Seminar „Open Source Software Engineering"

Wintersemester 2008/2009

Reimplementation in Gentoo

Betreuer: Christopher Oezbek

02.03.2009

Zusammenfassung

Dieser Seminarbeitrag behandelt zwei wissenschaftliche Artikel über Gentoo Linux. Die beiden Autoren der Artikel – Thomas Østerlie und Letizia Jaccheri – werden kurz vorgestellt. Danach werden beide Artikel erläutert und die Ergebnisse beschrieben. Beim ersten Artikel handelt es sich um eine empirische Studie der Linux-Distribution Gentoo. Østerlie analysiert darin die Verteilung der Kontrolle innerhalb des Gentoo-Projektes mit Hilfe der Actor network theory. Der zweite Artikel ist eine Fallstudie über das Neuschreiben des Gentoo-Paketmanagers, welche Østerlie und Jaccheri gemeinsam durchgeführt haben.

Inhaltsverzeichnis

1 Einleitung

Dieser Artikel ist Teil des Seminars „Open Source Software Engineering"[1] im Wintersemester 2008/2009 am Institut für Informatik der FU-Berlin. Das Seminar beschäftigt sich mit den Entwicklungsmodellen von freier Software. Dieser Seminarbeitrag behandelt zwei wissenschaftliche Artikel über Gentoo Linux. Die beiden Autoren der Artikel – Thomas Østerlie und Letizia Jaccheri – werden kurz vorgestellt. Danach werden beide Artikel erläutert und die Ergebnisse beschrieben.

1.1 Thomas Østerlie

Thomas Østerlie wurde 1973 im norwegischen Trondheim geboren. Er studierte am Institut für Computertechnik und Informationswissenschaften (*Institutt for datateknikk og informasjonsvitenska*p, IDI)[2] der Technisch-Naturwissenschaftlichen Universität Norwegens (*Norges teknisk-naturvitenskapelige universitet*, NTNU)[3]. Die NTNU ist Norwegens zweitgrößte Universität mit 20.000 Studenten und einem jährlichen Budget von umge-

[1] https://www.inf.fu-berlin.de/w/SE/SeminarOpenSource2008
[2] http://www.idi.ntnu.no/
[3] http://www.ntnu.no/

rechnet rund 500 Millionen Euro. Im Jahr 2003 erhielt Østerlie den Abschluss Master of Science, anschließend promovierte er. Nebenbei arbeitete er als Forscher am IDI, sowie als IT-Berater in einem privaten Unternehmen. Schwerpunkte seiner Arbeit dort waren Unix-Server und Computersicherheit. Er veröffentlichte zahlreiche wissenschaftliche Artikel, von denen viele auf seiner Webseite[4] heruntergeladen werden können. Des weiteren schrieb er Begutachtungen zu technischen Büchern, beispielsweise zu *Hacking Linux Exposed*[5]. Außerdem war bzw. ist er an der Entwicklung einiger freier Programme beteiligt. Seine Arbeit beschreibt er selber mit den Worten:

> „I study how software systems are developed; not how we ought to develop software [...]"

1.2 Letizia Jaccheri

Letizia Jaccheri wurde 1965 in Pisa geboren. Sie studierte an der dortigen Universität und erreichte 1988 den Abschluss als Master of Science. Anschließend arbeitete sie einige Zeit am Polytechnikum Turin (*Politecnico di Torino*)[6], wo sie 1994 promovierte. Seit 1997 ist sie an der NTNU tätig und wurde dort 2002 Professorin für Softwaretechnik. In den vergangenen Jahren hat sie über 85 wissenschaftliche Artikel veröffentlicht, darunter einige in Zusammenarbeit mit Thomas Østerlie. Die meisten Artikel können kostenlos von ihrer Homepage[7] heruntergeladen werden.

2 Verteilte Kontrolle in Gentoo Linux

Der vollständige Titel des ersten Artikels lautet „In the network: Distributed control in Gentoo Linux"[Oes04]. Es handelt sich um eine empirische Studie der Linux-Distribution Gentoo. Østerlie analysiert darin die Verteilung der Kontrolle innerhalb des Gentoo-Projektes mit Hilfe der Actor network theory. Als Kontrolle wird von ihm dabei die Macht verstanden, ein Problem zu definieren und die angemessene Lösung festzulegen. Bei zentralisierter Kontrolle kann dies durch hierarchische Autoritäten und die Delegation von Aufträgen erreicht werden. Wie dies aber

[4] http://www.idi.ntnu.no/~thomasos/
[5] http://www.linuxjournal.com/article/4836
[6] http://www.polito.it/
[7] http://www.idi.ntnu.no/~letizia/

bei verteilter Kontrolle funktioniert, ist nicht offensichtlich. Auch von Raymond wird dies in seinem Artikel „Die Kathedrale und der Basar“[Ray99] nicht erklärt. Daher stellt sich Østerlie zu Beginn die Frage, ob die Kontrolle bei allen Open Source Projekten zentralisiert sei.

Im Folgenden werden *Gentoo Linux* und *Actor network theory* kurz vorgestellt, danach wird der Ablauf der eigentlichen Studie geschildert.

2.1 Gentoo Linux

Gentoo Linux[8] ist eine Linux-Distribution, welche von der *Gentoo Foundation, Inc.* entwickelt wird. Im Unterschied zu vielen anderen Linux-Distributionen richtet sie sich an fortgeschrittene Anwender. Eine Besonderheit ist, dass Programme üblicherweise nicht in binärer Form heruntergeladen werden, sondern als Quelltext, welcher dann vom Anwender kompiliert wird. Dieser Vorgang kann bei großen Anwendungen sehr lange dauern. Als Vorteil sehen die Benutzer jedoch die hohe Flexibilität. So können Programme individuell auf ihre Bedürfnisse hin konfiguriert werden. Auch sind selber kompilierte Programme in der Regel etwas schneller, da sie beim Kompilieren für die Hardware, auf der sie anschließend ausgeführt werden, optimiert werden.

Ein weiterer Unterschied von Gentoo im Vergleich mit anderen Linux-Distributionen ist die Veröffentlichung neuer Versionen. So werden von Gentoo keine zyklischen Releases veröffentlicht. Stattdessen findet eine kontinuierliche Entwicklung statt. Eine Gentoo-Version ist daher nur eine Abbildung des momentanen Entwicklungsstandes. Durch die hohe Flexibilität dient Gentoo auch als Grundlage für andere Distributionen.

2.2 Actor network theory

Die Actor network theory (Akteur-Netzwerk-Theorie, ANT) stammt ursprünglich aus den Gesellschaftswissenschaften. Sie wurde in den 80er Jahren entwickelt und ist besonders im englischsprachigen Raum verbreitet. Sie dient der Analyse der Beziehungen zwischen Gesellschaft und Technik, oder allgemeiner ausgedrückt, zwischen verschiedenen Akteuren. Das Grundkonzept sind Netzwerke, bestehend aus Knoten und Kanten. Die Akteure bilden die Knoten, die über verschiedene Assoziationen miteinander verbunden sind, um deren Interaktion darzustellen. Akteure sind „alle Entitäten, die Dinge machen“, also auch Gegenstände, Maschinen, Geld, Nahrung oder Bakterien. Es herrscht Symmetrie zwischen allen

[8] http://www.gentoo.org/

Akteuren. Diese basiert auf der Annahme, dass alle Entitäten aufeinander einwirken können. Akteure wollen Ziele erreichen, daher versuchen sie andere Akteure für ihre Interessen zu gewinnen. Die Größe eines Akteurs hängt von der Größe des Netzwerkes ab, welches er beherrscht. Ein Netzwerk befindet sich im ständigen Wandel und bildet keinen starren Zustand ab.

Ein Beispiel für eine Betrachtung im Sinne von ANT sind Wissenschaftler, welche ihre Reputation und den Einfluss in ihrem Labor erhöhen wollen. Dazu etablieren sie Netzwerke, sie bewegen sich in Akteursnetzen mit vielen Personen, Materialien und Techniken. Diese von ihnen benutzte Infrastruktur ist wesentlich in den Beziehungen zwischen den Akteuren, da sie ohne diese keinen Erfolg haben werden.

Die Actor network theory ist nicht unumstritten. Ein Kritikpunkt ist die Gleichheit von Mensch und Technik, die den Kritikern widersprüchlich erscheint. So kann es ohne den Menschen keine Technik geben, auch kann Technik im Gegensatz zu Menschen kein Befehl erteilt werden. Des weiteren wird bezweifelt, ob Technik überhaupt Interessen haben kann.

2.3 Ablauf

Für die Durchführung der Studie sammelte Østerlie Daten aus verschiedenen Quellen. Neben der Gentoo-Website wurden auch die Archive der Mailingliste und IRC-Protokolle verwendet. Des weiteren nahm er auch selber für einige Zeit aktiv am Projekt teil, führte dabei Beobachtungen durch und sprach mit einem der Gentoo-Entwickler.

Seine Studie basiert im wesentlichen auf einem Treffen der Hauptentwickler im Dezember 2003. Zum damaligen Zeitpunkt wurde ein solches Treffen im IRC alle zwei Wochen durchgeführt als Teil der Organisation innerhalb des Projektes. Innerhalb dieses Treffens wurde von den Entwicklern ein Problem mit dem Paketmanager Portage angesprochen, einem der wesentlichen Bestandteile von Gentoo. Es gab Probleme dadurch, dass mehrere externe Programme direkt in den Datenbanken und Konfigurationsdateien von Portage schrieben. Da sich mittlerweile das Format dieser Konfigurationsdateien geändert hatte, kam es zu Problemen mit veraltetem Quellcode. Als Lösung sollte eine API entwickelt werden, durch die Zugriffe auf Konfigurationsdateien künftig erfolgen sollten. Zwei Entwickler wurden diesem Problem zugeordnet.

Das Problem wurde von den Entwicklern untersucht, wobei sich herausstellte, dass für bestimmte Codeteile kein Maintainer mehr existierte. Grund dafür war, dass die ehemals zuständigen Entwickler das Projekt

verlassen hatten und der Code unbeaufsichtig zurückgeblieben war. Weitere Probleme ergaben sich durch die Tatsache, dass es mehrere Tools von überlappender Funktionalität gab, von denen jedoch keines seine Arbeit wirklich gut erfüllen konnte. Die vielen offenen Bugs wurden von einem Entwickler mit den Worten beschrieben:

> „We don't need five half-working use flag editors. We need one really good one"

Die Frage, welche Probleme Priorität erhalten, entspricht der Frage nach der Kontrolle innerhalb des Projektes. Der Chef-Entwickler schlug eine API vor, basierend auf dem bisherigen Portage Code. Die beiden zugeordneten Entwickler hingegen sahen andere Probleme als dringlicher an. Dieser Ablauf wird von Østerlie als „Framing the problem" beschrieben. Im Laufe der Diskussion zwischen den Entwicklern wird die Rolle des Maintainers eingeführt. Aus Sicht von ANT nimmt die Maintainer-Rolle daher als Akteur am Prozess teil, ebenso wie die Entwickler. Ein weiterer Aspekt sind die offenen Fehlerberichte im Bug-Tracker, welche den Entwicklern zugeordnet sind. Auch sie tragen dazu bei, das Problem zu formulieren. Østerlie stellt die Situation grafisch mit der Abbildung 1 dar.

2.4 Schlussfolgerungen

Das Gentoo-Projekt ist in Projekte und Unterprojekte aufgeteilt, für deren Wartung jeweils Maintainer zuständig sind. Daher sehen sich die Entwickler in einer organisatorischen Hierarchie. Durch seine ANT-Analyse offenbart Østerlie die Rolle des Maintainers als Akteur im Netzwerk, gegenüber der sich der Chef-Entwickler mit seiner Idee einer API nicht durchsetzen kann. Da man annimmt, dass der Chef-Entwickler in der Hierarchie über dem Entwickler steht, ist die Verteilung der Kontrolle offenbar nicht hierarchisch. Stattdessen ist die Kontrolle im Netzwerk eingebunden, definiert durch gegenseitige Verwandtschaften zwischen den Akteuren. Die Kontrolle ist nicht im geografischen Sinne verteilt, sondern vielmehr zwischen den Akteuren. Im untersuchten Fall also zwischen den Entwicklern, der Maintainer-Rolle und den Fehlerberichten.

2.5 Kritik am Artikel

Der Artikel ist an vielen Stellen schlecht formuliert. Beim Lesen fallen lange, verschachtelte und schwer verständliche Sätze auf, die auf Dauer er-

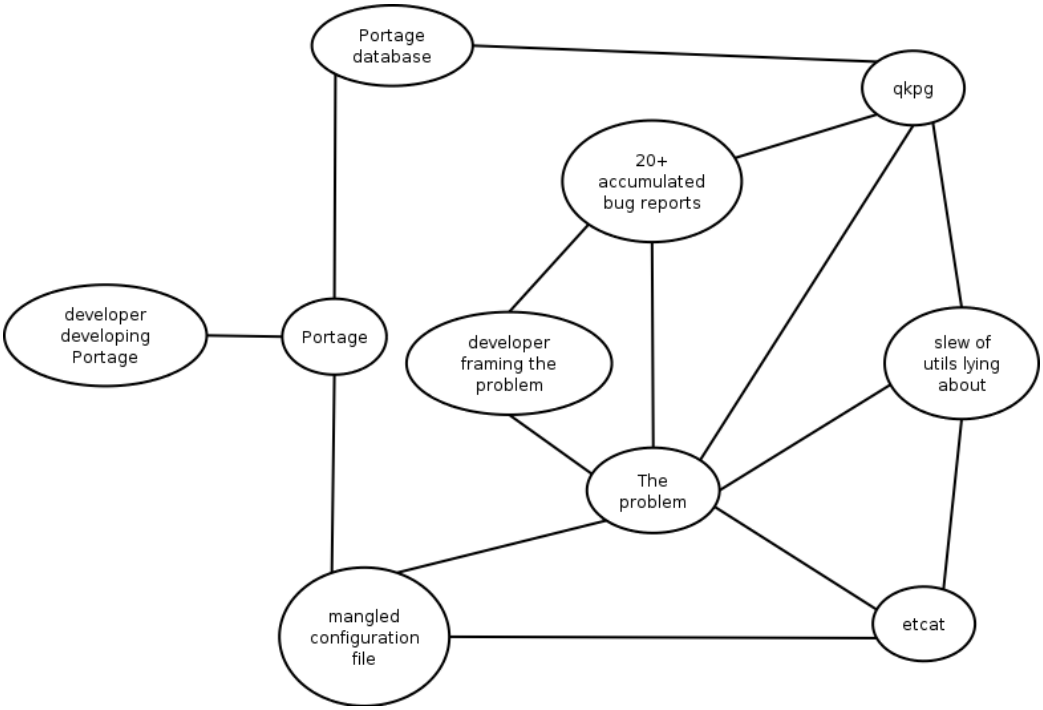

Abbildung 1: The problem framed

müdend sind. Als ein negatives Beispiel sei ein Satz aus dem letzten Abschnitt „Diskussion“ genannt:

> „By viewing of actors as inherently dispersed, thinking of the organization as an actor network shows that the hierarchical description of organization is just that: a hierarchical description of organization, an abstraction.“

Des weiteren ist negativ anzumerken, dass die von ihm entworfene Abbildung weder erklärt noch kommentiert wird. Østerlies Gedankengänge sind nicht immer nachvollziehbar. Dies wird beispielsweise im Abschnitt 3.3 des Artikels deutlich, dessen Sinn sich auch nach mehrfacher Lektüre nicht ganz erschließt. Letztlich fehlt es dem Artikel an klaren und interessanten Gesamtaussagen.

3 Fallstudie Paketmanager

Der vollständige Titel des zweiten Artikels lautet „Balancing Technological and Community Interest: The Case of Changing a Large Open Source

Software System“[OJ07]. Es handelt sich um eine Fallstudie über das Neuschreiben des Gentoo-Paketmanagers, welche Østerlie und Jaccheri gemeinsam durchgeführt haben. Bei einer Fallstudie werden durch möglichst genaue Beobachtung und Beschreibung Aussagen über eine Sache gemacht. Die Ergebnisse sind jedoch nicht repräsentativ oder direkt auf andere Fälle übertragbar.

Im Folgenden wird der Begriff *Paketmanager* erklärt, danach wird der Ablauf der Fallstudie geschildert.

3.1 Paketmanager

Ein Paketmanager verwaltet die auf einem Computer installierte Software und ist Bestandteil der meisten Linux-Distributionen. Alle Programme werden in Pakete verpackt und verteilt, wobei der Paketmanager die zentrale Stelle für die Installation und Deinstallation bildet. Dadurch ergeben sich einige Vorteile. So kann der Paketmanager Abhängigkeiten erkennen und – falls möglich – automatisch auflösen. Die Datenbank mit allen installierten Programmen ist Grundlage für eine automatisierte Aktualisierung aller Programme. Außerdem lassen sich nicht mehr benötigte Pakete einfach wieder entfernen.

Der Paketmanager von Gentoo heißt Portage[9] und ist eine der wichtigsten Komponenten der Distribution. Er wird üblicherweise über die Kommandozeile bedient, auch wenn es mittlerweile grafische Oberflächen für die Benutzung gibt.

3.2 Ablauf

Für ihre Fallstudie beobachteten und analysierten die beiden Autoren über zehn Monate hinweg den IRC-Channel des Gentoo-Projektes. Außerdem erhielten sie durch einen Projektmitarbeiter die Logdateien der vorangegangenen elf Monate. Sie führten informelle Gespräche via E-Mail und IRC mit mehreren Entwicklern und nahmen die Archive der Mailingliste, des Bug-Trackers und der Versionsverwaltung zu ihrer Datensammlung hinzu. Insgesamt ergab sich so Datenmaterial von 2002 bis Ende 2005.

Das zu untersuchende Problem war im wesentlichen die Frage, wie das Projekt die Interessen von Stabilität und Änderungen miteinander verbindet. Auf der einen Seite stehen die Endbenutzer, deren Hauptinteresse Stabilität ist. Dies widerspricht jedoch größeren Änderungen, weshalb Abwägungen notwendig sind. Je mehr Benutzer vorhanden sind, desto größer

[9] http://www.gentoo.org/proj/en/portage/

ist dabei der Konflikt. Dies wird beschrieben durch die Worte „inertia of the installed base“, was man sinngemäß als „Trägheit der Masse“ übersetzen kann. Wie groß dürfen Änderungen sein, wo und wann werden sie vorgenommen?

Seit die erste richtige Gentoo Version 2002 veröffentlicht worden war, war Gentoo zu einer der zehn größten Linux-Distributionen gewachsen[10]. Dies führte jedoch zunehmend zu Problemen mit dem Paketmanager Portage, da einige externe Programme direkt auf den Datenbanken und Konfigurationsdateien von Portage arbeiteten. Zudem war der Quellcode von schlechter Qualität. Einer der Entwickler beschrieb die Situation mit den Worten:

> „The source code is very fragile because it has evolved rather than being designed“

Alle Entwickler erkannten die Notwendigkeit, Portage neu zu schreiben und zu ersetzen, jedoch waren nur wenige zu substantiellen Änderungen in der Lage. Der erste Anlauf startete im November 2003, als vier Entwickler „Portage-ng“ – Portage next generation – ankündigten. Sie hatten die Idee eines modularen Ansatzes mit einer stabilen API. Den Kern der Anwendung wollten sie selber in Prolog schreiben, die übrigen Komponenten sollten von der Community entwickelt werden. Die Wahl von Prolog weckte in der Community jedoch Widerstand, da nur wenige Prolog beherrschten. Auch gab es Bedenken bezüglich der Geschwindigkeit. Im Dezember 2003 erschien ein konkurrierender Prototyp, geschrieben in ADA. Es folgten endlose Diskussionen über die richtige Programmiersprache. Bis Februar 2004 war das Projekt Portage-ng zum Stillstand gekommen.

Der zweite Anlauf begann im Februar 2004. Diesmal sollte der vorhandene Code modularisiert werden. Dieser Ansatz wurde jedoch durch parallele Änderungen am existierenden Code bereits nach kurzer Zeit zunichte gemacht.

Es folgte der dritte Anlauf, bei dem als Fortsetzung des zweiten Versuches eine API über die vorhandene Anwendung geschrieben werden sollte. Als Vorteil erhoffte man sich geringere Abhängigkeiten und die Möglichkeit, das Format der Konfigurationsdateien zu ändern. Auch dieser Versuch scheiterte nach anderthalb Monaten, da er sich als ungeeignet für Fernwartung erwies. Daraufhin folgte ein weiterer Anlauf, diesmal mit der Idee eines Unix Daemons, welcher von Anwendungen remote aufgerufen werden sollte.

[10] http://distrowatch.com/dwres.php?resource=major

Bis November 2006, dem Datum zu welchem der Artikel geschrieben wurde, erhielt Portage nur kleinere Änderungen und Bugfixes. Obwohl alle Entwickler darin einig waren, dass der existierende Paketmanager ersetzt werden musste, scheiterten alle Versuche über mehrere Jahre.

3.3 Analyse

In ihrer Analyse beschreiben Østerlie und Jaccheri die Gründe für das Scheitern von Portage-ng. So gab es nur vier Entwickler, die zu Änderungen in der Lage waren. Außerdem musste die Neuentwicklung parallel zur Wartung des existierenden Systems stattfinden, was von manchen Entwicklern als Verschwendung von wertvollen Ressourcen gesehen wurde. Ein tatsächliches Neuschreiben wäre nur möglich gewesen, wenn das existierende Portage für 6-12 Monate ohne Updates hätte bleiben können, was unmöglich war.

Ein weiterer Grund für das Scheitern ist das gescheiterte Mobilisieren von Ressourcen in der Community für die Entwicklung der Module. Viele Ressourcen werden in Diskussionen gebunden und der Kern-Entwicklung fehlt es an Entwicklern. So kommt es, dass der Prototyp niemals fertig wird und seinen geplanten Zweck als Kondensationskeim nicht erfüllen kann.

Weitere Gründe sind die Konkurrenzkämpfe innerhalb des Gentoo-Projektes. Zudem fehlt eine klare Übergangsstrategie, auf die sich alle geeinigt haben, daher wird ständig neu über Umfang und Reihenfolge der Änderungen diskutiert. Für die Community sind die Fähigkeiten des existierenden Systems ihre Bedürfnisse zu erfüllen offenbar wichtiger, als die Vorteile, die ein Systemwechsel gebracht hätte.

3.4 Anmerkungen

Die Analyse von Østerlie und Jaccheri ist ausführlich und in sich gut begründet. Zwar handelt es sich um eine Gentoo-spezifische Studie, doch lassen sich einige der genannten Probleme auch auf andere Fälle übertragen. In einer vergleichbaren Fallstudie über das – erfolgreiche – Neuschreiben eines großen Subsystems einer Software zur Personenverfolgung kommen Satpathy, Siebel und Rodríguez[SSR02] zu weiteren, wichtigen Schlussfolgerungen. So sehen sie die Planung im Voraus als besonders wichtig an. Entwickler sollten trainiert werden, schlecht wartbaren Code zu erkennen. Dieser Prozess ließe sich durch geeignete Werkzeuge unterstützen. Die Autoren sehen auch den Konflikt zwischen aufwändi-

gem Neuschreiben, welches vor allem von Entwicklern favorisiert wird, und dem Pflegen des vorhandenen Codes, welches von den Managern bevorzugt wird. Als Lösung sehen sie ein sorgfältiges Abwägen beider Interessen. Auch wird betont, dass jedem Rewrite ein genaues Prozessmodell zugrunde liegen sollte, was im Falle des gescheiterten Portage-Rewrites nicht der Fall war.

Literatur

[Oes04] Thomas Oesterlie. In the network: Distributed control in gentoo linux. *Proceedings of the 4th Workshop on Open Source Software Engineering*, pages 76–80, 2004.

[OJ07] Thomas Oesterlie and Letizia Jaccheri. Balancing technological and community interest: The case of changing a large open source software system. *Proceedings of the 30th Information Systems Research Conference (IRIS'30)*, pages 66–80, 2007.

[Ray99] Eric Steven Raymond. The cathedral and the bazaar. *O'Reilly*, 1999.

[SSR02] Manoranjan Satpathy, Nils T Siebel, and Daniel Rodriguez. Maintenance of object oriented systems through re-engineering: A case study. *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, pages 540–549, 2002.