner zu verzeichnen (November 2008).

Agilefant veröffentlicht in kurzen Abständen neue Versionen und Bugfixes³¹, im letzten Jahr 16 Releases.

Trac fasst die Versionsstände auf der Projektseite des Werkzeugs in einer Roadmap zusammen³². Die Anzahl der Releases lag im letzten Jahr bei 5.

Agilo bietet keine analoge Release-Übersicht, auf Nachfrage teilte einer der Entwickler per E-Mail einen etwa monatlichen Releasezyklus mit: „[...] normally we have a minor release every month, a release every quarter, and patch releases, after a release every 1 week or 2 weeks (release sprints).“ Das ergibt 12 Releases.

Fazit

Zu Agilefant erscheinen in kurzen Abständen Bugfixes und Releases. Ähnlich große Aktivität zeigen Agilo und Trac. Lediglich zu XPlanner ist kein aktuelles Release verfügbar.

Tabelle 5.11: Bewertung Aktivität

XPlanner	0
Agilefant	8
Trac	6
Agilo	7

5.3.8 Auswahl einer Open Source-Projektmanagementsoftware

In Tabelle 5.12 werden die Erfüllungsgrade der einzelnen Kriterien für die jeweiligen Open Source-Projektmanagementsoftware mit den Gewichtungsfaktoren aus Tabelle 5.3 multipliziert. Die Spaltensummen ergeben schließlich den Nutzwert der Software.

³¹<http://www.agilefant.org/wiki/display/AEF/Downloads>, 09.03.2009

³²<http://trac.edgewall.org/roadmap?show=all>, 09.03.2009

Zu beachten ist, dass die Ergebnisse keine absolute Aussage über den tatsächlichen Nutzwert für die praktische Projektarbeit zulassen. Da die Werte aus einem Vergleich resultieren, können die Ergebnisse nur relativ zueinander gesehen werden. Ebenso wurde die Einteilung der Skala der Zielerfüllungsfaktoren in Tabelle 5.5 nach freiem Ermessen vorgenommen. Die Auswahl und Bewertung der Kriterien wurde ebenfalls weitgehend subjektiv getroffen.

Insbesondere die Auswahl der Kriterien beeinflussen das Ergebnis stark, wie bei den Kriterien *Praktiken* und *Konfigurierbarkeit* deutlich zu sehen ist.

Auf die Gründe für die Auswahl und Gewichtung der Kriterien wurde bereits im Abschnitt 3.9 und Abschnitt 5.3.1 eingegangen. Mit diesen Vorgaben erweist sich Agilo als die Open Source-Projektmanagementsoftware mit dem höchsten Nutzwert.

Tabelle 5.12: Nutzwertermittlung

		XPlanner	Agilefant	Trac		Agilo	
1	Anforderungsmanagement	5	1,3	7	1,9	5	1,3
2	Release-/Iterationsplanung	3	0,5	7	1,2	5	0,8
3	Praktiken	1	0,1	1	0,1	6	0,4
4	Konfigurierbarkeit	2	0,5	1	0,2	7	1,6
5	Einfache Handhabung	5	0,8	6	1,0	4	0,7
6	Aktivität	0	0,0	8	0,8	6	0,6
	Nutzwert		3,2		5,1		5,5
							7,0

5.4 Zusammenfassung

Zur Unterstützung des agilen Entwicklungsprozesses stehen praxistaugliche Open Source-Lösungen zur Verfügung. Unter den Voraussetzungen nach denen die Evaluation durchgeführt wurde, ist *Agilo* das Werkzeug mit dem größten Nutzwert.

Dabei ist auch deutlich geworden, dass Scrum das Verfahren ist, welches am besten unterstützt wird. Unter diesem Gesichtspunkt ist auch *Agilefant* eine nützliche Alternative. Das ist insbesondere dann der Fall, wenn mehrere Produkte entwickelt werden und auf die Anpassung des Werkzeugs verzichtet werden kann.

Trac bietet mit seinem generischen Ansatz ausreichende Möglichkeiten agiles Projektmanagement auch auf andere Projektarten, wie beispielsweise Organisationsprojekte zu übertragen.

Damit kann auch die in Abschnitt 1.1 gestellte Frage nach einem geeigneten Softwaresystem beantwortet werden:

Antwort auf Frage 5 *Die Open Source-Projektmanagementsoftware mit dem größten Nutzwert zur Unterstützung agiler Verfahren – insbesondere Scrum – ist Agilo.*

Kapitel 6

Fazit und Ausblick

Wenn Informationsstrahler vorhanden sind, brauchen Passanten keine Fragen zu stellen; die Informationen treffen im Vorbeigehen auf sie.

(Alistair Cockburn)

Die vergangenen vier Jahrzehnte Softwareentwicklung haben eine unübersehbare Zahl von Verfahren, Praktiken und Methoden hervorgebracht. Viele dieser Prinzipien haben sich bewährt, einige gelten als überholt, werden aber teilweise dennoch mit ungebremster Vehemenz angewendet.

Mit der vorliegenden Arbeit wurde versucht, diese Entwicklung zu reflektieren, wobei der Schwerpunkt auf ausgewählten agilen Verfahren und deren Unterstützung durch Open Source-Projektmanagementsoftware liegt. In diesem abschließenden Kapitel werden die Erkenntnisse zusammengefasst und versucht, einen Blick auf die künftige Entwicklung zu werfen.

Agile Verfahren nehmen keine Außenseiterposition (mehr) ein und sind genau wie die klassischen Verfahren etabliert. Die Erfolgsquoten agil durchgeföhrter Projekte sind in den vorliegenden Studien höher als bei Anwendung klassischer Verfahren. Dabei muss berücksichtigt werden, dass diese Ergebnisse durch Internet-Befragungen zustande gekommen sind. Insofern ist die Repräsentativität problematisch.

Ein Projekterfolg stellt sich auch bei agilen Verfahren nicht automatisch ein. Je nach konkreter Projektsituation ist ein agiles Verfahren mehr oder weniger gut geeignet. Für Projekte mit einem feststehenden Termin und Funktionsumfang, Klarheit über die Anforderungen und geringen Risiken und Unsicherheit sind agile Verfahren nicht geeignet. Bei Projekten, die eine hohes Risiko an Anforderungsänderung,

große Unsicherheiten und einen verhandelbaren Funktionsumfang haben sind agile Verfahren Erfolg versprechend. Bezuglich der Größe erfolgreicher Teams – unabhängig vom Aspekt Agilität – gelten 7 Mitglieder als Maximalgröße.

Der größte Unterschied zu klassischen Verfahren liegt in der Konzentration auf den Faktor Mensch. In agilen Verfahren kommt der direkten Kommunikation, Kooperation und Zufriedenheit der Teammitglieder und sonstiger Stakeholder eine wesentlich stärkere Bedeutung zu. Der Erfolg eines agil durchgeführten Projekts ist im wesentlichen von der Akzeptanz dieser Grundsätze abhängig. Das gilt gleichermaßen für das Team wie für das Management.

Trotz der Erfolgsaussichten, die agile Verfahren haben, ist Skepsis geboten. Insbesondere Extreme Programming (XP) ist kritisch zu betrachten. Wie in Abschnitt 3.6.1 ausgeführt, birgt XP durch sein „Alles-oder-Nichts-Prinzip“ ein nicht zu vernachlässigendes Risiko. Dabei muss betont werden, dass die XP-Praktiken auch ohne die strenge Kopplung mit XP nutzbringend eingesetzt werden können.

Scrum hingegen bietet einen Ansatz, der selbst agil genug ist, um an die spezifischen Projektbedingungen angepasst werden zu können. Scrum lässt dabei ausreichend Freiheit zu Ausgestaltung im konkreten Projekt. Für die Softwareentwicklung ist die Kombination mit Praktiken (Testing, Continuous Integration, Coding Standards, u.a.) als Erfolgsfaktor zu werten. Scrum ist das in Deutschland am häufigsten eingesetzte Verfahren.

Die Unterstützung von agilen Verfahren durch Open Source-Projektmanagementsoftware ist prinzipiell möglich und unter bestimmten Voraussetzungen notwendig. Die Notwendigkeit ergibt sich bei verteil arbeitenden Teams und einem zusätzlichen Nutzwert, den die Verwendung einer Projektmanagementsoftware mit sich bringt. Von großem Nutzwert können beispielsweise die Speicherung und Nachvollziehbarkeit von Anforderungen oder erweiterte Auswertungsmöglichkeiten sein.

Die Evaluation ausgewählter Werkzeuge mit Hilfe einer Nutzwertanalyse hat den höchsten Nutzwert für die Open Source-Projektmanagementsoftware *Agilo* ergeben.

Ein wesentlicher Schwerpunkt agiler Verfahren liegt auf der kreativen Entfaltung von selbstorganisierenden Teams. Das scheint nur schwer mit typischem Ingenieurs-Denken vereinbar, was die zum Teil heftig geführten Debatten Pro und Contra agiler Verfahren zeigen. Für weitere Arbeiten wäre deshalb eine vertiefende Untersuchung zu diesem

Thema wünschenswert. Dabei sind insbesondere die sozialen Fähigkeiten von typischen Scrum-Rollen, wie Product Owner und Scrum-Master von großem Interesse.

Danksagung

Mein Dank gilt den Betreuern der Diplomarbeit, Prof. Dr. Herbert Neunteufel und Prof. Dr. Jürgen Cleve.

Meiner Partnerin Andrea danke ich für orthografische Richtigstellungen und dafür, dass ich mich ganz meiner Arbeit widmen konnte und Carl, dass er es respektiert hat, in seinem Zimmer Cello zu üben.

Für Korrekturlesen danke ich Heike Wünsch und Tino Morgenstern für die CD-Erstellung.

Ganz besonders dankbar bin ich den Hinweisen und Ratschlägen von Dipl.-Ing. Falk Atmanspacher jeden Dienstagabend in und nach der Sauna sowie für sein kompetentes Review der Arbeit.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Chemnitz, 17. April 2009

Unterschrift

Abbildungsverzeichnis

1.1	Chaos-Reports 1994-2004	2
1.2	Allgemeine Darstellung Vorgehensmodell	8
3.1	Rational Unified Process Prozessstruktur [PEF ⁺ 07]	18
3.2	V-Modell XT	21
3.3	XP Praktiken	35
3.4	Prinzipielle Darstellung Scrum [Sof08]	41
3.5	Burndown-Diagramm	44
3.6	APM-Iterationswolkenmetapher [Oos08a]	49
3.7	APM-Planungsebenen [Oos08a]	50
3.8	Änderungskosten nach Boehm	54
3.9	Änderungskosten nach Beck	55
3.10	Änderungskostenverlagerung durch XP in die Projektstartphase	56
3.11	Anwendung agiler Techniken auf die Änderungskostenkurve [Amb08b]	57
3.12	XP@Scrum [Cha]	60
3.13	Fallender Erfolg bei steigender Komplexität mit klassischen Verfahren [Sut98]	68
3.14	Wachsender Erfolg bei steigender Komplexität mit flexiblen (agilen) Verfahren [Sut98]	69
3.15	Erfolgsquoten von Softwareentwicklungsprojekten [Amb07] .	70
3.16	Erfolgsfaktor Vorgehensweise [Oos09]	70
3.17	Produktivität in Abhängigkeit von der Teamgröße [Sut98b]	73
4.1	Projektmanager?	78
5.1	XPlanner Story-Task-Beziehung [Glo08]	101
5.2	Backlog Item in Agilefant	101
5.3	Ticket in Trac	102
5.4	User Story in Agilo	103
5.5	XPlanner Projekt-Iterations-Beziehung [Glo08]	104
5.6	Agilefant Produkt-Projekt-Iterations-Beziehung	104
5.7	Meilenstein in Trac	105
5.8	Roadmap in Agilo	106
5.9	Build Status in Trac	108
5.10	Versionskontrolle in Trac	109

5.11 Administrationsmodul in Trac	110
5.12 Benutzerdefinierter Ticket-Report in Trac	111
5.13 Administrationsmodul in Agilo	111
A.1 Historische Entwicklung objektorientierter Methoden und der UML [Oes06]	xxvii
A.2 APM-Iteration i (Timebox)	xxviii
A.3 Effektivität agiler Verfahren im Vergleich mit traditionellen Verfahren [Amb08a]	xxix
A.4 XPlanner Statistik	xxx
A.5 Agilefant Sprint	xxxi
A.6 Trac - Scrum Burndown Plugin	xxxii
A.7 Agilo - Sprint Backlog und Burndown Chart	xxxii
A.8 Versionsvergleich in Trac	xxxiii

Tabellenverzeichnis

3.1	Einsatz von XP-Praktiken mit Scrum	61
5.1	Relevanz von Open Source-Projektmanagementsoftware . . .	96
5.2	Auswahl der Bewertungskriterien	97
5.3	Gewichtungsfaktoren ermitteln	99
5.4	Legende Ermittlung Gewichtungsfaktoren	99
5.5	Skala Zielerfüllungsfaktoren	100
5.6	Bewertung Anforderungsmanagement	103
5.7	Bewertung Release-/Iterationsplanung	106
5.8	Bewertung Praktiken	109
5.9	Bewertung Konfigurierbarkeit	112
5.10	Bewertung Einfache Handhabung	113
5.11	Bewertung Aktivität	114
5.12	Nutzwertermittlung	115

Glossar

AJAX	AJAX ist die Abkürzung für „Asynchronous JavaScript And XML“. Mit diesem Konzept wird die asynchrone Datenübertragung zwischen Server und Browser realisiert, so dass für den Benutzer der Eindruck entsteht, mit einer nativen Anwendung zu arbeiten.
Backlog	Verzeichnis von User Stories, Anforderungen oder Aufgaben. In Form von Product Backlog und Sprint Backlog Bestandteil des agilen Verfahrens Scrum (Vgl. Seite 38).
Best Practices	Erfolgsrezept, auch Erfolgsmethode genannt, stammt aus der angloamerikanischen Betriebswirtschaft. Eine Sammlung von bewährten und kostengünstigen Verfahren und Prozessen.
Burn Down Chart	Ein Diagramm, das den verbleibenden Aufwand in Beziehung zur Zeit darstellt. Burn Down Charts werden meist in Scrum eingesetzt.
Feature	Die Funktionalität einer Software. Die Fähigkeit, eine bestimmte Aufgabe oder Menge von Aufgaben zu lösen.
Führungsspanne	Anzahl der Mitarbeiter, die eine Führungskraft führen kann.
Index Card	Etwa DIN A6 große Pappkärtchen, auf denen projektbezogene Notizen vermerkt werden. Index Cards kommen häufig in agilen Verfahren zum Einsatz und werden auf Pinwänden systematisch geordnet.

Iteration	Bezeichnet einen einzelnen Entwicklungszyklus, bei dem – je nach Verfahren – bestimmte Phasen durchlaufen werden.
Meilenstein	Eine veränderliche Zeitspanne mit feststehendem Umfang von z.B. Aufgaben. Ein definierter Termin ist als Planungsgröße aufzufassen und kann bei Bedarf verschoben werden. Das Gegenstück ist die <i>Timelinebox</i> .
Nutzwertanalyse	Analyse einer Menge komplexer Handlungsalternativen mit dem Zweck, die Elemente dieser Menge entsprechend den Präferenzen des Entscheidungsträgers bezüglich eines multidimensionalen Zielsystems zu ordnen. Die Abbildung der Ordnung erfolgt durch die Angabe der Nutzwerte (Gesamtwerte) der Alternativen. [Zag76]
Planning Poker	Eine konsens-basierte Methode zur Aufwandsschätzung in agilen Verfahren. Planning Poker ist eine Variante der Schätzmethode „Wideband Delphi“.
Release	Fertiggestellte und veröffentlichte Version eines Softwareprodukts.
Roadmap	Die Roadmap gibt in der Softwareentwicklung die Zeitpunkte (Meilensteine) an, zu denen eine bestimmte Funktionalität fertiggestellt sein soll.
RSS-Feed	Ein RSS-Feed ist ein Benachrichtigungsservice. Nachdem ein RSS-Feed abonniert wurde liefert er regelmäßig Informationen. Dafür ist ein Client – ein sog. RSS-Newsreader erforderlich.
Screencast	Digitaler Film, der die Abläufe der Verwendung einer Software zeigt. Häufig wird die Darstellung durch textuelle oder gesprochene Kommentare begleitet.

Stakeholder	Person, die ein Interesse am Projekt hat. Dazu zählt das Management, Geldgeber und Benutzer.
Task Board	Ein Mittel, um den täglichen Zyklus eines Sprints zu visualisieren. Aktivitäten werden in den Zuständen <i>offen</i> , <i>in Bearbeitung</i> und <i>erledigt</i> in übersichtlicher Weise dargestellt. Die einfachste Form ist die einer Tafel, auf der die Aktivitäten mit Karten angebracht werden. Es existieren eine Reihe von Softwarelösungen, die ein Task Board implementieren.
Timebox	Eine feststehende Zeitspanne, die nicht überschritten werden darf. Der Umfang innerhalb dieser Zeitspanne ist veränderlich (z.B. die Anzahl der Aufgaben eines Sprints in Scrum). Wird eine Timebox angewendet, spricht man von <i>Time Boxing</i> . Anwendung findet Time Boxing u.a. bei Scrum (siehe Abschnitt 3.4). Das Gegenstück ist der <i>Meilenstein</i> .
Versionskontrolle	Mit einem Versionskontrollsystem werden Änderungen am Quellcode in einem Archiv gespeichert. Somit ist eine Versionierung der des Quellcodes möglich. Wichtige Vertreter sind CVS, Git, Subversion (Open-Source) und ClearCase (Closed-Source).
Wiki	Ein Wiki ist ein Online-System, das von den Benutzern gelesen und mit einer einfachen Syntax bearbeitet werden kann. Verknüpfungen zwischen Wiki-Seiten werden durch Hyperlinks hergestellt. Wikis werden meist in der verteilten Zusammenarbeit von Teams oder Nutzergruppen verwendet.

Literaturverzeichnis

- [A+98] ANDERSON, Ann u.a.: *Chrysler Goes to “Extremes”*. <http://www.xprogramming.com/publications/dc9810cs.pdf>, Oktober 1998
- [Abe01] ABENDROTH, Dieter: *FuE-Förderung von Informatiksystemen*. http://mapkit.informatik.uni-essen.de/oeffentl/Dokumente/AllgBerichte/AbschlussFolien/BMBF_FuE_Foerderung_2001.ppt, 2001
- [ADM96] ADVANCED DEVELOPMENT METHODS, Inc: *Controlled Chaos : Living on the Edge*. <http://www.controlchaos.com/download/LivingontheEdge.pdf>, 1996
- [ADM08] ADVANCED DEVELOPMENT METHODS, Inc: *What is Scrum?* <http://www.controlchaos.com/about/>, 2008
- [ADMkA] ADVANCED DEVELOPMENT METHODS, Inc: *Controlled-Chaos Software Development*. <http://www.controlchaos.com/download/Controlled-ChaosSoftwareDevelopment.pdf>, k.A.
- [All06] ALLIANCE, Agile: *What Is Agile Software Development?* <http://agilealliance.org/show/2>, May 2006
- [Amb02] AMBLER, Scott W.: *Agile Modelling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, 2002
- [Amb07] AMBLER, Scott W.: *IT Project Success Rates Survey: August 2007*. <http://www.ambysoft.com/surveys/success2007.html>, August 2007
- [Amb08a] AMBLER, Scott W.: *Agile Adoption Survey 2008*. <http://www.ambysoft.com/surveys/agileFebruary2008.html>, Februar 2008
- [Amb08b] AMBLER, Scott W.: *Answering the "Where is the Proof That Agile Methods Work" Question*. <http://www.agilemodeling.com/essays/proof.htm>, 2008
- [Bal97] BALZERT, Helmut: *Lehrbuch der Software-Technik. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Heidelberg : Spektrum, Akad. Verl., 1997
- [Bec03] BECK, Kent: *Extreme Programming, Das Manifest*. Addison-Wesley, 2003

-
- [Bee] BEEDLE, Mike: *Enterprise Agile Process.* <http://www.e-architects.com/AE/practices.html>, zuletzt besucht: 11.01.2009
 - [Boe81] BOEHM, Barry: *Software Engineering Economics.* Prentice-Hall, 1981
 - [Boe88] BOEHM, Barry W.: A Spiral Model of Software Development and Enhancement. In: *Computer* 21 (1988), May, Nr. 5, 61–72. <http://dx.doi.org/http://dx.doi.org/10.1109/2.59>. – DOI <http://dx.doi.org/10.1109/2.59>. – ISSN 0018-9162
 - [Boe06] BOEHM, Barry: A View of 20th and 21st Century Software Engineering / University of Southern California University Park Campus, Los Angeles. <http://csse.usc.edu/csse/TECHRPTS/2006/usccsse2006-626/usccsse2006-626.pdf>, 2006. – Forschungsbericht
 - [Boo00] BOOTH, Rose: IT Project Failures Costly, TechRepublic/Gartner Study Finds. (2000), 11. http://articles.techrepublic.com.com/5100-10878_11-1062043.html, Abruf: 20.11.2008
 - [Bro87] BROOKS, Frederick P.: No Silver Bullet: Essence and Accidents of Software Engineering. In: *Computer* 20 (1987), S. 10–19
 - [Bro03] BROOKS, Frederick P.: *The Mythical Man-Month.* Addison-Wesley, 2003
 - [Cha] CHAOS, Control: *XP @ Scrum.* <http://www.controlchaos.com/about/xp.php>, zuletzt besucht: 11.01.2009
 - [Coc03] COCKBURN, Alistair: *Agile Software-Entwicklung.* Mitp-Verlag, 2003
 - [Coh04] COHN, Mike: *Advantages Of User Stories For Requirements Advantages Of User Stories For Requirements.* <http://www.mountaingoatsoftware.com/article/27-advantages-of-user-stories-for-requirements>, Oktober 2004
 - [Coh07a] COHN, Mike: *Advice on Conducting the Scrum of Scrums Meeting.* <http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting>, Mai 2007
 - [Coh07b] COHN, Mike: *Differences Between Scrum and Extreme Programming.* <http://blog.mountaingoatsoftware.com/?p=8>, April 2007
 - [Coh08] COHN, Mike: *Agile Estimation and Planning.* 7. Auflage. Pearson Education Inc., 2008
 - [D⁺07] DEMARCO, Tom u. a.: *Adrenalin Junkies Formular Zombies.* Hanser, 2007
 - [DeM98] DEMARCO, Tom: *Der Termin. Ein Roman über Projektmanagement.* Hanser, 1998

-
- [DeM09] DEMARCO, Tom: The other three pillars of IT Performance. http://www.sigs.de/kongresse/oop_2009/OOP_2009_ConferenceProgram.pdf, 2009
- [Den08] DENERT, Ernst: Software-Engineering als Mittel zum Zweck. In: *OBJEKTSPEKTRUM* (2008), November/Dezember, Nr. 6, S. 28–33
- [DS08] DUBAKOV, Michael ; STEVENS, Peter: *Agile Tools. The Good, the Bad and the Ugly.* <http://www.targetprocess.com/LearnAgile/Whitepapers/AgileTools.aspx>, 2008
- [Eck09] ECKSTEIN, Jutta: Vom Sinn und Unsinn unternehmensweiter Vorgehensmodelle. In: *OBJEKTSPEKTRUM* (2009), Februar, Nr. 1, S. 18–21
- [EG72] ELDREDGE, Niles ; GOULD, Stephen J. ; T.SCHOPF (Hrsg.): *Models in Paleobiology.* <http://www.blackwellpublishing.com/ridley/classictexts/eldredge.pdf> : Freeman, Cooper and Co., 1972 (82-115)
- [F⁺05] FILSS, Christop u. a.: Rahmen zur Auswahl von Vorgehensmodellen / GI Fachgruppe WI-VM Arbeitskreis "Vorgehensmodelltypen". http://www.faw.unilinz.ac.at/PublicationFullText/2005vm/ms_ArbeitsberichtWI-VM05.pdf, März 2005. – Forschungsbericht
- [Fra04] FRANZ, Stephan: *Grundlagen des ökonomischen Ansatzes: Das Erklärungskonzept des Homo Oeconomicus.* 2004
- [Glo08] GLOGER, Boris: *Scrum Tools — XPlanner — Review.* <http://scrum4you.wordpress.com/2008/08/06/scrum-tools-xplanner-review/>, August 2008
- [Göt06] GÖTTSCHE, Michael: Guter Teamgeist. In: *Linux-Magazin* (2006), April, Nr. 4, S. 130–134
- [Gra04] GRASSMUCK, Volker: *Freie Software.* 2. Auflage. Bundeszentrale für politische Bildung (bpb), 2004 (Band 458)
- [HL07] HTWK-LEIPZIG: *Chaos-Report.* <http://www.imn.htwk-leipzig.de/~weicker/pmwiki/pmwiki.php>Main\Chaos-Report>, Januar 2007
- [HOT⁺08] HEULER06 ; ODI ; THORNARD u. a.: *Open- Source im Unternehmen: Betriebswirtschaftliche Betrachtung von Open-Source-Software.* http://de.wikibooks.org/wiki/Open-Source_im_Unternehmen:_Betriebswirtschaftliche_Betrachtung_von_Open-Source-Software, August 2008
- [Huh07] HUHMANN, Jochem: Spuren im Wald. In: *iX* (2007), April, Nr. 4, S. 78–80
- [Hüt08] HÜTTERMANN, Michael: *Agile Java-Entwicklung in der Praxis.* 1. Auflage. O'Reilly Verlag, 2008

-
- [Hut09] HUTH, Stefan: Probleme und Fehler im Requirements-Engineering: Ergebnisse einer aktuellen Studie. In: *OBJEKTspektrum* (2009), Februar, Nr. 1, S. 11–13
 - [IBM04] IBM: *RUP for Extreme Programming (XP) Plug-Ins.* <http://www.ibm.com/developerworks/rational/library/4156.html>, April 2004
 - [JBR98] JACOBSON, Ivar ; BOOCH, Grady ; RUMBAUGH, James: *The Unified Software Development Process*. Addison Wesley Longman, 1998
 - [Jef00] JEFFRIES, Ron: *Extreme Programming Installed*. Addison-Wesley, 2000
 - [Jen06] JENKINS, Nick: *A Project Management Primer*. <http://www.nickjenkins.net/prose/projectPrimer.pdf>, 2006
 - [Kru99] KRUCHTEN, Philippe: *Der rational unified process : Eine Einführung*. Addison-Wesley-Longman, 1999
 - [Lit07] LITKE, Hans-Dieter: *Projektmanagement: Methoden, Techniken, Verhaltensweisen*. 5. Auflage. Hanser, 2007
 - [LW00] LEFFINGWELL, Dean ; WIDRIG, Don: *Management software requirements: a unified approach*. Addison-Wesley, 2000
 - [McB02] MCBREEN, Pete: *Questioning Extreme Programming*. Addison-Wesley, 2002
 - [Mel02] MELOCHE, Thomas: *The Rational Unified Process, A Well Documented, Complete yet Complex Methodology*. 2002
 - [Mül08] MÜLLER, Frank: Zur Sache, Schätzchen. In: *iX* 9 (2008), September, S. 76–79
 - [Nar07] NARASIMHAN, A.: *Tipping the Scale: What to Know Before You Add a Second Level of Scrum*. <http://www.scrumalliance.org/articles/81-tipping-the-scale>, Dezember 2007
 - [Nik02] NIKLAS, Cornelia: Mehr Entscheidungssicherheit mit der Nutzwertanalyse. In: *Projekt Magazin* (2002), Februar, Nr. 23
 - [NR05] NIEBUHR, Dirk ; RAUSCH, J. Prof. Dr. A.: Neuer Glanz, Erfolgreiche Projekte mit dem V-Modell XT. In: *iX* (2005), Juni, Nr. 6, S. 106–108
 - [OBJ09] OBJEKTSPEKTRUM: Open-Source kein Hype. In: *OBJEKTspektrum* (2009), Februar, Nr. 2, S. 6
 - [Oes98] OESTEREICH, Bernd: *Objektorientierte Softwareentwicklung - Analyse und Design mit der UML*. 4., aktualisierte Auflage. Oldenburg Verlag, 1998
 - [Oes06] OESTEREICH, Bernd: *Analyse und Design mit UML 2.1*. 6., aktualisierte Auflage. Oldenbourg Verlag, 2006
 - [Oes09] OESTEREICH, Bernd: Der Weg ist das Ziel. In: *OBJEKTspektrum* (2009), Februar, Nr. 1, S. 14–17

-
- [Oos08a] OOSE INNOVATIVE INFORMATIK GMBH: *APM - Agiles Projektmanagement - Download.* <http://www.oose.de/apm/download>, 2008
- [Oos08b] OOSE INNOVATIVE INFORMATIK GMBH: Erfolgsfaktor Mensch - IT Expertise. (2008). http://oose.de/Soft_Skills.htm, Abruf: 22.11.2008
- [Oos09] OOSE INNOVATIVE INFORMATIK GMBH: *OOSE. PM-Studie 2009.* Februar 2009
- [Ost05] OSTER, Sebastian: *Rational Unified Process, Seminar Software Evolution.* 2005
- [OW08] OESTERREICH, Bernd ; WEISS, Christian: *APM - Agiles Projektmanagement.* 1. Auflage. dpunkt.verlag, 2008
- [Pau08] PAULISCH, Frances: Wissen ist Macht. In: *OBJEKTspektrum* (2008), Mai/Juni, Nr. 3, S. 3
- [PEF⁺07] PÉRAIRE, Cécile ; EDWARDS, Mike ; FERNANDES, Anangelo ; MANCIN, Enrico ; CARROLL, Kathy: *The IBM Rational Unified Process for System z.* 1. Ausgabe. <http://www.redbooks.ibm.com/redbooks/pdfs/sg247362.pdf> : IBM, 2007
- [Pic08] PICHLER, Roman: Erfolgsfaktor „Product Owner“. In: *OBJEKTspektrum* (2008), Mai/Juni, Nr. 3, S. 14–15
- [Rau06] RAUSCH, J. Prof. Dr. A.: IT Projekte erfolgreich... mit dem neuen V-Modell XT / Technische Universität Kaiserslautern, Fachbereich Informatik, AG Softwarearchitektur. <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Infomaterial/Einfuehrungsvortrag/VM-Einfuehrungsvortrag.ppt>, 2006. – Forschungsbericht
- [Ray99] RAYMOND, Eric S.: *Die Kathedrale und der Basar.* <http://gnuwin.epfl.ch/articles/de/Kathedrale/>, August 1999
- [Roy70] ROYCE, Winston: Managing the Development of Large Software Systems. In: *Technical Papers of Western Electronic Show and Convention (WesCon)* (1970), August
- [Sah08] SAHLING, Carsten: ”DSDM ATERN”: Die (bislang nur) Britisch strukturierte Agilität. In: *OBJEKTspektrum* (2008), Mai/Juni, Nr. 3, S. 29–33
- [SB01] SCHWABER, Ken ; BEEDLE, Mike: *Agile Software Development with Scrum.* Prentice Hall, 2001
- [Sch07a] SCHELLE, Heinz: *Projekte zum Erfolg führen. Projektmanagement systematisch und kompakt.* 5. Auflage. dtv, 2007
- [Sch07b] SCHWABER, Ken: *Agiles Projektmanagement mit Scrum.* Microsoft Press, 2007
- [Sch08] SCHIEL, Jim: *Implementing ISO9001 with Scrum.* <http://www.scrumalliance.org/resources/440>, Oktober 2008

-
- [SH05] STAHLKNECHT, Peter ; HASENKAMP, Ulrich: *Einführung in die Wirtschaftsinformatik*. 11. Auflage. Springer-Verlag, 2005
 - [Sof08] SOFTWARE, Mountain G.: *The Scrum Development Process*. <http://mountaingoatsoftware.com/scrum>, 2008
 - [SR03] STEPHENS, Matt ; ROSENBERG, Doug: *Extreme Programming Refactored: The Case Against XP*. Apress, 2003
 - [Sta95] STANDISH GROUP: *The Standish Group Report*. <http://net.educause.edu/ir/library/pdf/NCP08083B.pdf>, 1995
 - [Sta02] STALLMAN, Richard: *Free Software Free Society*. GNU Press, 2002
 - [Sta08] STAL, Michael: Bubbles don't crash. In: *iX* 11 (2008), November, S. 118–121
 - [Str03] STROHMEIER, Helmut: Was ist eigentlich Projekterfolg? In: *GPM-Magazin PMaktuell* 3 (2003), S. 29–32
 - [Sut98a] SUTHERLAND, Jeff: *SCRUM Lowers Risk*. <http://jeffsutherland.com/objwld98/proscrum.html>, 1995-1998
 - [Sut98b] SUTHERLAND, Jeff: *Team Size: Development Productivity Index*. http://jeffsutherland.com/objwld98/ow_scrum.html, 1995-1998
 - [Sut06] SUTHERLAND, Jeff: *Scrum Tuning: Lessons learned from Scrum implementation at Google*. <http://video.google.com/videoplay?docid=8795214308797356840>, Dezember 2006
 - [Sut07] SUTHERLAND, Jeff: *Scrum Burndown using Trac*. <http://jeffsutherland.com/scrum/2007/03/scrum-burndown-using-trac.html>, März 2007
 - [Sut98] SUTHERLAND, Jeff: *Risk with Current Methodologies*. <http://jeffsutherland.com/objwld98/probab.html>, 1995-98
 - [Tar09] TARGETPROCESS: *Agile Project Management Tools*. <http://www.agile-tools.net/>, 2009
 - [TN86] TAKEUCHI, H. ; NONAKA, I.: The new new product development game. In: *Harvard Business Review* (1986)
 - [TOS08] TAKEUCHI, Hirotaka ; OSONO, Emi ; SHIMIZU, Norihiko: The Contradictions That Drive Toyota's Success. In: *Harward Business Online* (2008), Juni
 - [Ver08] VERSIONONE, LLC: *Agile Tool Evaluator Guide 2008*. <http://www.versionone.com/pdf/AgileToolEvaluator.pdf>, 2008
 - [VW06] VERSTEEGEN, Gerhard ; WIEBEL, Rupert: Wunschlos glücklich; Im Vergleich: Werkzeuge für Anforderungsmanagement. In: *iX* (2006), Juni, Nr. 6, S. 88–92
 - [Web04] WEBER, Steven: *The Success of Open Source*. Harvard University Press, 2004. – ISBN 0674012925

- [WR08] WOLF, Henning ; ROOCK, Arne: Agilität wird Mainstream: Ergebnisse der Online-Umfrage 2008. In: *OBJEKTSPEKTRUM* (2008), Mai/Juni, Nr. 3, S. 11–13
- [WW01] WITT, Jürgen ; WITT, Thomas: *Der kontinuierliche Verbesserungsprozess : (KVP) ; Konzept - System - Maßnahmen.* Heidelberg : Sauer, 2001
- [Zag76] ZAGEMEISTER, Christof: *Nutzwertanalyse in der Systemtechnik.* 4. Auflage. Verlagskommission Wittemannsche Buchhandlung, 1976

Anhang A

Abbildungen

Entwicklung objektorientierter Methoden

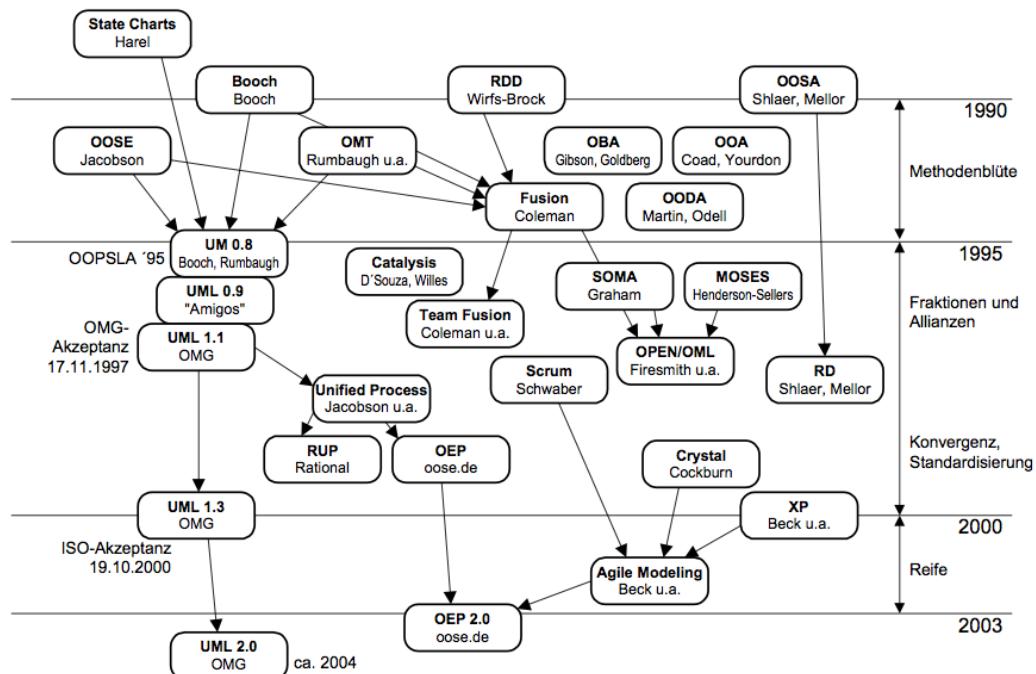


Abbildung A.1: Historische Entwicklung objektorientierter Methoden und der UML [Oes06]

APM-Iteration

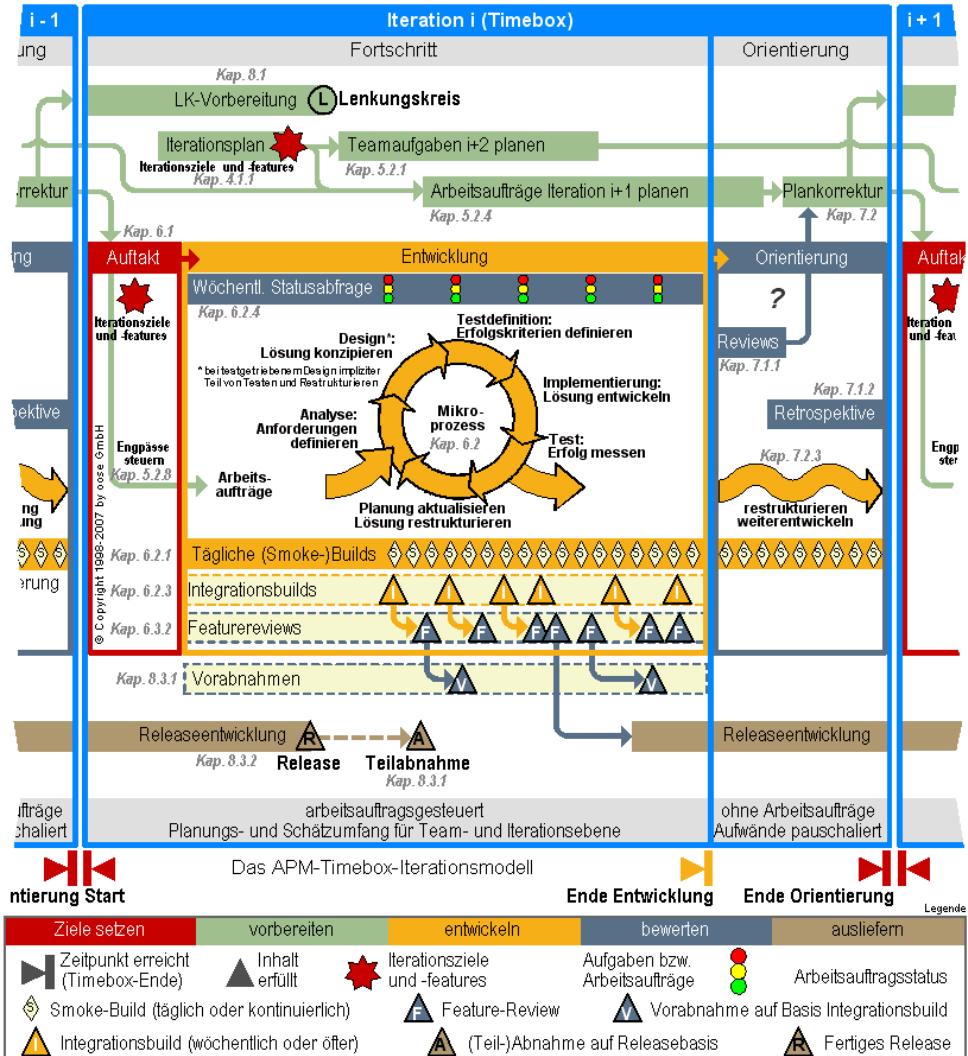
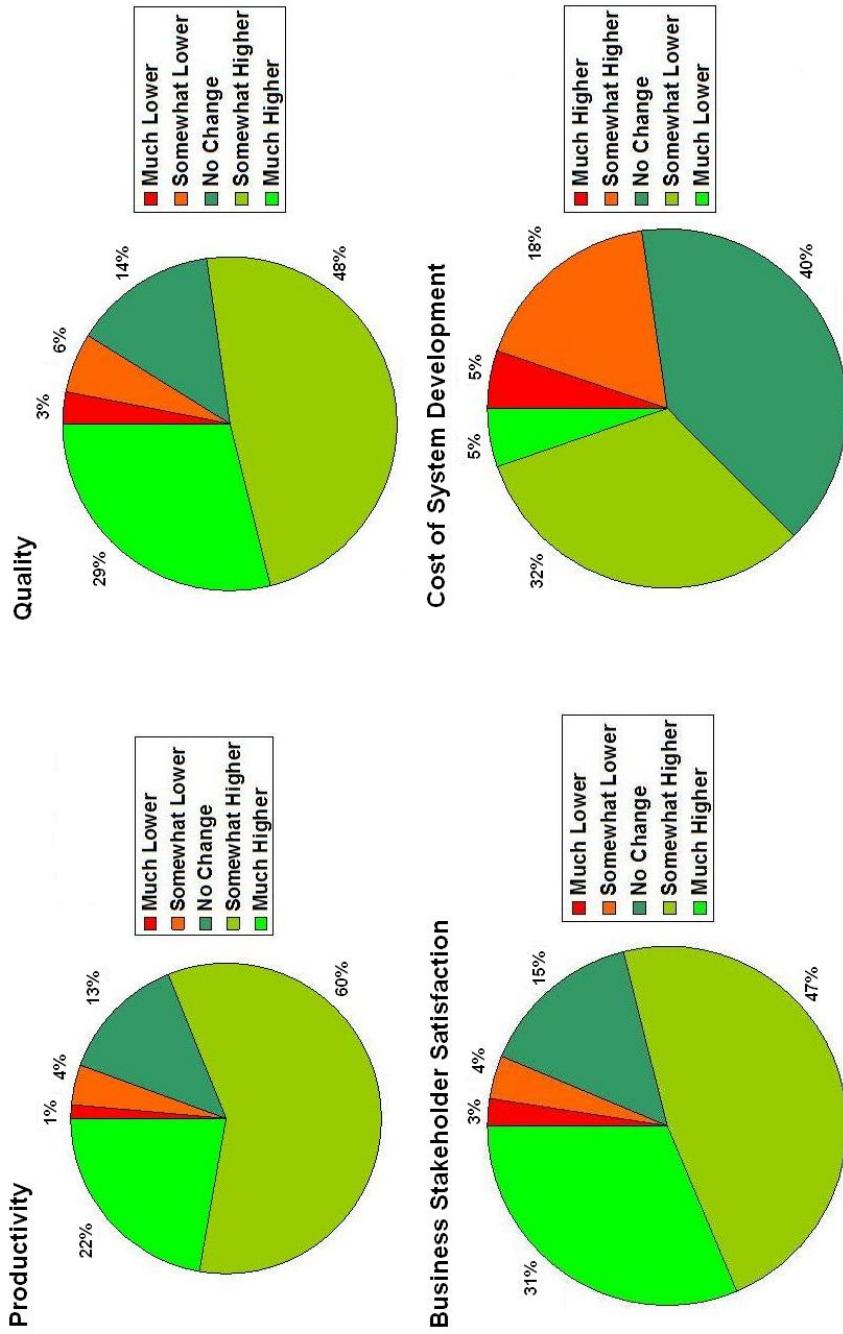


Abbildung A.2: APM-Iteration i (Timebox)

Agile Works



Copyright 2008 Scott W. Ambler

Source: Dr. Dobb's Journal 2008 Agile Adoption Survey

Abbildung A.3: Effektivität agiler Verfahren im Vergleich mit traditionellen Verfahren [Amb08a]

Open Source-Projektmanagementsoftware

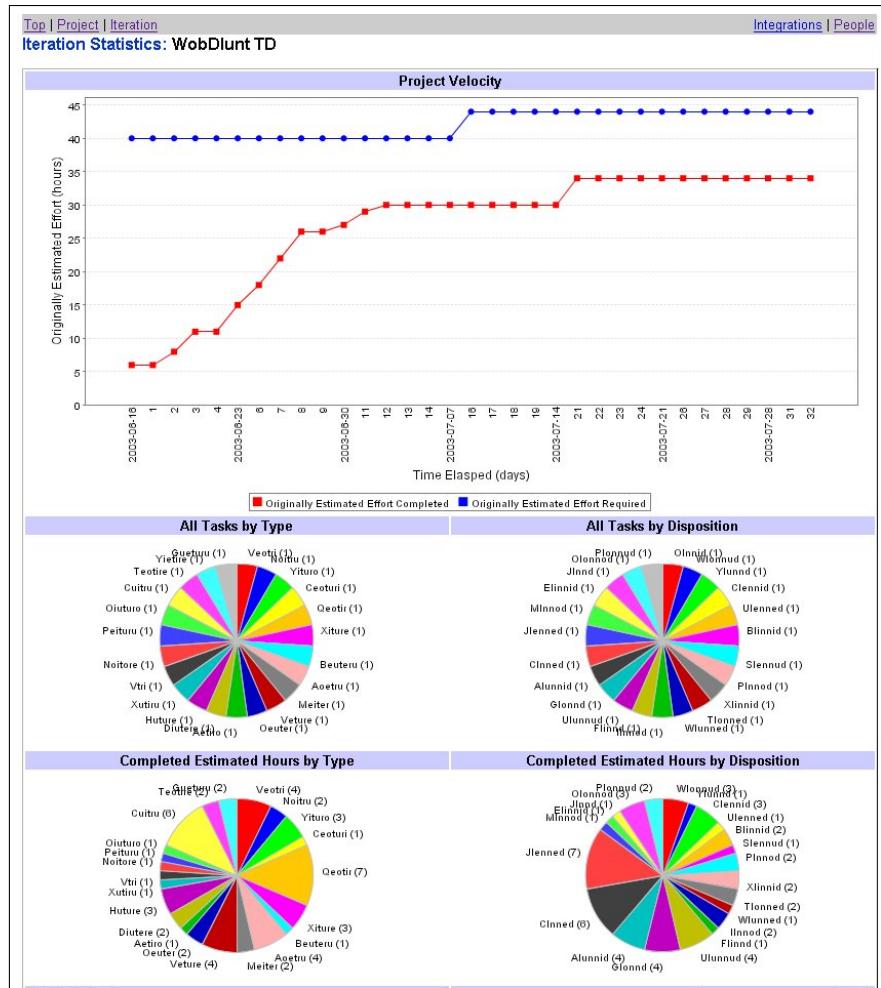


Abbildung A.4: XPlanner Statistik

> Backend Server > Release 2.0 > Sprint 1

Sprint 1

Details

Reference ID	BL:12	
Planned iteration size	-	
Timeframe	25.5.2008 - 24.6.2008	
		Velocity 0h / day Done 25% (1 / 4)

Themes Attach theme »

Name	Planned spending	Progress	Actions
Algo 2	20h	0% (0 / 1)	

Iteration goals

Backlog items

Name	Iteration Goal	Responsibles	Priority	Progress	Effort Left	Original Effort	Effort Spent	Actions
BugStory5	demo	undefined	1 / 2 TODOs done		1h	2h	2h	
Algo 2 Story1	jt	undefined	Started		2h	5h	—	
Story3	AP, dc	++++	0 / 1 TODOs done		10h	10h	2h	
Story2		undefined	Done		0h	—	—	
					13h	17h + 1 non-est.	4h BLI	

Abbildung A.5: Agilefant Sprint

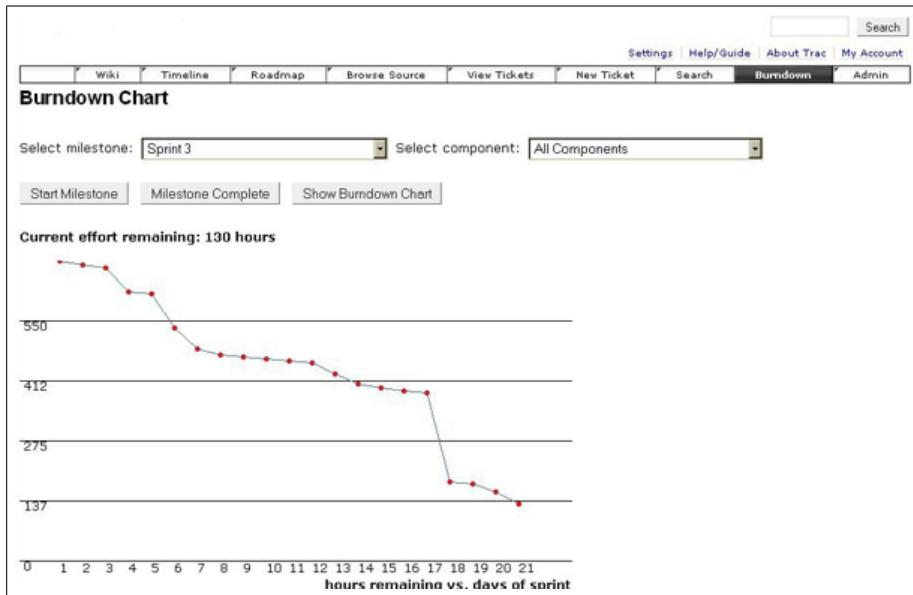


Abbildung A.6: Trac - Scrum Burndown Plugin

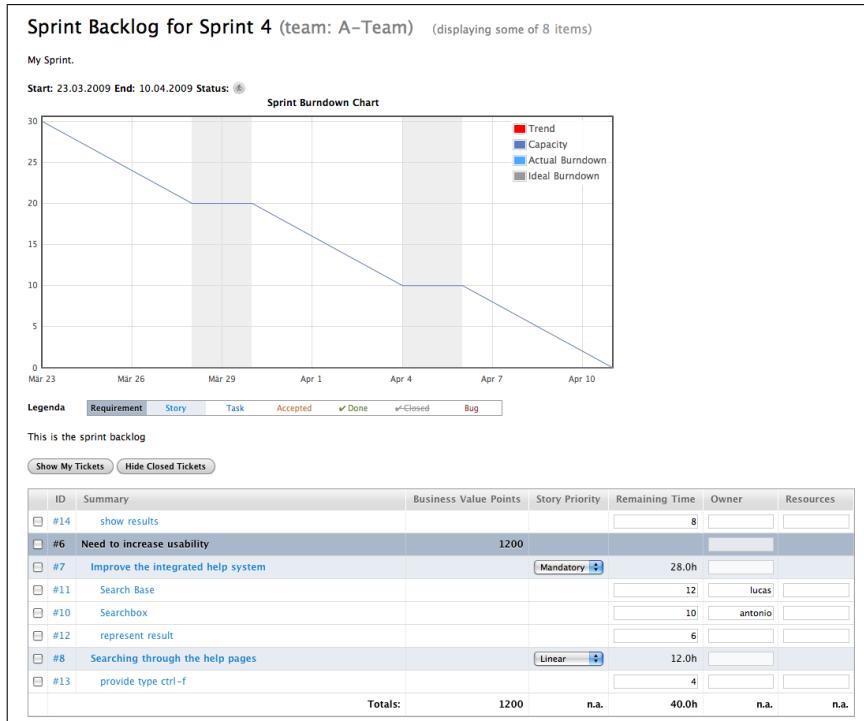


Abbildung A.7: Agilo - Sprint Backlog und Burndown Chart

← Previous Change | Next Change →

Changeset 7533 for trunk/ChangeLog

Timestamp: 09/12/08 16:53:43 (6 months ago)
Author: cboos

Message: 0.12dev: merged [7405,7414,7447] via svnmerge from
[branches/0.11-stable](#)

Files: 1 modified
trunk/ChangeLog (3 diffs)

View differences inline

Show lines around each change

Ignore:
 Blank lines
 Case changes
 White space changes

Unmodified Added Removed

r7532	r7533	trunk/ChangeLog	Tabular Unified
1	1	Trac 0.11.1 (??, 2008)	
2	2	Trac 0.11.1 (August 6, 2008)	
3	3	http://svn.edgewall.org/repos/trac/tags/trac-0.11.1	
...	...		
5	5	The following list contains only a few highlights:	
6	6		
7	7	* Safer default umask value for tracd (can be set using --umask option)	
8	8	* Improved DB connection handling (new connection pool)	
9	8	* Better MySQL backend unicode support. "utf8" and "utf8_bin" is the	
...	...		
12	11	* error messages.	
13	12	* Fixes roadmap layout glitch in Firefox 3.	
13	12	* Safer default umask value for tracd (can be set using --umask option)	
14	13	* Better default PYTHON_EGG_CACHE value.	
14	15		
15	16	The complete list of closed tickets can be found here:	

Abbildung A.8: Versionsvergleich in Trac