Open Source Software Development*

Matthias Luder

9. Dezember 2003

Inhaltsverzeichnis

*Seminararbeit in Software Engineering zum Thema Agile vs. klassische Methoden der Software-Entwicklung. Vorgelegt von Matthias Luder, Solothurn, 00-713-248. Angefertigt am Institut für Informatik der Universität Zürich, Prof. Dr. M. Glinz. Assistent: Christian Seybold.

1 Einleitung

Open Source Software hat in den letzten zehn Jahren einen starken Reifungsprozess durchgemacht. Mittlerweile interessieren sich viele Firmen für Open Source Lösungen, sei es als Kunde oder als Entwickler. Auch deren Prozesse, Methoden haben sich in dieser Zeit verändert und mit der weiteren Verbreitung auch vermanigfaltigt.

Der Versuch, gemeinsame Prozesse und Methoden zu beschreiben, führt zwangsläufig dazu, dass stark abstrahiert werden muss, um die Allgemeingültigkeit zu wahren.

Das folgende Kapitel betrachtet den Begriff Open Source Software Development von verschiedenen Seiten.

In einem nächsten Kapitel werden die Prozesse und Methoden mit denjenigen der agilen Software Entwicklung verglichen. Denn auch agile Software Entwicklung hat in der letzten Zeit stark an Aufmerksamkeit gewonnen.

2 Open Source Software

2.1 Definition von Open Source Software

Der Namen "Open Source" konnte bis heute nicht geschützt werden. Wird der Name im eigentlichen Sinne für Software verwendet, reicht das Offenlegen des Codes nicht aus. Open Source Software definiert sich als Software, die den Bedingungen der Open Source Definition entsprechen [2]. Die Open Source Definition ist ein Dokument, das von der Open Source Initiative unterhalten wird [11]:

> Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:
>
> **1. Free Redistribution**
>
> The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
>
> **2. Source Code**
>
> The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well–publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the

program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code
The license may restrict source–code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups
The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor
The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License
The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product
The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software
The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral
No provision of the license may be predicated on any individual technology or style of interface.

Die Kriterien der Open Source Definition sollen es dem Benutzer erlauben, die Software in irgendeiner Form weiterzuverteilen, was ebenfalls den Verkauf einschliesst. Der Zugang zum Quellcode muss gewährleistet sein, so dass dieser untersucht oder verändert werden kann. Das dritte Kriterium erweitert den Sinn von einsehbarem Code, indem verlangt wird, dass dieser frei veränderbar ist und dass Änderungen weitergeben werden können. Die

Möglichkeit für den Benutzer, den Code zu ändern ist der eigentliche Kerngedanke der Open Source Software; diese erlaubt überhaupt 'peer review' und 'parallel development' [2]. Die Lizenz kann allerdings verlangen, dass Änderungen im Code durch ändern des Namens sichtbar gemacht werden müssen, was somit eine gewisse Kontrolle über Projekte gewährleistet. Des weiteren darf die Lizenz keine Gruppen und Produkte oder Gebrauchsbereiche oder dritte Software auschliessen bzw. einschränken. Punkt sieben soll verhindern, dass die Lizenz "geschlossen" [2] werden kann.

Von der Open Source Definition abgeleitet gibt es zahlreiche Lizenzen und Lizenzmodelle, wie zum Beispiel die GNU General Public Licence (GPL), die Berkeley Software Distribution (BSD) Licence, um nur zwei (bedeutende) zu nennen. Die Lizenzen stellen eine wichtigen Bestandteil der Open Source Software dar; weitere Ausführungen würden den Rahmen dieser Arbeit sprengen.

2.2 Geschichte

Die Entstehung von Open Source Software kann nicht auf ein bestimmtes Ereignis zurückgeführt werden.

Verschiedene Personen und Gruppierungen begannen erstmals in den frühen 80iger Jahren, den Code ihrer Programme offenzulegen und anderen zugänglich zu machen, meist einem kleinen Kreis von Interessierten. Zu den Pionieren unter anderen, welche namentlich genannt werden sollten, gehören Bill Joy, der eine Gruppe von Softwareingeniuere anführte - die spätere Berkeley Software Distribution (BSD)-, Donald Knuth und Richard Stallmann. Die BSD Gruppe (Angestellte bei AT&T/Western Electric) modifizierte und verbesserte das damalige Unix, ursprünglich in den Bell Telephone Labaratories [7] entwickelt, und verteilte es, da AT&T wegen Antikartellbestimmungen keine Gewinne ausserhalb ihres Tätigkeitsbereichs erwirtschaften durfte, an wenige Personen, welche wiederum über 'usegroups' auf die weitere Entwicklung Einfluss nahmen. So entwickelten sich duzende Unix–Abarten, welche zumeist nur auf einem System liefen und vollständig inkompatibel waren. Dass Unix seinen Ursprung im Telekommunikationssektor hat ist nicht ganz zufällig; Rechenzeit und Telematik Dienstleistungen waren zu dieser Zeit keine Massenware und nur einem kleinen Kreis von Entwicklern zugänglich.

Auch Donald Knuth kann als Vordenker des Open Source Geistes angesehen werden. Das von ihm entwickelte und immer noch weltweit verwendete Satzprogramm TeX war das erste grosse Software Projekt, bei dem der Code offen gelegt wurde [2].

Richard Stallmann, der 1985 die Free Software Foundation gründete, investierte einen grossen Teil seiner Zeit in die Software der GNU Familie (GNU steht für GNU is not unix), aus der der berühmte GNU C Compiler GCC stammt.

Die im Zusammenhang mit Open Source wohl bekannteste Person ist Linus Torvalds. Den Wunsch ein Unix Betriebssystem auf seinem IBM PC 386 zu installieren, veranlassten den 21-jährigen Studenten dazu, ein eigenes Betriebssystem zu entwickeln bzw. bestehendes umzuschreiben. Als Vorlage diente ihm Minix, ein Unix-Derivat entwickelt von Andrew Tanenbaum an der Freien Universität von Amsterdam.

Mit einer ersten lauffähigen Version von Linux (der erste Name war Freax - von free, freak und dem obligaten "x") bat Torvalds 1991 zuerst Freunde und anschliessend die ganze Welt um Hilfe bei seinem Projekt. Der Hilferuf fand Resonanz und so halfen tausende von Entwicklern weltweit mit. Die Schätzungen über die Zahl der Entwicklern geht weit auseinander, nach Torvalds selbst sind es bis "hundreds of thousands of developers" [8], wobei 1200 Entwickler [2] für ein solches Projekt realistisch erscheint. Es wird geschätzt, dass Microsoft für sein Produkt "Windows 2000" über 400 Entwickler und 250 Tester engagiert hat - die Zahl von Beta Testern wird auf 100'000 geschätzt [2].

Mittlerweile ist Linux weiter verbreitet als Unix, auf dem es eigentlich basiert und viele Unix-Varianten sind verschwunden. Der Erfolg des Linux Kernels bereitet auch dem Open Source Gedanken ein feste Basis, so dass Linux oft als Synonym für Open Source gehalten wird.

Weitere grosse Open Source Projekte wie der Webserver Apache folgten und seit Netscape das Open Source Projekt Mozilla (bestehend aus dem Netscape Code) gestartet hat, lancierten viele gestanden Software Unternehmen eigene Open Source Projekte. Open Source gehört heute zum "mainstream" [8] in der Software Entwicklung; die Weiterentwicklung von der Informationstechnologie macht nicht halt, ebenso bleibt Open Source Entwicklung ein dynamischer Prozess und entwickelt sich ständig weiter.

Dass sich Open Source Software in den frühen 90iger Jahren verbreitete, geht mit der Erschwinglichkeit von Tele- und Informationstechnologie (für Studenten) einher. Der Personal Computer und das Ur-Internet mit den 'usegroups' können als treibende Kraft für Open Source angesehen werden.

2.3 Die Bazaar Metpaher

Der Artikel von Eric S. Raymond "The Cathedral and the Bazaar" [6] ist eines der ersten und das wohl bekannteste Dokumente zur Entwicklung von

Open Source Software [5]. In seinem Artikel setzt Raymond die klassische Software Entwicklung mit dem Bau einer mittelalterlichen Kathedrale gleich. Beim Bau einer Kathedrale gibt es eine klare Verteilung der Rollen und Aufgaben. Der Projektleiter entwirft und plant die Kathedrale von Anfang bis zum Schluss minutiös und legt die Prozesse bis ins Detail fest. Für Projektmitarbeiter gibt es nur einen sehr begrenzten Spielraum, ihre eigenen Ideen einzubringen.

Dem Kathedralen Modell steht der Bazaar gegenüber, welcher für die freie Software steht. Auf dem Bazaar gibt es keine zentrale Aufsicht, welche die Macht hätte, Prozesse vollständig zu kontrollieren oder die nächste Ausbaustufe zu planen. Auch die Rolle der Teilnehmer kann sich ständig ändern, Verkäufer können zum Kunden werden und vice versa, ohne Instruktionen von aussen. Der Bazaar hält einige gute Methoden bereit, welche den Vorteil und die Möglichkeiten des offenliegenden Quellcodes nutzt. Das Einbinden der Co-Entwickler und der Benutzer in den Prozess der Entwicklung gewährt eine hohe Qualität, da die Wahrscheinlichkeit, dass Fehler gefunden oder korrigiert werden, gross ist.

2.4 Projekt Lebenszyklus

Open Source Projekte folgen keinem strikten Muster für Releases, die Lebenszyklen können unterschiedlich aussehen und im Verlauf der Zeit sogar ändern. Ein allgemeiner Lebenszyklus könnte folgendermassen aussehen [7]:

1. Ein persönliches, unbefriedigtes Bedürfnis zwingt jemanden dazu eine eigene Lösung zu entwickeln, was Raymond "scratching an itch“ [6] nennt. Der Initiator muss aber die Fähigkeit haben, zumindest eine erste einigermassen brauchbare Lösung zu entwerfen.

2. Diese Person fragt seine Freunde, was sie über dieses Problem wissen. Einige können die gleichen oder ähnliche Probleme haben, aber ohne eigene Lösung.

3. Alle interessierten Personen tauschen ihr Wissen über das Thema. Dabei wird ein vages Bild über die zentralen Anforderungen gezeichnet.

4. Eine interessierte Gruppe, welche bereit ist, ihre Ressourcen in die Lösungsfindung zu investieren, startet ein informelles Projekt.

5. Die Projektmitarbeiter arbeiten an einer Aufgabe bis sie eine befriedigende Lösung haben.

6. Die Lösung wird an einer zentralen Stelle der Öffentlichkeit zur Verfügung gestellt, so dass möglichst viele Personen Zugang haben.

7. Weitere Personen erkennen eigene Bedürfnisse in diesem Projekt und sind ebenfalls an einer überzeugenden Lösung interessiert. Sie werden die Lösung reviewen (zum Beispiel durch deren Gebrauch). Dadurch, dass sie das Projekt unter einem anderen, neuen Winkel betrachten, können Verbesserungsvorschläge einfliessen oder sie können sogar am Projekt selber mitarbeiten.

8. Das Projekt wächst und eine grosse Menge an Feedback hilft das Thema besser zu verstehen und zeigt viele verschiedene Lösungsstrategien.

9. Neu Informationen und Ressourcen kommen hinzu. Die Lösung wird immer besser angenähert.

10. Ein Zyklus ist geschlossen und es kann wieder mit Punkt 5 begonnen werden.

11. Eine Projektgemeinschaft hat sich eingeschworen und kann auf neue Forderungen eingehen.

Eine allgemeine Klassifikation von Open Source Software kann wie folgt vorgenommen werden:

- Planning
- Pre-Alpha
- Alpha
- Beta
- Stable
- Mature

In der Planungsphase liegt auschliesslich eine Idee vor. Noch kein Code ist geschrieben worden. Mit dem ersten Code kommt das Projekt in die Pre-Alpha Phase. Dabei liegen die ersten Codefragmente vor, welche nicht zwingend kompiliert werden können. Kann der Code oder ein Teil davon kompiliert werden und kann man eine Richtung des Projektes erkennen, so spricht man von Alpha. Sind die meisten Funktionalitäten hinzugefügt hat man Beta-Software. Diese kann noch beträchtliche Fehler erhalten. Ist die Software

dann von den meiste Fehlern befreit und für den täglichen Gebrauch geeignet spricht man von Stable-Software. Die letzte Phase ist Mature. An der Software werden nur noch kleinste Änderungen vorgenommen, falls überhaupt.

Interessant ist die Verteilung der Projektstati über eine grosse Zahl von Projekten anzusehen, dargestellt in Tabelle 1.

Projektstatus	Sourceforge.net	Freshmeat.net
Planning	13522 (25%)	71 (0%)
Pre-Alpha	9590 (18%)	635 (4%)
Alpha	9036 (17%)	2453 (14%)
Beta	10981 (21%)	5511 (32%)
Stable	8751 (16%)	7566 (44%)
Mature	836 (2%)	1146 (7%)

Tabelle 1: Open Source Projekte auf den Seiten Sourceforge.net [12] und Freshmeat.net [10] aufgeliestet nach Projektstatus (Stand 22.11.2003)

Die meisten Open Source Projekte auf der Seite Sourceforge.net [12] befinden sich in der Planungsphase. Das kann damit erklärt werden, dass es sehr einfach ist ein Projekt aufzusetzen und nur wenige Minuten dauert. Die Erfolgsrate für wenig überlegte Projektideen scheint nicht sehr hoch zu sein und Raymond [6] argumentiert, dass zumindest die ersten Funktionalitäten (Pre-Alpha oder Alpha) vom Initiant selbst entworfen werden müssen, bevor andere Entwickler an Bord genommen werden können.

2.5 Prozesse und Organisation

Obwohl es nicht den speziellen Prozess für die Herstellung von Open Source Software gibt, kann man gewisse allgemeingültige Annahmen treffen.

- Die Anwendung von klassischen Methoden kann nicht ausgeschlossen werden; dies hat mit dem Hinzutreten der gestandenen Software Entwicklungsfirmen wahrscheinlich noch zugenommen.
- Die Problemstellung wird von den Entwicklern meist freiwillig angegangen
- Die Entwicklung findet in kleinen Gruppen statt, welche meistens von einem (Projekt-)Leiter angeführt werden [1].

- Das System wächst in kleinen Inkrementen.
- Die Art wie die Anforderungen in einem Projekt integriert werden, hängt sehr stark vom Entwicklungsstand und der Grösse des Projektes ab. In einer ersten Projektphase sind Entwickler und Benutzer einunddieselbe Person. Zu einem späteren Zeitpunkt werden die Anforderung über Email Listen und Diskussionsforen geführt. Darin kann man klar die Anforderungen von aktiven Entwicklern und Endbenutzern unterscheiden [5].
- Mit zunehmendem Interesse an einem Open Source Projekt steigt auch die geographische Verteilung. Klassische Projekte sind im Gegensatz dazu physisch zentralisiert mit unternehmensweiter standartisierter Soft- und Hardware. Die geographische Verteilung der Open Source Mitarbeiter und Benutzer stellt also nicht nur höhere Anforderungen an die technische und sprachlichen Kommunikation, sondern erhöht auch direkt die Hard- und Softwarekompatibilität der entwickelten Software.
- Ein direkte finanzieller Anreiz ist nicht gegeben [5].
- Open Source Development geschieht formlos; Prozessdokumentation fehlt [1].
- Ein Problem stellt das Fehlen eines Gesamtprojektdesigns, dies beobachtet man häufig bei kleinen Projekten. Bei kleineren Projekten ist ausschliesslich der Entwickler im Besitz einer Gesamtübersicht, jedenfalls in seinem Kopf.
- Ein Projekt braucht ein gemeinsames Interesse der Entwickler; Software für ein Röntgenapparat als Open Source zu entwickeln dürfte sich schwierig gestalten.

3 Agile Methoden im Vergleich mit Open Source

Der Vergleich von Open Source und Agilen Methoden ist nicht ganz bedenkenlos. Open Source hat nicht wirklich eine Methode oder ein Prozess zum Entwickeln von Software. Aus den verschiedenen Prozessen, die hinter Open Source stehen, kann man aber gewisse Gemeinsamkeiten zum Vergleich mit den ebenso vielfältigen Agilen Methoden heranziehen.

Agile Methoden gehen von den folgenden Prinzipien aus [4]:

- Je kleiner die Teilaufgaben und je kürzer die Rückkopplungszyklen desto besser
- Der Kunde gestaltet das Produkt während seiner Entstehung
- Je freier und konzentrierter die Entwickler arbeiten können, desto besser
- Die Qualität wird an der Quelle sichergestellt
- Keine Arbeit auf Vorrat

3.1 Kleine Teilaufgaben und kurze Rückkopplungszyklen

Die kleinen Teilaufgaben bei den Agilen Methoden verringern den Planungsaufwand, da sie sich in der Regel in einem Schritt erledigen lassen [4]. In kurzen Iterationen werden die Teilschritte in das Gesamtprojekt integriert und ausgeliefert. Die kurzfristige Planung und schnelle Auslieferung verkürzt den Rückkopplungszyklus mit dem Kunden. Anforderungen können so laufend aufgenommen werden und jede abgenommene Iteration kann als Fundament für weitere Schritte angesehen werden [3].

Die Open Source Software Entwicklung startet meistens mit frühen und häufigen Releases. Die typische Weiterentwicklung funktioniert ebenfalls mit kleinen Inkrementen. "Release early, release often" [6], wird bei Open Source Software aber viel stärker ausgeprägt beobachtet. Durch die dezentrale Entwicklung sieht man sich gezwungen, zum Teil mehrfach täglich die neuesten Releases zur Verfügung zu stellen. Dies ist bei Agilen Methoden nicht möglich, da der Kunde höhere Anforderungen an die Releases hat; es werden stabile Versionen erwartet. Jede Iteration stellt die Grundlage für den nächsten Zyklus dar und sollte abgeschlossen sein. Open Source Entwicklung ermöglicht hingegen auch paralleles Entwickeln. Mehrere Module könne von verschiedenen Entwickler(teams) gleichzeitig vorangetrieben werden. Dies erhöht nicht nur die Geschwindigkeit der Produktion, sondern im Nebeneffekt auch die Kohäsion der einzelnen Module; ein Grund für relativ hohe Qualität [2] der Software. Die Kohäsion der einzelnen Iterationen kann bei den agilen Methoden nicht schlüssig beurteilt werden.

3.2 Der Kunde gestaltet das Produkt

Agile Methoden setzen voraus, dass der Kunde aktiv an der Produktion seiner Software mitarbeitet. Dafür muss er nicht von Anfang an wissen, wie die Software am Schluss aussehen soll. Die wenigsten Kunden können

ihre Anforderungen im voraus genau spezifizieren, wieso sich Anforderungen zwangsläufig ändern.

Von Open Source Kunden muss die Mitarbeit nicht direkt verlangt werden, die Mitarbeit geschieht freiwillig und wird v.a. durch Motivationsanreize gesteuert (z.B. durch direktes Feedback). Oft ist der Kunde auch Entwickler selbst. Dies garantiert auf der einen Seite eine reibungslose Zusammenarbeit (zumindest aus sprachlicher Sicht), provoziert aber auf der anderen komplexere Anforderungen und ebensowenig sind Machtkämpfe unter den Entwicklern ausgeschlossen.

3.3 Freies und konzentriertes Arbeiten der Entwickler

Auch die Agilen Methoden haben erkannt, dass der Mensch im Zentrum des Entwicklungsprozesses steht. Unzufriedene Arbeiter leisten bedeutend weniger. Agile Methoden versuchen durch einfache gestaltete Prozesse und das Offenlassen von Freiräumen, eine Produktive Umgebung zu schaffen.

Die grösstmöglichen Freiheiten hat man auf dem "Bazaar"; die Motivation hinter der Open Source Entwicklung ist dieselbe. Man könnte argumentieren, dass die Motivation höher ist, da es sich um "a personal itch" [6], eine persönliche Herausforderung handelt. Stellt man dem aber die hohe Anzahl an Projekten gegenüber, welche aufgegeben wurden, weil das Interesse verloren ging, relativiert sich dieses Argument.

3.4 Qualität an der Quelle sichergestellt

Qualitätssicherung gestaltet sich bei den Agilen Methoden nicht einfach. Die geringe Planbarkeit verlangt, dass die Qualität an der Quelle sichergestellt wird. Dies funktioniert zum Beispiel, indem die Software als Pair Programming [4] entwickelt wird. Zudem sollte jedes Inkrement sorgfältig mit einem Satz von Testfällen getestet werden.

Die Qualität wird bei Open Source Software durch den Benutzer sichergestellt. Als Co-Entwickler übernehmen sie das Debugging. Es ist zu erwarten, dass Probleme gefunden werden und somit von jemandem gelöst werden wird [6].

Die Qualität von Open Source Software wird generell als hoch angesehen. Die hat verschiedene Gründe. Zum einen werden Open Source Projekte erst relativ spät und ohne Zeitdruck als stabiler Release freigegeben und somit offiziell tauglich für den täglichen Gebrauch. Agile Methode haben sich genauso wie klassische Methoden an Zeitpläne zu halten, die entweder vom

Kunde oder durch den Markt vorgeben werden; Software wird früher freigegeben, als ihr Entwicklungsstand eigentlich erlauben würde.

Zum andere wird von Vertretern der Open Source Entwicklung (u.a. [6] und [2]) argumentiert, dass sich in Open Source Projekten besonders erfahrene Entwickler zusammenfinden. Nach Untersuchungen [2] entwickeln 20% der Entwickler rund 80% des Codes. Die Entwickler trennen sich also in ein Kernteam und eine grosse Menge an Gelegenheitsmitarbeiter. Da es selten Dokumentationen gibt, welche die Gesamtkomposition des Projektes wiedergibt, macht es einem Entwickler schwer, dem Kernteam beizutreten, jedenfalls nur mit einem immensen Aufwand. Empirische Untersuchungen [7] mittels Stichproben in verschiedenen Freshmeat.net [10] und Sourceforg.net [12] Projekten haben allerdings ergeben, dass sich die Erfahrung der Entwickler nicht von klassischen Entwickler Teams unterscheiden lässt.

3.5 Keine Arbeit auf Vorrat

Die Struktur der Lösung wird bei agiler Software Entwicklung nicht vorausgeplant, sie entwickelt sich mit dem Verlauf des Projekts [4]. Die bis anhin entwickelten Teillösungen werden nur bei Problemen restrukturiert und Lösungen bleiben sehr problemspezifisch.

Die Restrukturierung nach Bedarf entspricht ganz der Vorgehensweise bei Open Source Software. Die räumliche Distribution und die Heterogenität der Nutzer bei grossen Projekten erfordert einen hohen Grad an Flexibilität. Für gewisse Problemstellungen findet man nicht nur eine spezifische Lösung, sonder eine generelle Lösung des Problems.

4 Endbetrachtung

Wir haben festgestellt, dass es zwischen den Agilen Methoden und der Entwicklung von Open Source Software auf einem sehr allgemeinen Niveau Parallelen gibt. Im Zentrum steht vor allem der Mensch, sowohl der Entwickler, der möglichst viele Freiheiten geniessen möchte, als auch der Kunde, der möglichst stark in den Entwicklungsprozess eingebunden wird. Detaillierte und langfristige Planung fehlen; ihre Abwesenheit ist somit massgebend für die Form der Prozesse. Die Rahmenbedingungen sind aber für beide Methoden andere. Agile Methoden werden meist für einen Kunden (oder Kundengruppe) entworfen, wohingegen bei Open Source der Unterschied zwischen Kunde und Entwickler fliessend ist.

Versucht man die einzelnen Methoden (Extreme Programming, Scrum, Context Driven Testing usw.) aus der Agilen Entwicklung, welche eigentlich

nur eine Zusammenstellung von 'best practice' [9] Ansätzen sind, mit dem Open Source Paradigma zu vergleichen, stellt man fest, dass der Unterschied doch viel grösser ist, als auf dem konzeptionellen Niveau.

Insofern stellt sich die Frage, ob solche Vergleiche überhaupt angestellt werden können. Die unterschiedlichen Rahmenbedingungen müssen berücksichtigt werden und es ist sicher falsch Open Source Software Entwicklung in die Reihe der agilen Methoden einordnen zu wollen.

Literatur

[1] Boldyreff, C., Lavery, J., Nutter, D., Rank, St. (2003): Open-Source Development Proceses and Tools. In: 3rd Workshop on Open Source Software Engineering. International Conference on Software Engineering, Oregon.

[2] Feller, J., Fitzgeral, B. (2002): Understanding Open Source Software Development. Addision-Wesley Longman, Amsterdam.

[3] Fowler, M. (2003): The New Methodology. http://www.martinfowler.com/articles/newmethodology.html (18.10.2003)

[4] Glinz, M. (2002): Software Engineering I. Vorlesungsskript. Institut für Informatik der Universität Zürich, Zürich.

[5] González-Barahona, J.M., Robles, G. (2003): Free Software Engineering: A Field to Explore. Upgrade. Vol. IV, No.4. Software Engineering - State of an Art. http://www.upgrade-cepis.org (15.11.2003)

[6] Raymond, E.S. (1997): The Cathedral and the Bazaar. Version as of 1998/08/11. http://www.openresources.com/documents/cathedral-bazaar/ (15.11.2003)

[7] Rothfuss, G.J. (2002): A Framework for Open Source Projects. Master Thesis in Computer Science. Departement of Information Technology, Universität Zürich.

[8] Torvalds, L., Diamond, D. (2001): Just for Fun: The Story of an Accidental Revolutionary. HarperCollins, New York.

[9] Warsta, J., Abrahamsson, P. (2003). Is Open Source Development Essentially an Agile Method? In: 3rd Workshop on Open Source Software Engineering. International Conference on Software Engineering, Oregon.

[10] http://freshmeat.net (22.11.2003)

[11] http://www.opensource.org (15.11.2003)

[12] http://sourceforge.net (22.11.2003)

Tabellenverzeichnis