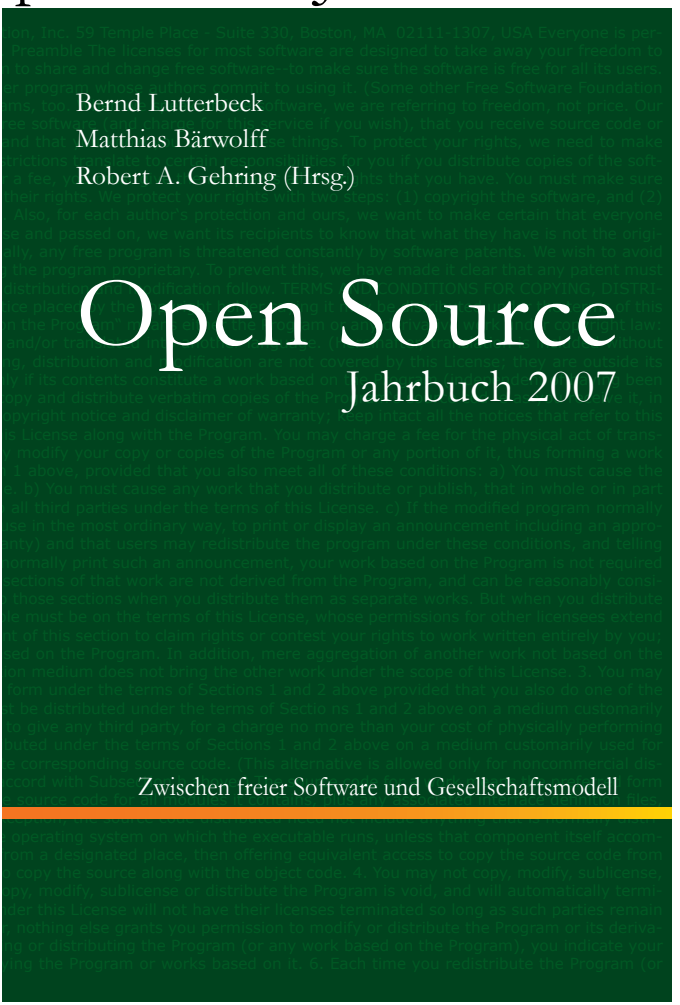
Dieser Artikel ist Teil des

Open Source Jahrbuchs 2007



erhältlich unter www.opensourcejahrbuch.de.

Die komplette Ausgabe enthält viele weitere interessante Artikel. Sie können diesen und andere Artikel im Open-Source-Jahrbuch-Portal kommentieren oder bewerten: www.opensourcejahrbuch.de/portal/. Lob und Kritik sowie weitere Anregungen können Sie uns auch per E-Mail mitteilen.

Einbeziehung von Usability-Experten in Open-Source-Community-Projekte: Erfahrungen aus dem OpenUsability-Projekt

ELLEN REITMAYR

Bis heute verfügen wenige Open-Source-Projekte über eine ausgeprägte Usability-Community[1]. Selbst große Projekte wie die Linux-Desktops *KDE* oder *GNOME* haben nur einen geringen Anteil an fachlich geschulten Usability-Mitwirkenden. In den drei Jahren aktiver Usability-Arbeit im Rahmen des Projektes *OpenUsability*[2] konnten verschiedene Faktoren identifiziert werden, welche die kontinuierliche Zusammenarbeit zwischen Usability und Open-Source-Entwicklung fördern oder behindern können. Diese lassen sich zwei Bereichen zuorden: Zum einen handelt es sich um Voraussetzungen erfolgreicher Usability-Arbeit, die in Open-Source-Projekten oft nicht gegeben sind, zum anderen um strukturelle Eigenschaften der Projekte wie Entscheidungswege oder Integration der Usability-Contributors in die Community. Im folgenden Artikel werden diese Faktoren näher beschrieben und beispielhaft Lösungsansätze aufgezeigt.

Schlüsselwörter: Gebrauchstauglichkeit · Benutzerfreundlichkeit · Usability · Integration · OpenUsability · Community · Open-Source-Projekte

1 Einleitung – Usability im Entwicklungsprozess

„Mit intuitiver Bedienung", „Jetzt noch einfacher", „Einfach wie nie!" – In der Werbung oder in Pressemitteilungen zum Release einer neuen Software gewinnt neben der Funktionalität auch die Einfachheit der Bedienung immer stärker an Bedeutung.

1 Usability (Gebrauchstauglichkeit) ist eine Forschungsrichtung, die Produkte bezüglich ihrer Benutzbarkeit für bestimmte Nutzergruppen und Aufgaben optimiert.

2 Die Plattform *OpenUsability* wurde 2004 gegründet, um Usability und Open-Source-Entwicklung zusammenzubringen: http://www.openusability.org.

Damit ist die Popularität von Usability, also von Methoden, welche die Benutzerfreundlichkeit einer Software erhöhen, in den letzten Jahren stark angestiegen.

Dieser Trend macht auch vor Open-Source-Projekten nicht halt: Mit dem wachsenden Anteil von Open-Source-Software (OSS) auf dem Desktop gilt in vielen Projekten nicht mehr der Grundsatz „von Geeks[1] für Geeks". Die Frage, was der Desktop-Nutzer versteht und was nicht, bleibt jedoch oft reine Spekulation. Um Entscheidungshilfen zu geben, steigt die Nachfrage nach Usability-Experten[2], die User Interfaces nach Nutzungshindernissen durchforsten. Häufig zeigt sich jedoch ein grundsätzliches Missverständnis, was Usability ausmacht. Benutzungsfreundlichkeit wird oft als *yet another feature* wahrgenommen, das einfach zur Liste der Neuerungen beim nächsten Release hinzugefügt werden kann.

Um Software tatsächlich benutzungsfreundlich zu machen, muss Usability jedoch von Anfang an in den Entwicklungsprozess integriert werden. Dieses Prinzip wird in Form der ISO-Norm 13407 *Human-Centred Design Process for Interactive Systems* manifestiert und auch als *usability engineering* bezeichnet. Usability wird hier nicht als punktueller Eingriff definiert, in dem ein bestehendes System auf Nutzungshindernisse überprüft wird, sondern als ein den gesamten Entwicklungs-Zeitraum umspannender Prozess, der mit der Erfassung des Nutzungskontexts und der Definition von Nutzer-Anforderungen noch vor der technischen Planung beginnt. In einem iterativen Prozess werden auf Basis der Nutzer-Anforderungen Design-Entwürfe entwickelt und anhand von Usability-Methoden überprüft. Das Methoden-Repertoire ist an die jeweilige Phase des Entwicklungsprozesses gebunden.[3] Zur Erfassung der Anforderungen werden beispielsweise Aufgabenanalyse oder Personas[4] eingesetzt, in der Phase der Konzeption werden Design-Entwürfe anhand von Papierprototypen verifiziert oder die Informationsstruktur wird mit Hilfe von *Card Sorting* ermittelt. Nutzungshindernisse werden so schon frühzeitig aufgedeckt und können schnell und kostengünstig korrigiert werden.

Die Durchführung einer solchen nutzerorientierten Entwicklung setzt voraus, dass Usability-Spezialisten entscheidend in die Entwicklung einbezogen werden. Ihre Aufgabe ist nicht nur das Design der grafischen Nutzungsoberfläche, sondern auch das Erstellen von Entscheidungshilfen bei der Priorisierung von Funktionalität: Durch Usability-Methoden werden Funktionalitäten identifiziert, die von den Nutzern drin-

1 *Geek* ist ein Anglizismus, der sich aus dem Wort Geck herleiten könnte. Im Zusammenhang mit Computern ist er auch positiv besetzt und bezeichnet hier den engagierten Spezialisten.

2 *Usability-Experte* ist in Deutschland und den meisten Teilen Europas bisher kein Ausbildungsberuf. Die meisten sind Quereinsteiger, die vorher als Informatiker, Ingenieure, Soziologen, Psychologen oder Designer gearbeitet und sich entsprechend weitergebildet haben (Reitmayr et al. 2004). In vielen Studienfächern gibt es heute auch entsprechende Vertiefungen. Für eine Liste an Studienmöglichkeiten siehe Stößel (2005).

3 Für einen Überblick über die Usabilitymethoden im Entwicklungsprozess siehe beispielsweise http://www.usabilitynet.org/tools/methods.htm.

4 Unter einer Persona versteht man daher ein imaginäres Modell einer Person mit allerdings sehr konkreten (Charakter-)Eigenschaften oder etwa Nutzungsverhalten.

gend benötigt werden, und es wird überprüft, wie diese Funktionalitäten am besten in die bestehende Software integriert werden können.

Das Bewusstsein von Usability als *Prozess* fehlt sowohl in der proprietären als auch in der Open-Source-Entwicklung noch zu großen Teilen. In der proprietären Software-Welt ist Usability oft ein externer Dienstleister, der zwar zu bestimmten Zeitpunkten den Pfad der Entwicklung kreuzt, jedoch wenig konzeptionelle Arbeit leistet. Erfreuliche Ausnahmen bilden hier Groß-Firmen wie *Microsoft*[5], *SAP*[6] oder *IBM*[7], die schon seit Jahren *user-centered design* propagieren und durch eigene Abteilungen aktiv betreiben.

2 Usability in Open-Source-Projekten

Bis heute verfügen wenige Open-Source-Projekte über eine ausgeprägte Usability–Community. Zwar haben Nutzer, Entwickler und andere Interessenvertreter die Möglichkeit, Kommentare und Verbesserungswünsche in Usability-Mailinglisten oder -Foren zu diskutieren, doch ist der Anteil der fachlich geschulten Mitwirkenden verschwindend gering. Selbst große Projekte wie die Linux-Desktops *KDE* und *GNOME* verfügen nur über einen geringen Anteil an ausgebildeten Usability-Studenten oder -Professionals.

Das geringe Engagement von Usability-Experten in Open-Source-Projekten scheint auf den ersten Blick verwunderlich: Wie Mühlig (2005) beschrieb, kann die Usability–Arbeit in OSS-Projekten äußerst erfüllend sein. Da individuelle Entwickler oft über hohen Einfluss verfügen, können substanzielle Änderungen aus Usability-Empfehlungen zeitnah und ohne Diskussionen umgesetzt werden – ein Erfolgsgefühl, das man derart in kommerziellen Projekten selten erfährt. Auch eignen sich Open-Source–Projekte sehr gut zur Aus- und Weiterbildung, da selten Grenzen durch Zeitdruck gesetzt sind und keine Vertraulichkeits-Vereinbarungen getroffen werden müssen.

Eventuell sind die Schwellen bei der Annäherung zu hoch: Welches Open-Source–Projekt soll man als Usability-Spezialist unterstützen, wie kann man es ansprechen? Wie kommt man als Open-Source-Projekt an kompetente Usability-Leute? Um diesen Erst-Kontakt zu erleichtern, wurde 2004 die Plattform *OpenUsability*[8] gegründet, die interessierte Open-Source-Projekte und Usability-Spezialisten zusammenbringt.

Die Plattform erlebte seit ihrer Gründung – sowohl auf Seite der Entwickler als auch der Usability-Spezialisten – einen starken Zuspruch, der sich unter anderem in der Zahl an Registrierungen oder Pressemeldungen widerspiegelte. Trotzdem wird bis heute nur ein Bruchteil der rund 170 registrierten Projekte (Stand: Oktober 2006) aktiv betreut. Während hierfür zum Teil Schwächen der Plattform verantwortlich

5 *Microsoft User Interface Design and Development*: http://msdn.microsoft.com/ui/.
6 *SAP Design Guild*: http://www.sapdesignguild.org.
7 *IBM Ease of Use*: http://www-03.ibm.com/easy/.
8 http://www.openusability.org

sind – Usability-Spezialisten werden nicht in ausreichender Weise über interessante Projekte oder offene Jobs informiert – stellte sich zusätzlich heraus, dass unter bestimmten strukturellen Bedingungen Probleme bei der Kooperation zwischen Usability und Open-Source-Entwicklung auftreten, die dazu führen können, dass die Zusammenarbeit frühzeitig im Sande verläuft.

In den drei Jahren aktiver Usability-Arbeit im Rahmen des Projektes *OpenUsability* konnten wir Faktoren identifizieren, welche die fortlaufende Zusammenarbeit zwischen Usability und Entwicklung in Open-Source-Projekten beeinflussen. Diese beschreiben zum einen Voraussetzungen der Usability-Arbeit in Open-Source-Projekten, zum anderen strukturelle Eigenschaften in der Community.

2.1 Voraussetzungen der Usability-Arbeit in Open-Source-Projekten

Ein wesentliches Mantra der Usability ist der Leitsatz „Know your User". Denn ohne Kenntnis, wer eine Software später verwenden wird, welche Anforderungen er hat und welche Aufgaben er mit der Software bestreiten wird, ist eine nutzerorientierte Entwicklung, wie in der Einleitung beschrieben, nicht möglich.

Beschreibung der Zielgruppe

In Open-Source-Projekten ist die Frage der Zielgruppe oft nicht eindeutig geklärt. Fragt man Entwickler, für wen sie Software entwickeln, erhält man selten klare Aussagen. So wird häufig der wenig spezifische Durchschnittsnutzer oder *Jedermann* genannt (z. B. Trillitzsch 2006). Der Gedanke des *freien Quellcodes* wird hier auf die Nutzerschaft übertragen: Was für alle frei zugänglich sein soll, soll auch für alle gleichermaßen zur Nutzung bereitstehen. Außer Acht wird dabei gelassen, dass die vielzitierte „Tante Tillie" (z. B. Raymond 2004) und der „Geek", also der professionelle Softwareentwickler, sehr unterschiedliche Ansprüche an eine Software haben. Für den Usability-Betreuer ist es schwierig, die Software auf den Endnutzer abzustimmen, da je nach Fragestellung die eigenen Bedürfnisse, die „des Powerusers" oder „meiner Oma" angeführt werden.[9]

Auch im GIMP-Projekt[10], einer Software zur Manipulation von Bildern, zeichnete sich dieser Trend ab: Als lange Zeit einzige ernstzunehmende freie Bildbearbeitungs–Software versuchte der *GIMP*, die gesamte Bandbreite an potenziellen Nutzern zu

9 Ein anschauliches Beispiel bietet eine Diskussion über die Usability von Suchdialogen in der Mailingliste *kde-core-devel* unter http://lists.kde.org/?t=114767296400002&r=1&w=2. In insgesamt 41 E-Mails diskutierten die Entwickler über mögliche Verbesserungen des Suchdialogs. Eine Streitfrage bildete die Suche nach regulären Ausdrücken, die von einigen Entwicklern selbst benötigt und damit stark verteidigt wurde. Andere argumentierten mit dem *average user* oder der *grandma*, und eine Stimme behauptete sogar, dass *KDE* gar nicht für Entwickler gemacht werden solle. Die Diskussionen drehten sich im Kreis, da die Beteiligten meist auf ihren Standpunkt beharrten. Welche Implementierung schließlich umgesetzt wurde, bleibt im Thread offen.

10 *GIMP* steht für das *GNU Image Manipulation Program*.

bedienen. Diese reichte von Gelegenheitsnutzern, die beispielsweise die Bildgröße oder den Rote-Augen-Effekt eines Fotos manipulieren wollten, bis hin zum professionellen, künstlerischen Einsatz.

Die fehlende Ausrichtung auf eine definierte Zielgruppe resultierte in einem unkontrollierten Zuwachs an Funktionalität, vor allem im Bereich der Filter und Erweiterungen. Selbst für geübte Nutzer wurde es immer schwieriger, einen Überblick über die Funktionalität zu erhalten. Mehrere Usability-Berater, die im Rahmen des Projektes *OpenUsability* über mögliche Lösungsansätze bezüglich der Usability des *GIMP* diskutierten, drehten sich immer wieder im Kreis, wenn es zu der Frage kam, welche Funktionalitäten eine Überarbeitung benötigten.

In einer Sitzung zur Definition der Zielgruppe des *GIMP* bei der *Libre Graphics Conference* in Lyon 2006 wurde die Zielgruppe schließlich klar auf High-End-Nutzer festgelegt (Joost und Quinet 2006). Diese richtungsweisenden Vorgaben erlauben nun die Priorisierung der Schritte bei der anstehenden Überarbeitung der Informationsarchitektur und des Interaktionsdesigns des *GIMP*.

Repräsentative Nutzung des user-feedback

Aber Usability-Arbeit bedeutet nicht zwangsläufig wie beim *GIMP* eine Neudefinition der bestehenden Nutzergruppe. Oft fehlt in Open-Source-Projekten lediglich eine klare und übersichtliche Erfassung von Nutzer-Bedürfnissen.

Bug-Tracking-Systeme oder User-Foren erlauben zwar prinzipiell jedem Nutzer, seine Bedürfnisse zu kommunizieren, doch werden sie oft nur von einem bestimmten Teil der Gesamt-Nutzerschaft verwendet. Auch fühlen sich Entwickler oft überfordert von der Menge an Einträgen, so dass sie zu vorschnellen Urteilen beim Akzeptieren oder Ablehnen von Anfragen neigen: In einer Analyse der Bugtracking-Systeme verschiedener GNOME-Applikationen verglich Yeats (2006) die Reaktionen von Entwicklern auf Feedback von sogenannten Code-Savvy-Nutzern – also Bug-Reports, die Wissen über die Quellcode-Struktur erkennen ließen – mit Reaktionen auf Bug-Reports ohne Bezug zum Quellcode. Die Kommentare der *normalen Nutzer* wurden meist zugunsten der Code-Savvy-Kommentare vernachlässigt.

Auch in Nutzer-Foren werden *feature requests* von *normalen Nutzern* häufig vorschnell anhand technischer Maßstäbe bewertet (Yeats 2006). Als Resultat greifen *user innovation cycles* (von Hippel 2005) nicht mehr. Oft werden lange nicht alle innovativen Ideen, die aus kontinuierlichen Rückmeldungen von Nutzern der eigenen Software gewonnen werden könnten, auch tatsächlich aufgegriffen. Der Erfolg des Apache-Servers gegenüber proprietären Alternativen ist laut von Hippel beispielsweise maßgeblich auf derartige Feedback-Zyklen zurückzuführen.

Sinnvoller wäre folglich eine Bewertung anhand der Gewichtung der Einträge: Ausschlaggebend sollte sein, ob ein *feature request*, also der Wunsch nach zusätzlicher Funktionalität, von einer Einzelperson gestützt wird oder ob eine Vielzahl an Nutzern dahintersteht.

Um Bewertungen, Meinungen und *feature requests* zusammenzuführen, eignen sich repräsentative Umfragen innerhalb der Nutzer-Basis. Bei zahlreichen Projekten, die auf *OpenUsability* registriert sind, konnten Umfragen helfen, Features zu priorisieren und so die Entwicklung in die Richtung der Nutzer-Bedürfnisse zu lenken, statt sie durch technische Machbarkeit oder Herausforderung leiten zu lassen.

2.2 Strukturelle Eigenschaften der Community

Strukturelle Eigenschaften der Community kommen vor allem in großen Projekten zum Tragen, in denen die Zusammenarbeit nicht mehr ohne ein gewisses Maß an Management und Organisation funktioniert. Hier handelt es sich nicht nur um OSS–spezifische Merkmale. Auch proprietäre Software-Entwicklung kämpft mit ähnlichen Problemen, wenn Kommunikationspfaden und Wissensmanagement beim Wachstum zu wenig Beachtung geschenkt wurde.

Trotzdem treten gewisse strukturelle Merkmale, die Auswirkungen auf die Usability-Arbeit haben, in OSS-Projekten häufiger auf als in proprietären Projekten. Eines dieser Merkmale ist die hierarchische Struktur, die häufig mit dem Erfolg einer Open–Source-Software korreliert (Healy und Schussman 2003).

Auch in der Usability-Arbeit stellte sich dieser Faktor als relevant heraus: Fehlende hierarchische Entscheidungswege stellen in großen Projekten Probleme bei der Zusammenarbeit dar. Möchte ein Usability-Spezialist hier Vorschläge umsetzen, muss er jeden einzelnen Entwickler von den Vorteilen überzeugen, statt eines einzelnen *project leaders*[11].

Dass Entwickler in Open-Source-Projekten normalerweise *das letzte Wort* haben, bezeichnete Patricia Jung (2006) als das „Primat der Programmierer“: Diejenigen, die den Code schreiben, tragen die Entscheidung über Features und Gestaltung des User Interfaces. Dieses „Primat“ ist unproblematisch, so lange die Entwickler gegenüber Usability aufgeschlossen sind und Vorschläge ernsthaft diskutieren und umsetzen. Sind einzelne Entwickler jedoch skeptisch oder gar abgeneigt, so kann eine ganzheitliche Umsetzung von Änderungen am User Interface nicht mehr garantiert werden.

Im Projekt *KDE*, einer Desktop-Umgebung für *Linux*, zeigen sich die Auswirkungen hierarchischer Entscheidungswege und des „Primats der Programmierer“ besonders deutlich. Durch seine Größe und Struktur stellt die Summe der KDE–Teilprojekte ein gutes Vergleichsstudienobjekt dar, da es neben dem *Core* aus einer Vielzahl an kleineren, unabhängigen Software-Projekten besteht. Eine konsistente hierarchische Struktur ist zwischen den einzelnen KDE-Teilprojekten nicht zu beobachten. Während manche Projekte ein strenges Reglement dahingehend aufweisen, welcher Entwickler welchen Code beitragen darf, sind andere offen. Dies hat auch Auswirkungen auf die Möglichkeiten von Usability-Spezialisten und ihre Bereitschaft, sich aktiv an Open-Source-Projekten zu beteiligen.

11 Ein *leader* ist im Sinne von Healy und Schussman (2003) ein Kern-Entwickler, dessen Ansichten respektiert werden und dessen Vorgaben der Rest des Projektes folgt.

In den folgenden Abschnitten werden die Begebenheiten bezüglich der Beobachtungen von Jung (2006) und Healy und Schussman (2003) in einzelnen KDE-Teilprojekten näher beschrieben und der Zusammenhang zur Integration von Usability–Methoden veranschaulicht.

Die Rolle der Usability-Befürworter im Projekt

Zunächst betreuten verschiedene OpenUsability-Berater einzelne KDE-Projekte wie *Kontact*, *Kivio*, *DigiKam* oder *Kuroo*. Jedes dieser Projekte umfasste eine überschaubare Anzahl an Entwicklern, zudem hatten jeweils die einflussreichen Kern-Entwickler – *leader* im Sinne von Healy und Schussman (2003) – um Usability-Beratung gebeten, waren der Thematik gegenüber also aufgeschlossen. Die Zusammenarbeit wurde von beiden Seiten als fruchtbar bewertet.

Anders der Beginn der OpenUsability-Aktivität im Projekt *Amarok*, einem Audio–Player für *KDE*: Hier bat ein einzelner Entwickler um Usability-Beratung, ohne die Kern-Entwickler des Projektes darüber zu informieren. Ein Usability-Spezialist führte bald darauf eine Analyse durch und leitete die Ergebnisse an das Projekt weiter. Die Core-Developer, die ja nicht selbständig um Usability gebeten hatten, fühlten sich durch den Report angegriffen und beachteten ihn nicht weiter.

Ein Grund, der zu dieser enttäuschenden Reaktion führte, war die fehlende Absprache mit den Projektleitern. Wie in der proprietären Software-Entwicklung auch kann Usability nur effektiv umgesetzt werden, wenn die einflussreichen Stellen von der Idee überzeugt sind und Usability aktiv fördern.

Zur Verteidigung der Amarok-Entwickler muss eingeräumt werden, dass der Usability-Spezialist im Vorfeld nicht mit dem Projekt über dessen Ziele und Zielgruppe gesprochen hat: Während der Report generelle Richtlinien berücksichtigte, hatten die Amarok-Entwickler damals eine besondere Vorstellung, wie sich ihre Software verhalten sollte. Eine klare Absprache über diese Ziele vor Beginn der Usability-Aktivität hätte Missverständnisse verhindert. Erfahrungen aus anderen Projekte stützen diese Annahme.

Kontinuierliches Engagement der Usability-Spezialisten

Punktuelle und generelle Reports sind die einfachste Methode, Usability-Feedback an ein Projekt zurückzumelden. Usability-Spezialisten können den Zeitaufwand für das Erstellen des Reports leicht einschätzen, mit etwas Extra-Zeit für die Diskussion ist der Beitrag für sie zügig erledigt. In der Praxis hat sich jedoch gezeigt, dass Entwickler unterschiedliche Prioritäten bei der Implementierung der Vorschläge setzen. Statt sie im Nutzungs-Kontext zu beurteilen, werden *quick fixes* in der Regel zuerst implementiert, selbst wenn im Report andere Prioritäten gesetzt sind. Bei *Kontact*, dem *KDE Personal Information Manager*, führte dies zu Inkonsistenzen in der Menüstruktur

der Sub-Komponenten Mail, Adressbuch und Kalender, da Empfehlungen für einige Teile umgesetzt wurden, für andere nicht.

Insgesamt hat sich gezeigt, dass punktuelle Reports anhand genereller Usability–Heuristiken lediglich in einer Vielzahl an *quick fixes* resultieren. Die häufig notwendigen, umfangreicheren Änderungen am Interaktions-Design oder der Informations–Architektur werden jedoch selten umgesetzt. Auch dies gilt in gleicher Weise für die proprietäre Software-Entwicklung – punktuelle Reports ohne weitere Anstrengungen widersprechen der Idee eines nutzerorientierten Entwicklungsprozesses (vgl. Abschnitt 1).

Dennoch: Punktuelle Reports sind besser als kein Feedback und können als Einstiegspunkt für langfristige Usability-Arbeit dienen. Dass dies häufig nicht der Fall ist, lässt sich wiederum auf das „Primat der Programmierer" (Jung 2006) und die von Yeats (2006) beschriebenen Reaktionen auf nicht-technisches Feedback zurückführen: Beiträge von Usability-Mitwirkenden, die sich innerhalb der Community noch keinen Namen gemacht haben, werden häufig vorschnell mit einem *won't fix* abgetan. Nur hochmotivierte Usability-Spezialisten werden sich dann auf weitere Diskussionen einlassen.

Im KDE-Projekt wurde deshalb versucht, die Usability-Mitwirkenden durch Blogs und Interviews stärker in die Community einzubinden. Auf der einen Seite erhöhte dies die Bindung und damit das kontinuierliche Engagement der Usability-Berater, auf der anderen Seite wurde unter den Entwicklern das Vertrauen in Usability gestärkt. Die Usability-Experten waren keine Außenstehenden mehr, die ungefragt die eigene Software kritisieren, sondern wurden zu gleichwertigen Mitgliedern des Projektes.

Verantwortlichkeiten und Entscheidungswege

Mit steigendem Engagement im Projekt realisierten die Usability-Berater in *KDE* bald, dass sie immer wieder die gleichen Probleme an unterschiedlichen Stellen des Projektes angingen. So wurden ähnliche Interaktions-Elemente benötigt, die jeweils in einem Unterprojekt neu implementiert wurden.[12] Auch erläuterten sie gegenüber einzelnen Programmierern stets die gleichen Usability-Grundsätze, was viel Zeit und Energie kostete.

Es zeigte sich, dass dem KDE-Desktop eine einheitliche Richtung fehlte. Die Usability-Spezialisten sahen sich einer neuen Herausforderung gegenübergestellt: Um Änderungen konsistent in *KDE* umsetzen zu können, wurden allgemeine *guidelines* sowie Änderungen an den Bibliotheken der Desktop-Umgebung notwendig.

Um dies zu erreichen, ist eine enge Zusammenarbeit mit den Entwicklern ausschlaggebend. An den Bibliotheken ist jedoch eine Vielzahl Entwickler beteiligt, kla-

12 Eine optimierte Darstellung der Zoom-Funktion auf dem Toolbar wurde beispielsweise in einem Office-Programm, dem PDF-Viewer und einem weiteren Programm jeweils einzeln implementiert. Sinnvoller wäre eine Manifestierung der Optimierung in den Bibliotheken, doch fand sich hierfür bisher kein Programmierer.

re Verantwortlichkeiten fehlen. Die Implementierung von Usability-Verbesserungen scheiterte somit oft daran, dass die Usability-Berater zunächst selbständig einen Entwickler finden mussten, der Zeit und Fähigkeiten besitzt, die Änderung umzusetzen. Mailinglisten stellten sich als ein wenig geeignetes Medium heraus, um Entwickler zu werben, da Requests oft ungehört im Archiv untergingen.

Um diesen Problemen zu begegnen, wurden Ende 2005 verschiedene Umstrukturierungen in der KDE-Community durchgesetzt.[13] Mit der Schaffung einer technischen Arbeitsgruppe[14] wurde das Problem der fehlenden Verantwortlichkeiten angegangen, so sollten klare Ansprechpartner für bestimmte technische Bereiche geschaffen werden. Für Anfang 2007 sind Job-Listen geplant, auf denen unter anderem Requests des Usability-Teams permanent ausgeschrieben sind. *Human-interface guidelines*[15] als allgemein verbindliche Richtlinien bei der Gestaltung von Anwendungen sind in der Entstehung.

3 Fazit – Integration in die Community

Entscheidet sich eine Community, nicht (nur) für sich *selbst*, sondern für Nutzer mit anderen Bedürfnissen zu entwickeln, kann der Grundsatz *just for fun* nur noch bedingt gelten. Wie in der proprietären Softwareentwicklung ist Usability in Open-Source–Projekten kein Ziel, das man von heute auf morgen umsetzen kann. Die Einführung von *Usability-Engineering* bedeutet eine klare Definition von Zielstellungen, auf welche die Software hin *maßgeschneidert* wird, sowie strukturelle Änderungen, bei denen Usability-Spezialisten zu gleichberechtigten Mitgliedern im Projekt werden.

Insbesondere der letzte Punkt, also die kontinuierliche Integration von Usability in den Open-Source-Entwicklungsprozess, stellt sich immer wieder als problematisch heraus. Wird Usability nicht ausreichend von einflussreichen Entwicklern propagiert und aktiv gefördert, bleiben Verbesserungsvorschläge aus *usability reports* häufig unbeachtet. Werden diese *reports* von Usability-Spezialisten erstellt, die im Projekt bisher unbekannt sind, finden die *reports* leider noch weniger Aufmerksamkeit – als Konsequenz wenden sich diese Usability-Spezialisten dann anderen Aufgaben zu.

Doch selbst, wenn einflussreiche Entwickler Usability befürworten, ist ihre Umsetzung in der Software noch nicht garantiert. Vor allem in Projekten, in denen klare Entscheidungswege und Entwicklungs-Richtlinien fehlen, muss dann bei jedem einzelnen Entwickler ein Bewusstsein für die Anforderungen der Usability vorhanden sein. Eine Verankerung von Usability-Zielstellungen, etwa mittels einer *Vision* oder durch Festschreibung von Richtlinien, sowie *Aufklärungsarbeit* in Blogs, Artikeln

13 Die Änderungen wurden in der jährlichen Vereinssitzung des *KDE e. V.* mehrheitlich beschlossen (siehe http://ev.kde.org/meetings/2005.php).

14 Siehe http://dot.kde.org/1137180087/ und http://dot.kde.org/1139614608/.

15 Siehe http://wiki.openusability.org/guidelines.

oder Workshops zeigte gute Erfolge – und ist somit ein weiterer Schritt hin zu dem Grundsatz der Anwenderorientierung.

Literatur

Healy, K. und Schussman, A. (2003), The Ecology of Open-Source Software Development, Working Paper, Department of Sociology, University of Arizona.

Joost, R. und Quinet, R. (2006), 'The GIMP Developers Conference 2006 – GIMP vision', http://developer.gimp.org/gimpcon/2006/index.html#vision [17. Dez 2006].

Jung, P. (2006), Frauen-freie Zone Open Source?, *in* B. Lutterbeck, M. Bärwolff und R. A. Gehring (Hrsg.), 'Open Source Jahrbuch 2006 – Zwischen Softwareentwicklung und Gesellschaftsmodell', Lehmanns Media, Berlin, S. 235–250. http://www.opensourcejahrbuch.de/download/jb2006/chapter_05/osjb2006-05-03-jung [17. Dez 2006].

Mühlig, J. (2005), Open Source und Usability, *in* B. Lutterbeck, R. A. Gehring und M. Bärwolff (Hrsg.), 'Open Source Jahrbuch 2005 – Zwischen Softwareentwicklung und Gesellschaftsmodell', Lehmanns Media, Berlin, S. 87–94. http://www.opensourcejahrbuch.de/download/jb2005/chapter_02/osjb2005-02-02-muehlig [17. Dez 2006].

Raymond, E. S. (2004), 'The Luxury of Ignorance: An Open-Source Horror Story'. http://www.catb.org/~esr/writings/cups-horror.html [17. Dez 2006].

Reitmayr, E., Vogt, P., Beu, A., Mauch, D. und Röse, K. (2004), Branchenreport und Honorarspiegel 2003, Bericht, German Chapter der Usability Professionals' Association e.V., Stuttgart. http://www.gc-upa.de/files/upa_report_fn.pdf [17. Dez 2006].

Stößel, C. (2005), 'World Usability Day 2005: Informationsbroschüre Studienmöglichkeiten', German Chapter der Usability Professionals' Association e.V. http://www.gc-upa.de/files/UsabilityStudiengaenge.pdf [17. Dez 2006].

Trillitzsch, T. (2006), KDE Target Users: The Various Comments about the Intended KDE Audience. Unpublished Analysis of the 'People behind KDE' Interview Series.

Yeats, D. (2006), Open-Source Software Development and User-Centered Design: A Study of Open-Source Practices and Participants, PhD thesis, Texas Tech University, Lubbock, TX. http://etd.lib.ttu.edu/theses/available/etd-07232006-221754/unrestricted/Yeats_Dave_Diss.pdf [17. Dez 2006].

von Hippel, E. (2005), „Anwender-Innovationsnetzwerke": Hersteller entbehrlich, *in* B. Lutterbeck, R. A. Gehring und M. Bärwolff (Hrsg.), 'Open Source Jahrbuch 2005 – Zwischen Softwareentwicklung und Gesellschaftsmodell', Lehmanns Media, Berlin, S. 449–462. http://www.opensourcejahrbuch.de/download/jb2005/chapter_07/osjb2005-07-04-vonhippel [17. Dez 2006].