Dieser Artikel ist Teil des

Open Source Jahrbuch 2005

erhältlich unter http://www.opensourcejahrbuch.de.

Das Open Source Jahrbuch 2005 enthält neben vielen weiteren interessanten Artikeln ein Glossar und ein Stichwortverzeichnis.

Open Source und Usability

JAN MÜHLIG

Open-Source-Software (OSS) steht im Ruf, wenig nutzungsfreundlich und daher für „normale" Anwender kaum geeignet zu sein. Einige selbsternannte „Beobachter" führen dies vor allem auf das geringe Interesse der nicht–kommerziellen Entwickler zurück, sich mit Ergonomie oder Usability (Nutzungsfreundlichkeit) zu beschäftigen. Meine Erfahrungen mit verschiedenen Open-Source-Projekten führen zu einem differenzierteren Bild. In meinem Aufsatz gehe ich zunächst auf einige Strukturbedingungen der OSS-Entwicklung ein und zeige, wie sie sich auf Usability auswirken – soweit möglich im Vergleich zu kommerzieller Software-Entwicklung. Im Weiteren beschreibe ich die Herausforderungen, wie in Zukunft Benutzungsfreundlichkeit im Open-Source-Bereich umgesetzt und fest etabliert werden kann. Hier spielen zwei Faktoren eine wesentliche Rolle: Die Integration von Usability und Ergonomie in die Praxis der Open-Source-Entwicklung sowie die Bereitstellung und Schulung von Usability-Ressourcen (Experten, Einrichtungen, Wissen). Wenn dies umgesetzt werden kann, dann hat OSS mittelfristig die Chance, Benutzerfreundlichkeit als (Markt-)Vorteil auszuspielen.

1. Einleitung

Der Erfolg und die Verbreitung von Open-Source-Software (OSS) in den vergangenen Jahren gründen sich vor allem auf dem Server- und Backend-Bereich. Auf dem Desktop ist der Erfolg bislang begrenzt. Entscheidungen, wie die der Stadt München zur Migration mit Linux als Desktop-System, verleiten zu der Annahme, dass es nur noch eine Frage der Zeit ist, bis OSS seine Erfolgsgeschichte auch auf dem Desktop fortführt.

Um im Desktop-Bereich mittel- und langfristig konkurrieren zu können, muss OSS benutzerfreundlich sein, und zwar auch für „normale Anwender". Eine Software kann für den Anwender nur soweit nützlich und sinnvoll sein, wie er sie benutzen kann. Die Frage stellt sich also, wie benutzbar bzw. benutzerfreundlich OSS ist. Tatsächlich gibt es bislang kaum vergleichende Untersuchungen, und die von uns, der relevantive AG, im Frühjahr 2003 durchgeführte Usability-Studie (Mühlig et al. 2003) zu Linux[1] im

1 SuSE-Distribution mit KDE-Arbeitsumgebung

administrierten Desktop-Bereich war eine der ersten, die Daten dazu lieferte.[2] Seitdem sind leider kaum weitere Studien dieser Art dazugekommen.

Ebenfalls ist bislang wenig darüber bekannt, welche Rolle Usability in OSS-Projekten spielt. Pionier-Arbeit dazu leisteten Nichols und Twidale (2002) mit ihrem Aufsatz „Usability and Open Source Software", der allerdings vor allem aus den strukturellen Bedingungen Schlüsse zieht. Empirische Erfahrungen sind dagegen noch immer selten.

In diesem Aufsatz werde ich einige der Faktoren, die für oder gegen die Entwicklung von benutzerfreundlicher Software im Open-Source-Bereich sprechen, aufgreifen und – soweit möglich – mit meinen Erfahrungen aus verschiedenen Projekten, insbesondere dem KDE-Projekt, gegenüberstellen.

2. Entwickeln für die Community

Eines der häufigsten Argumente, warum OSS wenig benutzerfreundlich sei – in der Regel ohne konkreten Beleg – ist die traditionelle Ausrichtung an der Community, d. h. an anderen Hackern (Nielsen 2004). Ihr Urteil ist maßgebend für die Qualität einer Software, entsprechend wird sie mehr oder weniger für diese Gruppe als Endanwender geschrieben. Andere Projekte sind daraus entstanden, dass man eine Software entwickelte, die ziemlich genau das tut, was man selbst gerne haben möchte. Kann man dann verlangen, dass diese Software auch für „normale Anwender" benutzerfreundlich sein muss? „Wer andere Bedürfnisse hat, kann ja eine andere Software einsetzen" ist ein häufig gehörtes und durchaus plausibles Argument.

Diese Einstellung wird jedoch fragwürdig, wenn man Interesse daran hat, dass die eigene Software auch die Wahl für einen Behörden-Desktop oder Windows-Umsteiger ist. Solche Nutzer haben andere Ansprüche als die Entwickler. Deren Rationalität zielt nicht darauf herzustellen, was man möchte, sondern einzusetzen, was man braucht. Will man also Software für durchschnittliche Nutzer einfacher benutzbar machen, dann kann man sie nicht nur für sich selbst schreiben.

Damit tut sich eine sehr wichtige Konfliktlinie auf: Programmiert man für sich und andere OSS-Entwickler oder für „durchschnittliche" Nutzer? Dass sich beides häufig ausschließt, zeigt sich z. B. bei der Funktionsvielfalt. Die meiste OSS hat einen reichen Schatz an Funktionalität, der natürlich für die speziellen Bedürfnisse der Entwickler durchaus passt. So gibt es im Mailprogramm KMail die Einstellung, dass ein Ordner eine Mailingliste beinhalten kann. Diese Funktion ist praktisch für die Verwaltung von Mailinglisten. Ein Entwickler fand dieses Feature sinnvoll, er programmierte es, und es wurde in die Software aufgenommen. Die Zahl derer außerhalb der Hacker-Community, die dieses Feature verwenden, dürfte dagegen sehr klein sein. Trotzdem steht es auf einer Ebene mit z. B. Ordner-Namen oder Icons. Für den Nutzer wird diese Funktionsvielfalt schwierig, weil er nicht sofort unterscheiden kann, was wichtig für

2 Die Studie untersucht, wie leicht typische Aufgaben im beruflichen Einsatz unter Linux bzw. Windows XP erledigt werden können. Dabei wurden 60 (an Linux) bzw. 20 (an Windows XP) Personen getestet. Der Ergebnis-Report ist kostenlos unter http://www.relevantive.de/Linux-Usabilitystudie.html verfügbar.

ihn ist, und was nicht. Heißt das aber, dass man nur noch jene Funktionen anbietet, die für den „normalen" Nutzer relevant sind? Auf welche Funktionen verzichtet man? Was sagen jene Entwickler, die diese Funktion programmiert haben? Was sagen jene Nutzer, die bisher die Software genau deswegen einsetzen, weil es diese Funktion gibt? Diese Konflikte traten z. B. auf, als ich Menü-Dialoge von KMail konzeptionell überarbeitete, um sie leichter benutzbar zu machen. Die „Orchideen"-Features herauszunehmen oder zumindest zu verstecken, würde die Übersichtlichkeit verbessern und die schnelle Erfassung der zentralen Funktionen erhöhen. Auf der anderen Seite stößt man Entwickler vor den Kopf. Ob sich ein Projekt dafür entscheidet, die eigenen Wünsche und die der Community zurückzustellen, um „massenkompatibler" zu werden wird sicherlich in Zukunft an Bedeutung zunehmen. Vielleicht sind im Einzelfall Kompromisse möglich, mit denen es sich leben lässt. Doch ganz auflösen lässt sich dieser Konflikt nicht.

3. Usability ist einfach?

In einem sehr kontrovers diskutierten Aufsatz lässt sich der wichtige OSS-Vorkämpfer Raymond (2004) über die mangelhafte Benutzbarkeit der CUPS-Druckerkonfiguration aus, die es selbst ihm sehr schwer machte, einen Netzwerkdrucker in Betrieb zu nehmen. Raymond greift die Entwickler für ihre Nachlässigkeit an und resümiert, sie müssten nur „Tante Tillie"[3] vor Augen haben, dann würden sie die Software schon so gestalten, dass erstere sie leicht bedienen kann: Usability sei einfach.

Diese Einstellung ist sehr weit verbreitet, auch in der kommerziellen Softwareentwicklung. Tatsächlich wird dabei ein grundsätzliches Missverständnis darüber, was Usability bedeutet, sichtbar, denn: Woher weiß ich, was Tante Tillie braucht? Habe ich sie gefragt? Habe ich sie dabei beobachtet, wie sie eine Software einsetzt? Weiß ich, warum sie etwas so und so tut und nicht anders? Weiß ich, wie sie welche Begriffe versteht? Vermutlich nicht. Stattdessen wird eine Vorstellung von der Nutzungswelt einer fiktiven Tante erstellt, die zudem das „untere Ende" aller gewünschten Nutzer darstellen soll. Gruber (2004) weist vollkommen zurecht auf die Geringschätzung für die „dumb users" hin, die daraus spricht. OSS scheint für eine solche Haltung empfänglicher zu sein, weil sie ihren Erfolg im Zweifelsfall nicht von einem „Markt" abhängig machen muss, wie dies für kommerzielle Software der Fall ist.

Usability ist (leider) nicht so trivial, als dass man nur an den richtigen Nutzer denken müsste. Wäre das der Fall, dann würden wir vermutlich in einer Welt leben, in der Software „verschwindet", weil sie sich so nahtlos in unsere Bedürfnisse einfügt, dass wir sie gar nicht mehr wahrnehmen. Usability ist aber ein durchaus aufwändiger Prozess, im Verlauf dessen die Software *so* an die Nutzer angepasst wird, dass sie intuitiv, erfolgreich, effizient und angenehm anzuwenden ist. In der klassischen Softwareentwicklung wird dieses Ziel dadurch zu erreichen versucht, dass man die Anforderungen der zukünftigen Nutzer erhebt und bestimmt (Requirements), dann schrittweise die Software entwickelt und z. B. durch Prototypen wiederholt an der

3 Gemeint ist der berühmte „dümmste anzunehmende User" kurz DAU.

„Wirklichkeit", also Nutzern, überprüft. Dieses Verfahren ist aufwändig und teuer, benötigt Usability-Spezialisten und muss nicht zuletzt vom Projektmanagement gewollt sein. Andernfalls ist es auch in kommerzieller Softwareentwicklung lediglich ein Lippenbekenntnis. OSS-Projekte können für sich entscheiden, ob sie für durchschnittliche Nutzer geeignet sein sollen oder nicht. Entscheiden sie sich dafür, müssen sie genauso Verfahren entwickeln und Ressourcen aufbauen, um dieses Ziel zu erreichen.

4. OSS-Usability = Bazaar?

Das Mitwirken in OSS-Projekten wurde von Raymond (1998) in seinem berühmten Aufsatz „The Cathedral and the Bazaar" mit den Mechanismen eines Bazars beschrieben. OSS-Entwicklung zeichnet sich durch offene Kommunikationsstrukturen aus: Es ist leicht, mit den Entwicklern in Kontakt zu treten und Feedback einzubringen. Eigentlich müsste die Open-Source-Community sehr gute Voraussetzungen dafür haben, durch einen Austausch mit den Nutzern zu benutzerfreundlicher Software zu gelangen. Kann Usability-Kontribution gleichermaßen funktionieren? Gibt es wesentliche Unterschiede zwischen Usability-Kontribution und Code-Entwicklung, die dagegensprechen?

Die Maintainer eines Projektes können hervorragend abschätzen, was guter Code ist und was nicht, auch wenn sie den Entwickler nicht kennen. Das gleiche gilt für Bugs. Sie sind (im Regelfall) objektiv reproduzierbar und entweder tatsächlich ein Bug oder eben nicht.

Das sieht bei Usability-Beiträgen anders aus. Angenommen, jemand schickt den Maintainern des Projektes X einen Problembericht und einen Vorschlag zur Verbesserung. Woher wissen die Maintainer, dass es sich tatsächlich um ein Problem handelt? Woher wissen sie, dass der Vorschlag eine Verbesserung ist? Für wen?

Usability-Kontribution ist daher vermutlich nicht einfach mit Code vergleichbar. Die einzige wirklich seriöse Grundlage für Aussagen sind Tests oder Beobachtungen mit „echten" Nutzern. Auch Aussagen, die aufgrund von früheren Tests oder vergleichbaren, verifizierten Situationen stammen, sind ausreichend, wenn sie vorsichtig angewendet werden. Relativ einfach ist schließlich die Überprüfung auf Konsistenz oder Norm- bzw. Guidelines-Konformität („heuristic evaluation").

Ohne zumindest eine dieser Grundlagen ist Usability reine Spekulation oder schlicht Wichtigmacherei. Wenn also – wie in vielen Foren und Mailinglisten der Fall – kein Teilnehmer der Community diese Basis hat, kann das Ziel einer verbesserten Benutzbarkeit für „durchschnittliche Nutzer" kaum erreicht werden. Leider herrschen dort meist persönliche Meinungen oder Projektionen vor.

Daher erscheint es mir gegenwärtig unwahrscheinlich, dass Usability im OSS–Bereich das gleiche „Wunder" des Bazars erfährt, bei dem alle gleichzeitig sprechen und auf magische Weise das Richtige herauskommt. Wohl ist eher das Gegenteil der Fall. In der Vielzahl der Stimmen kann der Maintainer kaum entscheiden, was er nun berücksichtigen soll und was nicht. Dieser Umstand kann sich erst dann ändern, wenn wirklich auf Grundlage von Tatsachen argumentiert wird und diese tatsachenbasierten

Diskussionen eine kritische Masse erreicht haben, auch um einen Standard zu setzen. Gegenwärtig ist dieser Stand nicht erreicht.

Kommerzielle Software hat hier andere Voraussetzungen: Die Entscheidung, welche Probleme repräsentativ für die Zielgruppe sind, wird von Usability-Experten übernommen, nicht von Entwicklern. Aufgrund von Aufwandsabschätzungen durch die Entwickler entscheidet das Produktmanagement über die Umsetzung.

Die klassischen Feedback-Kanäle sind darüber hinaus gerade für „normale" Anwender schwierig zu benutzen: Sie sind mit IRC und Bug-Tracking-Systemen auf technisch versierte Nutzer ausgerichtet, filtern jene, die diese Voraussetzung nicht erfüllen, schlichtweg aus. Doch selbst wenn hier einfache Schnittstellen zwischen Anwendern und Entwicklern geschaffen würden, so wären sie kaum sinnvoll für die OSS-Projekte nutzbar. Wenn tausende von Nutzern „ihr Problem" oder „ihre Lösung" einreichen, bliebe kaum mehr Zeit zum Coden. Die Wirkung wäre einer DDoS-Attacke vergleichbar. Und das Problem des „wer hat jetzt Recht" bleibt auch hier schwierig zu entscheiden, insbesondere wenn eine klare Zielgruppen-Definition fehlt.

5. Fehlende Ressourcen

Aus dem bislang Beschriebenen wird deutlich, dass OSS-Projekte ein zentrales Problem haben: Es fehlen verfügbare Usability-Ressourcen, die zu einer verbesserten Benutzbarkeit für durchschnittliche Nutzer führen. Die Community besteht traditionell aus Programmierern, hingegen fehlen Usability-Experten fast vollständig. Selbst sehr große Desktop-Projekte, wie z. B. KDE, haben nur zwei bis drei Mitglieder, die eine profunde Kenntnis von Usability haben. Natürlich gibt es die Ausnahme, dass Firmen mit den eigenen Experten ein OSS-Projekt unterstützen (z. B. OpenOffice und Sun). Dahinter steckt jedoch ein kommerzielles Interesse, das nicht immer auf Gegenliebe stößt. Und der oben beschriebene Konflikt, ob man für die Community entwickelt oder für eine unbekannte Masse, wird dadurch eher verstärkt.

6. Geschwindigkeit als zentraler Vorteil

Bisher habe ich vor allem von den Schwierigkeiten für Open-Source-Projekte gesprochen, benutzerfreundliche Software zu produzieren. Es gibt jedoch einige Punkte, die OSS gerade gegenüber der klassischen Software-Entwicklung einen klaren Vorteil verschaffen. Der erste und vielleicht wichtigste Punkt ist das Prinzip „veröffentliche frühzeitig und häufig". Professionelles Usability-Engineering bezieht einen Teil seiner Stärke daraus, dass frühzeitig im Entwicklungsprozess Nutzer mit der Software konfrontiert werden können. Je früher diese Tests durchgeführt werden, desto leichter und kostengünstiger sind Veränderungen und Anpassungen. Deshalb werden Prototypen eingesetzt, die allerdings nicht immer so früh bereitstehen, wie es nötig wäre. Für OSS ist fortlaufende Veröffentlichung von inkrementellen Prototypen schlicht Teil des Prozesses – ein Traum-Zustand für jeden Usability-Experten! Durch die häufigen Releases wird es auch möglich, Verbesserungen nach und nach einzubauen. Klassische

Software muss sich hingegen auf große Releases konzentrieren, die keine Nachbesserungen mehr erlauben. Damit hat OSS einen strukturellen Vorteil, der potentiell zu besserer und einfacher zu bedienender Software führen kann. Diesen Vorteil gilt es zu nutzen, doch setzt er Usabilty-Ressourcen voraus.

7. OpenUsability

In der OSS-Entwicklung fehlt es bislang an Usability-Experten, und es fehlen erprobte Verfahren, diese einzubeziehen. Auf der Seite der potentiellen Usability-Kontributoren fehlt das Wissen, wie OSS-Entwicklung funktioniert, was getan werden kann und wie der Input gestaltet sein muss. Auf der Entwicklerseite ist unklar, was sie erwarten können, was gebraucht wird und ob den Vorschlägen der (häufig selbsternannten) Usability-Experten vertraut werden kann. Die Konsequenz ist beispielsweise, dass Anfragen in Mailinglisten oder per E-Mail unbeantwortet bleiben und die Sache im Sand verläuft.

Dies zu ändern ist eines der zentralen Anliegen des Projektes „OpenUsability". Die Idee dazu entstand auf der KDE-Entwickler-Konferenz 2003 in Nove Hrady, wo das Interesse an und die Bereitschaft zur Umsetzung von Usability sehr groß war, aber ein konkreter Ansatz, wie dies in der Praxis aussehen könnte, fehlte. Daraufhin wurde ein allen OSS-Projekten offenstehendes Portal konzipiert und entwickelt, das die Kommunikation und den Austausch zwischen Entwicklern und (externen) Usability-Experten ermöglicht.[4] Dazu gehören verschiedene Konzepte, die für beide Seiten zu einem annehmbaren Prozess führen sollen. So kann ein Projekt sich darin präsentieren und damit die Bereitschaft zeigen, Usability-Input umzusetzen. Umgekehrt können Usability-Experten direkt mit den verantwortlichen Projektmitgliedern in Kontakt treten und sich ein Bild über den Stand der Dinge verschaffen. Schließlich stehen zahlreiche *HOWTOs* zur Verfügung, die in die Besonderheiten der OSS-Entwicklung einführen und damit helfen, falsche Erwartungen zu vermeiden. Verschiedene Werkzeuge helfen, die Zusammenarbeit zu koordinieren und einen funktionierenden *Workflow* zu erreichen. Letzteres wurde insbesondere mit dem Projekt KDE-PIM (KDE Personal Information Management Tool) mehrfach durchgespielt und auf Praxistauglichkeit überprüft.

8. Neue Ressourcen

Gegenwärtig ist nicht eindeutig abzuschätzen, ob und in welchem Maße hier eine – entsprechend der Open-Source-Programmierung – freiwillige und unentgeltliche Unterstützung stattfinden wird. Jedoch hat „OpenUsability" bereits großes positives Feedback erhalten – und das, obwohl das Portal noch nicht fertiggestellt ist. Es scheint tatsächlich, als ob teilweise einfach die Schnittstelle gefehlt hat. Wenn „OpenUsability" diese Lücke füllen kann, wäre ein wichtiger Schritt getan. Es bleibt hoffentlich nicht

4 „OpenUsability" wurde von der relevantive AG als non-profit-Projekt konzipiert und entwickelt. Es ist quelloffen und wird voraussichtlich in Kürze in einen Verein übergehen http://www.openusability.org.

das einzige Projekt dieser Art. Unsere Hoffnung liegt darüber hinaus auf Universitäten und anderen Ausbildungseinrichtungen. So ist es vorstellbar, dass in den (viel zu wenigen) Kursen zu Ergonomie oder Usability anhand von Open-Source-Software gelehrt wird und Praxis-Projekte zur tatsächlichen Verbesserung von Software beitragen.

9. Die Zukunft

Open Source hat spezifische Nachteile bezüglich Benutzerfreundlichkeit. Dazu gehört insbesondere die traditionelle Orientierung an einer sehr Computer-affinen Nutzergruppe und damit der Vernachlässigung von „normalen Nutzern". Auch die starke Bevorzugung von Funktionalität gegenüber Zugänglichkeit und Benutzbarkeit gehört dazu. Damit einher geht die bisher schwache Einbindung von Usability-Experten und als Konsequenz die geringe Verfügbarkeit von Usability-Ressourcen. Es gibt jedoch starke Anzeichen dafür, dass sich dies ändert bzw. ändern kann. Das Interesse im KDE-Projekt an Usability und die Schritte, die zur Verbesserung der Software unternommen werden, sind sehr ermutigend. Nimmt Usability allgemein einen stärkeren Platz in der OSS-Entwicklung ein, kann es seine Stärken, insbesondere die schnelle, im Ansatz prototypische Entwicklung, voll ausspielen.

Unter diesen Voraussetzungen halte ich es für möglich, dass OSS mittelfristig gerade durch Benutzerfreundlichkeit seinen Siegeszug auf dem Desktop begründet.

Literatur

Gruber, J. (2004), 'Ronco-Spray on Usability', http://daringfireball.net/2004/04/spray_on_usability.

Mühlig, J., Horstmann, J., Brucherseifer, E. und Ackermann, R. (2003), Linux Usability Studie, techn. Bericht, relevantive AG. http://www.relevantive.de/Linux-Usabilitystudie.html.

Nichols, D. M. und Twidale, M. B. (2002), Usability and Open Source Software. Working Paper 02/10, Department of Computer Science, University of Waikato, New Zealand. http://www.cs.waikato.ac.nz/~daven/docs/oss-wp.html.

Nielsen, J. (2004), 'Developer Spotlight: Jakob Nielsen', Builder AU http://www.builderau.com.au/webdev/0,39024680,39130602,00.htm.

Raymond, E. S. (1998), 'The Cathedral and the Bazaar', *First Monday* **3**(3). http://firstmonday.org/issues/issue3_3/raymond/.

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