Analyse der Übertragbarkeit der Open Source-Entwicklungsmethodik in den kommerziellen Bereich

Tobias Hildenbrand and Michael Geisser and Markus Nospers

Working Paper 5/2006
Februar 2006

Working Papers in Information Systems

University of Mannheim
Department of Information Systems 1
D-68131 Mannheim/Germany
Phone +49 621 1811691, Fax +49 621 1811692
E-Mail: wifo1@uni-mannheim.de
Internet: `http://www.bwl.uni-mannheim.de/wifo1`

Analyse der Übertragbarkeit der Open Source-Entwicklungsmethodik in den kommerziellen Bereich

Tobias Hildenbrand, Michael Geisser und Markus Nospers
Universität Mannheim
Lehrstuhl für ABWL und Wirtschaftsinformatik
[hildenbrand, geisser]@uni-mannheim.de, markus@nospers.de

1 Einleitung

Open Source-Software findet bereits seit einiger Zeit erfolgreich Einzug in die Unternehmenspraxis. Das quelloffene Betriebssystem GNU/Linux sowie der HTTP- Server Apache haben sich mittlerweile in vielen großen wie kleinen Unternehmen zu einem Standard etabliert. Eine wesentliche Grundlage für den Erfolg dieser und weiterer Open Source-Produkte bilden die entsprechenden Entwicklungsmethoden, die zur Unterstützung verteilter Zusammenarbeit innerhalb der Open Source Communities entstanden sind. Dies hat zur Folge, dass Einflüsse aus der Open Source-Welt auf zweierlei Arten in die Unternehmenspraxis diffundieren: Neben dem vermehrten Einzug von quelloffene Anwendungen in unternehmenskritische Bereiche einerseits führt die Tatsache, dass diese Anwendungen verteilt und quelloffen entwickelt werden, andererseits dazu, dass immer mehr Unternehmen dazu übergehen, die entsprechenden Methoden und Techniken auch in ihren kommerziellen Softwareprojekten einzusetzen. Vor allem Letzteres soll im Zentrum dieser Arbeit stehen.

Innerhalb und zwischen großen, weltweit agierenden Unternehmen ist die Zusammenarbeit von geografisch verteilten Entwicklerteams in der Regel unvermeidlich. Die stärker werdende Nachfrage nach Software für die unterschiedlichsten Geschäftsbereiche erfordert zudem immer mehr und immer besser qualifiziertes Personal, das selten an einem geografischen Ort gebündelt zu finden ist. Da dieser wachsende Bedarf an hochqualifizierten Arbeitskräften mit sehr hohen Personalkosten verbunden ist, sind viele Unternehmen gezwungen, große Teile ihrer Softwareentwicklungsprojekte in so genannte "Billiglohnländer" auszulagern. Das hierbei beobachtbare Lohngefälle stellt einen weiteren Treiber für global verteilte Entwicklungstätigkeiten dar.

Methoden und Werkzeuge aus dem Open Source-Bereich werden bereits in Form von so genannten "kollaborativen Softwareentwicklungsplattformen" kommerziell genutzt. Die drei Anbieter CollabNet, Intland und VA Software verkaufen und/oder hosten diese Plattformen als fertige Lösungen für Unternehmenskunden. Allerdings gehen Open Source-Projekte von anderen Prämissen aus als kommerzielle Softwareprojekte, beispielsweise öffentlich zugängige Quelltexte und freie Wahl der Projektaufgaben. Softwareprojekte im Unternehmenskontext hingegen stellen ihrerseits spezielle Anforderungen an eine Methodik für die verteilte Softwareerstellung – z.B. eine sorgfältige Erfassung der Kundenanforderungen.

Zielsetzung dieses Beitrags ist es daher, die Vorteilhaftigkeit und Anwendbarkeit von Methoden und Werkzeugen aus dem Open Source-Bereich im Unternehmensumfeld und insbesondere in geografisch verteilten Szenarien zu untersuchen. Als beispielhaftes Szenario für die mögliche Überlegenheit von Open Source-basierten Methoden gegenüber traditionellen Vorgehensmodellen in einem verteilten Umfeld soll ein Offshore-Entwicklungsprojekt mit einem fernöstlichen Land dienen.

Der restliche Beitrag gliedert sich daher in zwei grundlegende Abschnitte, die den Stand der Technik bezüglich der Methoden und der Werkzeuge im Open Source-Bereich widerspiegeln (Abschnitte 2 und 3). Die Anwendbarkeit des Open Source-Modells für ein verteiltes Umfeld im Unternehmenskontext wird in Abschnitt 4 analysiert und anschließend in Abschnitt 5 anhand des Beispielszenarios "Offshore-Softwareentwicklung" bezüglich der Vorteilhaftigkeit dieser Vorgehensweise gegenüber herkömmlichen Methoden evaluiert. Eine Zusammenfassung sowie einen Ausblick auf künftige Forschungsarbeiten bietet schlussendlich Abschnitt 6.

2 Die Open Source-Entwicklungsmethodik

Zur erfolgreichen Erstellung von Open Source-Software (OSS) haben sich über die Jahre zahlreiche methodische Elemente und Werkzeuge als Defacto-Standards etabliert. Im Folgenden soll daher der Stand der Technik hinsichtlich der aktuell beobachtbaren Open Source-Entwicklungsmethodik wiedergegeben und analysiert werden.

OSS-Entwicklung (OSSE) ist keine klar definierte und festgeschriebene Entwicklungsmethodik wie beispielsweise das V-Modell, sondern vielmehr eine evolutionär gewachsene und flexible Sammlung von "Best Practices", anhand derer verteilte Softwareentwicklungsvorhaben effizient durchgeführt werden können. Es existieren hierbei im Gegensatz zu traditionellen Entwicklungsmethoden keine strikten Vorschriften, die eine Aufteilung des Entwicklungsprozesses in bestimmte zeitlich abgrenzbare Phasen vorschreiben. Allerdings weist die OSSE viele Gemeinsamkeiten mit den bekannten agilen Methoden auf [26], so z.B. die schnellen und stetigen Veröffentlichungen neuer Versionen der Software oder die kontinuierliche Berück-

sichtigung von Anforderungen an das System [1].

Im Vergleich zu traditionellen sequenziellen und/oder iterativ zyklischen Softwareentwicklungsmethoden wie dem V-Modell XT bzw. dem Rational Unified Process gibt es im OSSE-Prozess keine fest definierte Phaseneinteilung. Allerdings durchlaufen auch die meisten OSSE-Projekte einige, allerdings nicht strikt getrennte, Schritte. An erster Stelle steht dabei die Problemidentifikation, der in den meisten Projekten aber kein wirtschaftliches Interesse zu Grunde liegt, sondern der persönliche Eigennutzen der Entwickler ("Scratching a developer's personal itch" [22]) ausschlaggebend ist. Auf die Problemidentifikation folgen die Suche nach freiwilligen Helfern, die Identifikation der möglichen Lösung und die Erstellung des eigentlichen Quelltextes [24]. Diese drei Schritte sind allerdings nicht klar zu trennen, da zunächst eine kritische Masse an Quelltext und eine erste rudimentär lauffähige Version erstellt werden muss, um weitere freiwillige Programmierer in das Vorhaben involvieren zu können.

Bei der Offenlegung ehemals kommerzieller Programme und deren Weiterentwicklung durch die Open Source Community (OSC) hingegen entstehen diese Arten von Problemen nicht (vgl. beispielsweise OpenOffice.org und Mozilla). Hier ist die kritische Masse in Form einer lauffähigen Applikation und des entsprechenden Quelltextes bereits vorhanden. Diese Komplexität kann allerdings den Einstieg neuer, vor allem nicht professioneller Programmierer erschweren. Genau dieses Phänomen führte beispielsweise zu den Anlaufschwierigkeiten bei OpenOffice.org nach der Offenlegung von StarOffice durch Sun Microsystems. Allerdings behelfen sich die OSC zur Bewältigung komplexer Code-Strukturen mittlerweile spezieller Werkzeugen zum Verwalten und Durchsuchen der Quelltexte (siehe nächster Abschnitt).

Der Entwurf einer expliziten Systemarchitektur in grafischer Ausarbeitung findet bei OSSE-Projekten nur selten statt, so dass sich der Hauptteil der Entwicklung mit der Programmierung beschäftigt. Dieser liegen Praktiken zu Grunde, die sich in zahlreichen Open Source-Projekten wiederfinden und sich zusammen genommen als eine verteilte agile Entwicklungsmethodik für mehrere Entwickler(-teams) charakterisieren lassen [26]. Grundvoraussetzung hierbei ist eine modulare Struktur des zu entwickelnden Systems [18]. Die Modularisierbarkeit wird mit zunehmender Größe des Gesamtprojektes umso wichtiger und ermöglicht erst die vollständig dezentrale, zeitlich und räumlich verteilte Entwicklung komplexer Softwaresysteme. Somit wird einerseits eine schnellere Einarbeitung neuer Programmierer ermöglicht, ohne dass diese das System in seiner Gesamtheit begreifen müssen. Zum anderen wird ein schneller Entwicklungsprozess möglich, da sich zahlreiche Aktivitäten bei der Programmierung der einzelnen Module parallelisieren lassen [10]. Eine reibungslose Zusammenarbeit der einzelnen Module setzt allerdings eine klare und offene Schnittstellendefinition voraus. Hierbei wird darauf geachtet, soweit möglich, bereits existierende standardisierte Schnittstellen zu verwenden [5]. Die Modularisierung zum einen und der offen zugängliche Quelltext ermöglichen und fördern zudem die Wiederverwendung innerhalb und über Projektgrenzen hinweg. Hierbei geht es nicht nur um die Wiederverwendung ganzer Module oder Komponenten, sondern auch um die erneute Nutzung feingranularerer Quelltextsegmente und individueller Lösungen sowie des gespeicherten Wissens anderer Entwickler und Anwender [20].

Das Testen der Software wird als ein hoch parallelisierbarer Prozess angesehen und diese Parallelisierbarkeit wird umso stärker ausgenutzt je mehr Entwickler und/oder Endanwender das Programm nutzen ("Given enough eyeballs, all bugs are shallow" [22]). Auf diese Weise wird es auch ermöglicht, dass sicherheitsrelevante Programmteile eingehend von den Benutzern selbst untersucht und validiert werden können und somit bei kritischen Anwendungen kein blindes Vertrauen mehr geleistet werden muss. Nicht zuletzt ist es typisch für Open Source-Projekte, dass sie extrem kurze Freigabezyklen haben, was Eric S. Raymond als das Linus' Law bezeichnet ("Release early, Release often" [22]). Weitere Darstellungen des gemeinen OSSE-Lebenszyklus finden sich unter anderen bei Jorgensen [16] sowie Feller und Fitzgerald [11]).

Kernstück zur Unterstützung des beschriebenen Softwareerstellungsprozesses stellt eine offene Quelltextverwaltung dar. Es ist dabei in der Regel allen Interessierten möglich, über standardisierte Schnittstellen (siehe Abschnitt 3) den gesamten aktuellen Quelltext einzusehen – gegebenenfalls auch eine vollständige Versionshistorie. Die Entwickler besitzen in aller Regel nur Schreibrechte für die von ihnen bearbeiteten Programmteile und einzelne, so genannte "Maintainer" verfügen über Schreibrechte für Teilprojekte oder das gesamte Projekt [5]. Bei vielen OSS-Projekten ist ein stabiler Zweig abgetrennt, in den nur ausgewählte Maintainer validierten Quelltext in die stabile Version des Projekts übernehmen. Folglich liegt im Open Source-Bereich keine vollkommene "Single-Source-Base" vor, wie sie z.B. im Rahmen von eXtreme Programming propagiert wird [4]. Die vollständige Verfügbarkeit hat einen wichtigen qualitätssichernden Aspekt, da somit gegenseitige Kontrolle ("Peer-Reviewing") ermöglicht wird. Somit hat jeder die Möglichkeit, den Quelltext auf Funktionalität und Qualität zu testen, zu bewerten und gegebenenfalls Verbesserungsvorschläge einzubringen.

Die Kommunikation stellt ebenfalls einen zentralen Faktor im verteilten Open Source-Umfeld dar, da sie in einem verteilten Umfeld für die Koordination paralleler Aktivitäten bzw. den Informationsaus-

	Entwicklungsmethodiken		
	Agile Softwareentwicklung	**OSSE**	**Trad. Softwareentwicklung**
Entwickler	hochqualifiziert, agil, am selben Ort, kollaborativ, horizontale Arbeitsteilung	hochqualifiziert, agile Teams, geografisch verteilt, Domänenwissen, horizontale Arbeitsteilung	Spezialwissen, planorientiert, weniger Domänenwissen, vertikale Arbeitsteilung
Kunden	engagiert, vor Ort präsent, kollaborativ, repräsentativ, einflussreich	Kunde=(Mit-)Entwickler, engagiert, qualifiziert, kollaborativ, einflussreich	Vermittler zwischengeschaltet, repräsentativ
Anforderungen	schnelle und stetige Anpassung, schnelle Fertigstellung	stetige Anpassung, Entwicklung nie beendet	früh festgelegt, nur bedingt anpassbar, Garantien
Größe	kleine Teams, kleine bis mittlere Produkte	große dezentrale Teams, kleine bis mittlere und spezialisierte Produkte	große Teams, große generalisierte Produkte

Tabelle 1: Vergleich der OSS-Entwicklung mit agilen und traditionellen Methodiken (vgl. Warsta [26])

tausch essenziell ist. Das Besondere hierbei ist, dass eine vollkommen offene Kommunikationspolitik betrieben wird. Jedem ist es möglich die verschiedenen Kommunikationsmedien zu abonnieren, zu lesen oder gegebenenfalls auch eigene Beiträge verfassen zu können. Entwickler anderer Programmteile oder sogar Projektfremde können somit einen transparenten Entwicklungsprozess verfolgen und auch aktiv an Diskussionen teilnehmen. Auf der Anwenderseite ist die Nähe zu den Entwicklern bemerkenswert, da sich nicht nur die Anwender untereinander helfen, sondern in vielen Fällen auch Entwickler direkt auf Probleme der Anwender eingehen. Die Nähe zu den Endnutzern ermöglicht zudem ein kontinuierliches Anforderungsmanagement, da dieselben Kommunikationswege auch für Feature- und Change-Requests genutzt werden können. Folglich können die Entwickler früh und direkt die Wünsche und Vorschläge der Endanwender wahrnehmen. Jeder Anwender bekommt so die Möglichkeit als "Co-Entwickler" am Prozess teilzunehmen. Dies zeichnet im Wesentlichen den kollaborativen Charakter der OSSE-Methodik aus.

Aus der Zusammenfassung der Charakteristiken der OSSE in Tabelle 1 lässt sich erkennen, dass es sich um einen sehr agilen Prozess handelt, der darauf zugeschnitten ist, eine große Anzahl von dezentralen Entwickler(-gruppen) an einem Projekt zusammen arbeiten zu lassen. Die grundlegenden Elemente, die dies ermöglichen, sind im Wesentlichen

1. die zentrale und offene Quelltextverwaltung mit rollenbasierten Schreib- und Zugriffsrechten sowie
2. die intensive, direkte und kontinuierliche Kommunikation aller Beteiligten.

Zur Unterstützung dieser und weiterer OSSE-Merkmale bedarf es spezieller Werkzeuge, die in einem verteilten Umfeld mit einer Vielzahl von Nutzern einsetzbar sind. Der folgende Abschnitt gibt daher einen Überblick über die häufig verwendeten Werkzeuge im OSSE-Bereich und deren kommerzielle Adoption im Unternehmensumfeld.

3 Werkzeuge für die verteilte, kollaborative Softwareentwicklung

Wie bereits im vorangegangenen Abschnitt angedeutet, gruppieren sich OSC im Wesentlichen um ein gemeinsames Quelltext-Repository zur Koordination der Entwicklungsaktivitäten sowie Mailinglisten zur Kommunikation. Hinzu kommen des Weiteren Webbasierte Diskussionsforen und Versionierungssysteme wie CVS und Subversion. Moderne Versionierungssysteme erlauben hierbei, über unterschiedliche Zugangsrechte und zentrale Versionsverwaltung unterschiedlicher Artefakte, die grundlegende Koordination von verteilten Open Source-Entwicklungsaktivitäten.

Kollaborative Softwareentwicklungsplattformen (KSEP) wie beispielsweise sourceforge.net[1] und JavaForge[2] bieten OSC diese Werkzeuge als Dienstleistungen für nicht-kommerzielle Entwicklungsvorhaben in der Regel kostenlos an. Mit GForge[3] und Savane[4] existieren zudem quelloffen verfügbare Lösungen, die durch einzelne Communities selbst betrieben und gegebenenfalls im Quelltext modifiziert werden können. Sowohl bei GForge als auch bei Savane handelt es sich um "Ableger" der ehemals quelloffenen sourceforge.net-Plattform [9].

Neben den bereits erwähnten Basiskomponenten integrieren KSEP zusätzlich Tracking-Systeme zur Verfolgung von Anforderungen, Problembeschreibungen und Fehlerberichten über die gesamte Projektlaufzeit. Diese stellen ebenfalls ein zentrales Koordinations- und Kommunikationsinstrument in vielen OSC dar und substituieren teilweise Forumsbeiträge [5]. Die Tracker werden in erster Linie von den Entwicklern benutzt, wohingegen Foren und Mailinglisten auch von den weniger technisch versierten Anwendern genutzt werden. Beispiele für solche Systeme sind die Open Source-Lösungen Bugzilla[5], Blue-

[1] http://sourceforge.net/
[2] http://javaforge.com/
[3] http://gforge.org/
[4] https://gna.org/projects/savane/
[5] http://www.bugzilla.org/

tail Ticket[6] und Mantis[7].

Sehr große Projekte wie beispielsweise Mozilla benötigen darüber hinaus ein zentrales Konfigurationsmanagement- und Build-System. Dieses ermöglicht eine kontinuierliche Integration von neuen bzw. aktualisierten Modulen sowie automatisierte Kompiliervorgänge und Systemtests. Neu auftretende Fehlermeldungen werden ebenfalls automatisch an die verantwortlichen Entwickler berichtet. Quelloffene Beispiele für solche Systeme sind das Ant-basierte CruiseControl[8] und Tinderbox[9] aus dem Mozilla-Projekt.

Weitere Werkzeuge, die viele Kollaborationsplattformen zur Verfügung stellen, sind grafische Browser-Schnittstellen für das Quelltext-Repository. Diese ermöglichen eine schnelle und systematische Analyse von Projekten mit umfangreichen Quelltexten in unterschiedlichen Modulen. Des Weiteren wird so auch ein später Einstieg in bereits gestartete und komplexe Projekte erleichtert (vgl. Abschnitt 2). Häufig verwendete Code-Browser-Komponenten sind Bonsai[10] aus dem Mozilla-Projekt und ViewCV[11], welches unter anderem von Novel[12] eingesetzt wird.

Wiki-Systeme, die insbesondere durch die freie Enzyklopädie "Wikipedia"[13] sehr populär geworden sind, nehmen auch immer mehr Einzug in die OSSE-Welt. Sie werden hierbei komplementär zu den Mailinglisten, Foren und Tracking-Systemen als zusätzlicher Kommunikationskanal eingesetzt. Wiki-Systeme erlauben es, zusammenhängende Informationen sehr übersichtlich darzustellen und in Zusammenarbeit mit mehreren Personen zu editieren. Andere Ressourcen und Artefakte können über Hyperlinks eingebunden werden. Beispielsweise existiert mit Trac[14] eine sehr schlanke KSEP-Lösung, die lediglich aus Wiki, Subversion und einem Mail-Server besteht.

Zusammenfassend lässt sich festhalten, dass sich aufgrund der ganz besonderen Anforderungen der OSSE-Methodik ein breites und bereits sehr ausgereiftes Spektrum von Werkzeugen zur effektiven Projektkoordination und effizienten Kommunikation in einem verteilten Umfeld entwickelt hat. Die starke Verbreitung von KSEP im Open Source-Bereich hat zudem dazu geführt, dass solche Plattformen immer häufiger auch erfolgreich im Unternehmenskontext übernommen und eingesetzt werden [17].

Die Firma VA Software beispielsweise bietet ihre kostenlose Hosting-Plattform sourceforge.net auch als SourceForge Enterprise für den kommerziellen Einsatz in Unternehmen an. Ein ähnliches Produkt bietet CollabNet mit ihrer Lösung CollabNet Enterprise, wobei diese Plattform in erster Linie als Dienst im Internet angeboten wird ("software as a service"). Während VA Software mittlerweile ebenfalls unterschiedliche Dienstleistungen über das Internet anbietet, wird die CodeBeamer-Plattform der Firma Intland hingegen bisher ausschließlich für Installationen bei Anwenderunternehmen lizenziert.

In funktionaler Hinsicht unterscheiden sich diese drei kommerziellen Produkte kaum. In den aktuellen Versionen werden im Wesentlichen alle oben beschriebenen Koordinations- und Kommunikationswerkzeuge von Kollaborationsplattformen aus dem Open Source-Bereich unterstützt [17]. Allerdings weisen verteilte Softwareprojekte im kommerziellen Umfeld grundlegende strukturelle Unterschiede zu den Gegebenheiten im Open Source-Kontext auf und stellen somit andere Anforderungen an die unterstützende Technologie. Aus diesem Grund sollen diese Unterschiede im folgenden Abschnitt näher beleuchtet und die Übertragbarkeit der Open Source-Entwicklungsmethodik und zusätzliche Anforderungen im Unternehmenskontext analysiert werden.

4 Die Open Source-Entwicklungsmethodik als Modell für verteilte Projekte im Unternehmensumfeld

Bezogen auf die anfangs beschriebene Problemstellung stellen etablierte Methoden und Werkzeuge aus dem Open Source-Kontext grundsätzlich einen Ansatz zur Gestaltung von verteilten Projekten im Unternehmensumfeld dar [5]. Ein Indiz dafür ist unter anderem die immer stärkere Diffusion von KSEP auch im kommerziellen Umfeld [3, 17] sowie die informelle Zusammenarbeit von mehreren Unternehmen im Rahmen von OSC [14, 15]. Aus diesem Grund soll im Folgenden analysiert werden, inwiefern die OSSE-Methodik auf den Unternehmenskontext übertragbar ist.

Der größte Unterschied zwischen OSSE und kommerzieller Softwareentwicklung besteht in einem unterschiedlichen Rollenmodell. Da im Open Source-Umfeld im Gegensatz zur kommerziellen Softwareentwicklung jeder Entwickler auch gleichzeitig als Anwender der von ihm entwickelten Software agiert [19] und es zudem keine klare Rollenverteilung gibt, die sich an einzelnen Phasen des Entwicklungsprozesses orientiert (vgl. Abbildung 1), können im Rahmen der OSSE-Methodik nur zwei grundsätzlich unterschiedliche Rollen identifiziert werden: die Entwickler/Anwender-Rolle und die reine Anwender-Rolle. Während sich der typische Entwickler bei der OSSE neben der reinen Implementierung auch mit dem Testen seiner Software und anderer Programmteile ("Peer Review") beschäftigt, agieren die reinen Anwender, falls sie Fehlerberichte übermitteln, ebenfalls als Tester und werden somit zu Mit-Entwicklern ("Co-Developers") der Software [13].

[6] http://sourceforge.net/projects/btt/
[7] http://www.mantisbt.org
[8] http://cruisecontrol.sourceforge.net/
[9] http://www.mozilla.org/tinderbox.html
[10] http://www.mozilla.org/projects/bonsai/
[11] http://www.viewvc.org/
[12] http://forge.novell.com
[13] http://www.wikipedia.org/
[14] http://projects.edgewall.com/trac/

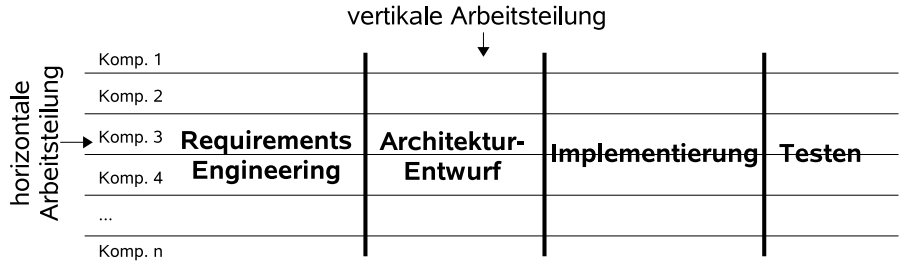

Abbildung 1: Vertikale und horizontale Arbeitsteilung in der Softwareentwicklung

Während sich die OSSE somit durch sehr generalistische Rollen auszeichnet, kommen im Unternehmenskontext oft Spezialisten zum Einsatz. Die einzelnen Rollen orientieren sich hier meist an den jeweiligen Phasen des Softwareentwicklungsprozesses, indem es beispielsweise Spezialisten für das Requirements Engineering (RE), für die Architekturentwurfsphase, für die Implementierung sowie für das Testen gibt. Eine solche Rollenaufteilung kann folglich als "vertikal" bezeichnet werden, indessen sich die Arbeitsteilung im OSS-Umfeld vielmehr an einzelnen Komponenten des Gesamtsystems denn an den Phasen des Softwarelebenszyklus orientiert und somit als "horizontal" bezeichnet werden kann. Dieser grundsätzliche Unterschied von traditioneller Arbeitsteilung und Aufgaben/Rollen-Zuordnung zu der im Open Source-Bereich soll in Abbildung 1 verdeutlicht werden.

Die wesentlichen Implikationen aus der Tatsache, dass im Open Source-Umfeld Entwickler auch Anwender darstellen, betreffen das RE, da hier die Anforderungen von den Anwendern der Software u.a. erhoben, analysiert, spezifiziert und validiert werden müssen [25]. Während in der Anforderungserhebung und -analyse im kommerziellen Umfeld auf die Wünsche der Anwender – sprich Kunden – eingegangen werden muss, indem die Anwender ihre Anforderungen übermitteln und auch priorisieren, werden in der OSSE meist die Wünsche der Entwickler selbst gesammelt und realisiert – hier liegt auch die höhere Motivation von OSS-Entwicklern begründet [6].

Bei der Spezifikationsphase im RE muss in der kommerziellen Softwareentwicklung, insbesondere bei der Erstellung von Unternehmenssoftware, zudem zwischen der Sprache der Fachanwender und jener der Systementwickler "übersetzt" werden. Um so mehr Domänenwissen letztere innehaben, desto weniger bedeutend wird die Spezifikation. Ein Extrem hierbei stellt das Szenario der Offshore-Entwicklung dar, in der die Entwickler neben kulturellen Unterschieden und Verständnisproblemen oft kein bis sehr wenig Fachwissen besitzen und infolgedessen eine sehr genaue Spezifikation benötigt wird (siehe Beispiel in Abschnitt 5). Als Gegenpol kann die originäre OSSE betrachtet werden, in der die Entwickler, wie weiter oben dargelegt, auch Anwender der Software sind und somit über ein sehr hohes Domänenwissen verfügen.

Auch die Validierung der Anforderungen erfolgt in der Regel in engem Diskurs zwischen Anwendern und Entwicklern [23] und verläuft daher im Open Source-Umfeld grundsätzlich unproblematischer. Des Weiteren substituieren die sehr kurzen Release-Zyklen und die inkrementelle prototypenbasierte Validierung im OSSE-Bereich einen großen Teil traditioneller RE-Prozesse. Für eine detaillierte Analyse und einen Vergleich der RE-Prozesse in der OSS-Entwicklung sei hier auf Scacchi [23] verwiesen.

Weitere wesentliche Unterschiede zwischen der OSSE und der Softwareentwicklung im Unternehmenskontext stellen der Umfang und die Art der zu erstellenden Dokumentation und eine lückenlose Rückverfolgbarkeit des Prozesses dar ("Traceability", vgl. [21]). Während in Unternehmen oft sehr genau und viel dokumentiert werden muss und ein systematisches Festhalten von Transformationsbeziehungen zwischen Artefakten ("Traceability Management") benötigt wird, werden sowohl Dokumentation als auch Traceability bei der OSSE bisher vernachlässigt. Anders als bei der kommerziellen Softwareentwicklung existieren im OSSE-Bereich weder die Notwendigkeit noch entsprechende regulatorische Vorschriften.

Folglich können Kollaborationsplattformen (KSEP) nur dann erfolgreich im Unternehmenskontext genutzt werden, wenn die hier identifizierten Defizite der OSSE-Methodik adressiert und behoben werden können. Somit wird es den Herstellern von kommerziellen KSEP obliegen, ihre jeweiligen Plattformen um Konzepte für ein verteiltes RE und ein systematisches Traceability-Management zu ergänzen. Können die hier identifizierten Defizite behoben werden, so steht einem erfolgreichen Einsatz von KSEP und somit einer teilweisen Übertragung der OSSE-Methodik in die kommerzielle Softwareentwicklung nichts mehr im Wege.

Neben den hier genannten Beispielen für den Einsatz von OSSE-Methoden und -Werkzeugen im Unternehmenskontext existieren noch weitere Modelle für die informelle Zusammenarbeit von Unternehmen im Rahmen von regulären Communities, die auch öffentlich zugänglich sind [14, 15].

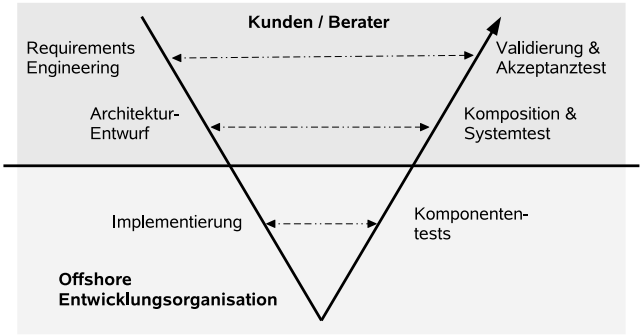

Abbildung 2: Traditionelle vertikale Arbeitsteilung in einem Offshore-Entwicklungsprojekt

5 Szenario "Offshore-Softwareentwicklung"

Um die bisherigen theoretischen Überlegungen zur Übertragbarkeit der OSSE-Methodik an einem Beispiel aus dem Unternehmensumfeld zu erörtern, bietet sich dazu ein global verteiltes "Offshore"-Softwareentwicklungsprojekt an. "Offshore" impliziert dabei sowohl aus europäischer als auch aus nordamerikanischer Perspektive die Zusammenarbeit mit oder die Gründung von Entwicklungsorganisationen im asiatischen Raum, insbesondere Indien [2, 12].

Einige der Gründe, die für eine Auslagerung von Entwicklungstätigkeiten in diese Länder sprechen, wurden anfangs bereits aufgezeigt. In erster Linie führen das immer noch starke Lohngefälle und die flexible Verfügbarkeit von zahlreichen hoch qualifizierten Arbeitskräften zur Durchführung von immer mehr Offshore-Softwareentwicklungsprojekten. Häufig kommen dabei traditionelle sequenzielle Vorgehensmodelle zum Einsatz, die implizit eine Arbeitsteilung entlang des Softwarelebenszyklus vorschlagen (vgl. Abbildung 1). In der Praxis bedeutet dies, dass ein Team beim Kunden die Anforderungen aufnimmt und analysiert und diese Informationen in Form einer Spezifikation komplett an das nächste Team zu Systementwurf und Implementierung übergibt.

Abbildung 2 zeigt ein beispielhaftes Offshore-Entwicklungsprojekt mit traditionell vertikaler Arbeitsteilung zwischen Beratern bei einem industriellen Kunden in Deutschland und einem Entwicklungsteam in Fernost. Das Vorgehensmodell definiert dabei das beratende Unternehmen in seiner Funktion als Projektmanagement-Dienstleister. Das Vorgehensmodell sieht in Anlehnung an das V-Modell neben der Anforderungserhebung und -spezifikation in Form von Use Cases und Architekturmodellen auch abschließende System- und Akzeptanztests auf Seiten des Beratungshauses bzw. beim Kunden vor [8].

Problematisch ist in diesem Beispiel der Übergang vom Architekturentwurf in die Implementierungsphase. Da die Anforderungen ausschließlich in Deutschland erhoben werden und auf dem Fachwissen der dortigen Berater und Kunden beruhen, muss die Software zu Prozessanfang sehr aufwändig spezifiziert werden. Aufgrund einer hohen Wissensasymmetrie zwischen der Beratungs- und der Offshore-Organisation treten an dieser Schnittstelle im Prozess große Informationsverluste auf. Diesem Effekt soll zum einen der Austausch von Mitarbeitern beider Seiten sowie ein erhöhter elektronischer Kommunikationsaufwand entgegenwirken.

Anders als bei diesem entkoppelten Vorgehen arbeiten in typischen OSSE-Projekten einzelne Entwickler oder Entwicklerteams über den gesamten Softwarelebenszyklus parallel an einer oder mehreren Komponenten des zu erstellenden Systems. Außerdem bieten zahlreiche synchrone und asynchrone Kommunikationskanäle die Möglichkeit, sich mit anderen Entwicklern und Anwendern kontinuierlich auszutauschen – was häufig auch sehr exzessiv praktiziert wird. Überdies sind die Methoden der OSSE im Gegensatz zu den traditionellen Vorgehensmodellen bereits in einem ausschließlich verteilten Umfeld entstanden und weiterentwickelt worden (vgl. Abschnitt 2).

Aus diesem Grund wählt die Firma VA Software zur Weiterentwicklung ihrer SourceForge-Plattform[15] eine alternative Vorgehensweise. In Anlehnung an die OSSE-Methodik, zu deren besseren Unterstützung SourceForge ursprünglich konzipiert worden war, wurde für die verteilte Zusammenarbeit mit Entwicklerteams in Chennai, Indien, die Plattform selbst eingesetzt. Entgegen der traditionellen Vorgehensweise wurden die Aufgaben dabei nicht nach Prozessschritten und Ländern verteilt (Abbildung 2) sondern horizontal entlang einzelner Plattformkomponenten in internationalen Teams (vgl. Abbildung 1). Diese Arbeitsteilung ist auch gerade deshalb möglich, da die Entwickler in Fernost über genügend Domänenwissen der Applikation, d.h. Fachwissen bzgl. der SourceForge-Plattform, verfügen und dieses mit jedem Arbeitstag erweitern [7].

Der Einsatz einer zentralen Integrations- und Kommunikationsplattform ermöglicht und unterstützt dabei die virtuellen Teams, die über die Ländergrenzen hinweg an ähnlichen Problemen arbeiten. Innerhalb der einzelnen Unterbereiche herrschen gemeinsame Verantwortlichkeiten und alle Projektbeteiligten werden vom VA-Projektmanagement gleich behandelt, auch wenn die indische Organisation dabei rechtlich selbstständig bleibt. Die gleichberechtigte und eigenverantwortliche Arbeitsweise erfordert jedoch wiederum aufwändige Schulungsmaßnahmen seitens VA Software. Überdies zeigte sich auch hier, dass sich der persönliche Kontakt ("face time") der Entwickler untereinander positiv auf Zufriedenheit und Produktivität auswirkt, auch wenn dies mit zusätzlichen Kosten verbunden ist [7].

Am extremen Beispiel eines Offshore-

[15] http://www.vasoftware.com/sourceforge/

Entwicklungsprojekts lassen sich somit die Vorteile einer an der OSS-Entwicklungsmethodik orientierten Vorgehensweise erkennen. Eine vertikale Arbeitsteilung entlang einzelner Phasen und Länder kann zu hohen Reibungsverlusten bezüglich der Wissensübermittlung und Verzögerungen im Prozess führen. Ein horizontales Vorgehen kann mit entsprechender Unterstützung durch Schulungsmaßnahmen und Kollaborationswerkzeuge diese Verluste minimieren und ein effizientes paralleles Arbeiten ermöglichen. Eine stärkere geografische Verteilung von Entwicklungsteams über mehrere Zeitzonen hinweg könnte zudem in Zukunft eher die Regel darstellen als eine Ausnahme.

6 Schlussbetrachtung

Die Entstehungsgeschichte und die Anwendung der Open Source-Methodik im nicht-kommerziellen Umfeld verdeutlicht, dass es sich um ein funktionierendes Modell für die verteilte, kollaborative Softwareentwicklung in einer nicht-restriktiven Umgebung handelt. Durch die am Markt vorhandenen Kollaborationsplattformen ist es prinzipiell möglich, die Methoden aus dem Open Source-Bereich auch im Unternehmenskontext anzuwenden. Inwieweit aber eine solche Übertragung der Methoden in die restriktive kommerzielle Softwareentwicklung erfolgreich sein kann, hängt von unterschiedlichen Faktoren ab.

Da es sich bei der allgemeinen Open Source-Methodik um eine agile Entwicklungsmethodik handelt, gelten hierbei ähnliche Empfehlungen wie bei anderen agilen Methoden. Wenn die Anforderungen zu Projektbeginn nicht eindeutig spezifiziert werden können, oder davon ausgegangen wird, dass sich die bestehenden Anforderungen im Laufe des Entwicklungsprojekts stark verändern, so ist eine agile Methode zu wählen. So bietet die Open Source-Methodik die Möglichkeit, mittels Kollaborationsplattformen auch im verteilten Umfeld von Unternehmen flexibel auf Veränderungen reagieren zu können.

Durch ein im Vergleich zur kommerziellen Softwareentwicklung stark unterschiedliches Rollenmodell treten bei der Übertragung der Open Source-Methodik auf das kommerzielle Umfeld allerdings einige Schwierigkeiten auf, welche vor allem im RE und in der erhöhten Forderung nach prozessualer Nachvollziehbarkeit (Traceability) anzusiedeln sind. Ein häufig anzutreffender Mangel an Domänenwissen der Entwickler im kommerziellen Umfeld verhindert zudem eine vollständige Übernahme der Open Source-Methodik.

Es ist dennoch auch schon heute möglich, die traditionelle verteilte Softwareentwicklung auf einer breiten Basis durch Softwareentwicklungsplattformen zu unterstützen und zu verbessern. Um eine ganzheitliche Plattform zu bieten, müssen KSEP in Zukunft vor allem die hier identifizierten Defizite im kommerziellen Umfeld ausgleichen, indem verteilte Techniken zur Unterstützung eines systematischen Requirements Engineering sowie eines umfassenden Traceability-Managements entwickelt werden. In diesem Feld werden in den nächsten Monaten sowohl Forscher durch die Entwicklung innovativer Prototypen und Konzepte, als auch kommerzielle KSEP-Hersteller durch die Übertragung der Innovationen in marktfähige Produkte gefordert sein.

Literatur

[1] Abrahamsson, P. ; Salo, O. ; Ronkainen, J. ; Warsta, J.: Agile Software Development Methods - Review and Analysis. In: *VTT Publications* 478 (2002)

[2] Anandasivam, G. ; Sivaramakrishnan, K. ; Krishnan, M. ; Mukhopadhyay, T.: Contracts in Offshore Software Development: An Empirical Analysis. In: *Management Science* 49 (2003), Dezember, Nr. 12, S. 1671–1683

[3] Augustin, L. ; Bressler, D. ; Smith, G.: Accelerating Software Development Through Collaboration. In: *Proc. of the 24th ICSE*. Orlando, Florida, USA : ACM Press, Mai 2002, S. 559–563

[4] Beck, K.: Embracing Change with Extreme Programming. In: *IEEE Computer* 32 (1999), Nr. 10, S. 70–77

[5] In: Behlendorf, B.: *Open Source as a Business Strategy*. O'Reilly, 149–170

[6] Bitzer, J. ; Schrettl, W. ; Schröder, P.: Intrinsic Motivation in Open Source Software Development / Economics Working Paper Archive EconWPA. 2005. – Forschungsbericht

[7] Bodell, C.: *Best Practices for Minimizing the Risks and Hidden Costs of Globally-Sourced Development*. Public Talk at the MIS Research Center, University of Minnesota, USA. `http://misrc.umn.edu/seminars/slides/Best%20Practices%20for%20Managing%20and%20Executing%20Offshore%20Development.pdf`. Version: März 2005

[8] Broy, M. ; Rausch, A.: Das neue V-Modell XT - Ein anpassbares Modell für Software- und System-Engineering. In: *Informatik Spektrum* 28 (2005), Nr. 3, S. 220–229

[9] DiBona, C.: EOF: Why Open Source Wins. In: *Linux Journal* (2004), März, Nr. 119, S. 17

[10] Feller, J. ; B.: A Framework Analysis of the Open Source Software Development Paradigm. In: *Proc. of the 21st International Conference on Information Systems*. Atlanta, GA, USA : Association for Information Systems, 2000, S. 58–69

[11] Feller, J. ; Fitzgerald, B.: *Understanding Open Source Software Development.* Addison-Wesley, 2001

[12] Heeks, R. ; Krishna, S. ; Nicholson, B. ; Sahay, S.: Synching or Sinking: Global Software Outsourcing Relationships. In: *IEEE Software* 18 (2001), März/April, Nr. 2, S. 54–60

[13] Henderson, L.: Requirements Elicitation in Open-Source Programs. In: *CrossTalk - The Journal of Defense Software Engineering* (2000)

[14] Henkel, J.: Software Development in Embedded Linux – Informal Collaboration of Competing Firms. In: *Proc. der 6. Internationalen Tagung Wirtschaftsinformatik.* Dresden, September 2003

[15] Henkel, J.: Open Source Software from Commercial Firms – Tools, Complements, and Collective Invention. In: *Zeitschrift für Betriebswirtschaft* (2004), Nr. 4

[16] Jorgensen, N.: Putting It All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project. In: *Information Systems Journal* 11 (2001), S. 321–336

[17] Koppany, J. ; Lehner, M.: Erfolgreicher Einsatz des Open-Source-Modells in der unternehmensweiten Softwareentwicklung. In: *OBJEKTspektrum* (2005), Nr. 03, S. 36–37

[18] Lerner, J. ; Tirole, J.: Some Simple Economics of Open. In: *Journal of Industrial Economics* 50 (2002), Nr. 2, S. 197–234

[19] Massey, B.: Where Do Open Source Requirements Come From (and What Should We Do About It)? In: *Proc. of the 2nd ICSE Workshop On Open Source Software Engineering* (2002)

[20] Ramesh, B.: Process Knowledge Management with Traceability. In: *IEEE Software* 19 (2002), Mai, Nr. 3, S. 50–52

[21] Ramesh, B. ; Jarke, M.: Toward Reference Models for Requirements Traceability. In: *IEEE Transactions on Software Engineering* 27 (2001), Januar, Nr. 1, S. 58–93

[22] Raymond, E.: The Cathedral and the Bazaar. In: *First Monday* 3 (1998), Nr. 3

[23] Scacchi, W.: Understanding the Requirements For Developing Open Source Software Systems. In: *IEE Proceedings – Software, 149 (1)* (2002), S. 24–39

[24] Sharma, S. ; Sugumaran, V. ; Rajagolpalan, B.: A Framework for Creating Hybrid-Open Source Software Communities. In: *Information Systems Journal* 12 (2002), Januar, Nr. 1, S. 7–25

[25] Sommerville, I.: *Software Engineering.* 7. Auflage. Addison-Wesley, 2004

[26] Warsta, J. ; Abrahamsson, P.: Is Open Source Software Development Essentially an Agile Method? In: *Proc. of the 3rd Workshop on Open Source Software Engineering (ICSE03).* Portland, Oregon, 2003, S. 143–147