Zusammenfassung wichtiger Gedanken aus dem Film *Revolution OS*

Harald Vajkonny

1 I am Your Worst Nightmare

I bumped into Craig Mundie of Microsoft in an elevator. I looked at his badge and said, "Ah, you work for Microsoft."

He looked back at me and said, "Oh yeah, and what do you do?" And I thought it was some kind of tad dismissive, here is a guy in a suit looking at a scruffy hacker. . . so I gave him a thousand yard stare and said, "I am your worst nightmare!"

(Eric S. Raymond)

2 Das Silicon Valley

früher: Neue Technologien, Neue Unternehmen, Gewinne

heute: Frontgraben im Kampf um den Wert der individuellen Freiheit

3 Was ist Linux / ein Betriebssystem? Warum gibt es Open Source?

- das Betriebssystem ist das, was dem Benutzer immer verborgen bleibt
- ursprünglich ging es nur darum einen Weg zur kollaborativen Softwareentwicklung zu finden, ohne ständig über Urheberrechtsfragen verhandeln zu müssen
- die Software sollte einfach möglichst schnell funktionieren und benutzbar sein, ohne ständig Rechtsanwälte einzubeziehen

4 Geschichte

Richard Stallman the Great Philosopher

Linus Thorvalds The Engineer

1971: Stallman arbeitet am MIT, damals in einer Hackeratmosphäre

Mid 70s: Die Atmosphäre wird unangenehmer als Passwörter eingeführt werden, die Administratoren beginnen, die Nutzer kontrollieren zu wollen

1976: Bill Gates *Open Letter To Hobbyists*: Weitergabe von Software ist Diebstahl geistigen Eigentums

1977: Ein erstes freies Unix-System entsteht: die Berkeley Software Distribution (BSD)

End 70s: Das Proprietary Software Model ist vollständig etabliert

1984: Beginn des GNU-Projekts, nach dem Gefühl, dass Closed Source-Politik allmählich die Arbeitsatmosphäre im Lab zerstört

1985: Gründung der Free Software Foundation

1989: GNU Public License

5 Was ist GNU Free Software?

- *Frei* ist zu verstehen wie in *freie Rede*, nicht wie in *Freibier*
- Man erhält die Freiheit, Änderungen vorzunehmen, um das Programm an eigene spezielle Bedürfnisse anzupassen
- Man darf das Programm verbessern
- Man darf das Programm mit oder ohne Veränderungen weiterverbreiten
- Das Programm hat ein Copyright, einen Copyrightinhaber und wird dem Nutzer über eine Lizenz überlassen. Es ist damit *nicht* Public Domain Software!
- Bei Public Domain Software, kann jeder eine kleine Änderung vornehmen, und dadurch das Programm zu proprietärer Software machen, d.h. jeder kann dann zwar die Software ausführen, Kooperation und Teilen ist dadurch aber ausgeschlossen
- Statt eines Copyright besitzt GNU-Software ein Copyleft, d.h. ein Copyright mit der Erlaubnis zu teilen und zu modifizieren. Die Redistribution ist nur unter der Bedingung erlaubt, dass das Ergebnis selbst wieder frei lizensiert wird.
- die Freiheit zur Kooperation und Bildung einer Community wird so unveräußerliches Recht

6 Creation of businesses based on Free Software

- GNU Manifesto: enthält Geschäftsmodelle, um mit Free Software Geld zu verdienen
- Cygnus: erste Firma die Consulting und Service um GNU Software kostenpflichtig anbietet
- Free Software erzeugt einen wirklich freien Markt für Service und Support,
- man erhält freie Auswahl unter konkurrierenden Support-Anbietern
- bei schlechtem Support geht man woandershin
- bei proprietärer Software kann nur der Hersteller Support geben, kein Wunder dass der Support dann meist schlecht ist

7 GNU vs. Linux

- 1990 war der GNU Toolkit weitgehend fertiggestellt, es fehlt nur noch der System-Kernel
- 1991 entwickelte Linus Thorvalds den Kernel und nannte ihn Linux
- es kam zur Verschmeldzung von Linux mit GNU
- Das Ergebnis hieß: GNU/Linux

8 Businessgeschichten

8.1 Apache

- Apache war 1995 die erste Killer-App für Linux
- Apache ist verantwortlich für den Erfolg von Linux, da dieser freie Webserver genau zu der Zeit erschien, als das Internet zu wachsen begann
- Apache erzeugt den ersten Business Case für Linux: Serverfarmen auf LAMP waren viel billiger als alle proprietären Alternativen

8.2 Red Hat

- 1993 wurde die Firma Red Hat gegründet, die eine Linux Distribution zusammenstellte und Support dafür leistete
- 1999: Börsengang von Red Hat mit 228%
- 1999 Übernahme von Cygnus
- 2003 Konzentration auf Business-Kunden, Abgabe der Entwicklung an das Fedora-Projekt
- 2006 Erwerb von JBoss

8.3 VA Linux

- 1993 gegründet um Linux -PCs zu verkaufen
- 1999 Börsengang, Aktienpreis $30, eröffnet schließlich mit bis zu $320, höchster Börsenstart jemals
- 2000 Absturz infolge des Platzens der Dotcom-Blase
- 2000 Kauf von Slashdot
- 2007 bekannt unter Sourceforge Inc.
- seit 2009 GeekNet Inc.

8.4 Netscape

- erste wirklich große Open Source-Firma
- Warum? Einzige Möglichkeit gegen Microsoft zu bestehen
- Microsoft-Strategie: Embrace, Extend and Extinguish, d.h. einen offenen Standard übernehmen, diesen erweitern, solange bis der ursprüngliche Standard nicht mehr kompatibel ist, dann den eigenen Standard mit geschlossener Software zum Monopol ausbauen.
- Hier: der offene HTTP und HTML-Standard sollte durch den Browser Internet Explorer komprommitiert werden
- Strategisches Ziel: durch die Vorherrschaft von IE sollte die Firma Netscape aus dem Server-Support-Markt gedrängt werden, womit sie ihren eigentlichen Gewinn machten
- Daher wurde 1998 der Netscape-Browser als Open Source-Projekt freigegeben
- Diese Entscheidung war eine direkte Folge der Lektüre von Raymonds *Cathedral and the Bazaar*

9 Cathedral and Bazaar (1977)

Beim **Kathedralen-Modell** wird der Quellcode eines Programmes gar nicht oder nur mit jeder neuen Software-Veröffentlichung für die Öffentlichkeit verfügbar gemacht. In den Entwicklungszeiträumen zwischen den Veröffentlichungen kann neuer Quellcode ausschließlich von einer einzigen Entwicklergruppe oder einem einzelnen Entwickler programmiert werden, die/der typischerweise bei einem Softwarehersteller angestellt ist. In diesem Fall wird der Quellcode oft als Betriebsgeheimnis behandelt und gar nicht veröffentlicht. Die Kathedrale symbolisiert die herkömmliche Entwicklungsweise: Ein Chef überwacht ein Team, welches stufenweise wie eine Pyramide aufgebaut ist. Es gibt einen Bauplan, und wenn dieser erfüllt ist, ist das Gebäude fertig.

Beim **Basar-Modell** ist der Quellcode dagegen in jedem Stadium über das Internet einsehbar. Die Entwicklung vieler Open-Source-Programme folgt diesem Schema. Dieses Modell hat sich als erfolgreicher als das Kathedralen-Modell erwiesen: Auf einem Basar bieten viele Menschen ihre Waren feil, ohne dass einer mächtiger als der andere wäre. So werden auch große Projekte koordiniert; das beste Beispiel ist der Linux-Kernel, dessen Maintainer Linus Torvalds ist. Es gibt meistens eine Person, die darauf achtet, dass das Marktrecht eingehalten wird. Zudem ist der Basar aus vielen kleinen Teilen aufgebaut – ist einer der Stände einmal nicht auf dem Basar vertreten, so ist dieser trotzdem vollständig.

Übertragen auf die Software-Entwicklung sind die Händler, welche ihre Waren feilbieten, die Programmierer, die neue Programmteile hinzufügen oder Verbesserungen vornehmen und in das Projekt integrieren wollen; der Wächter über das Marktrecht wiederum entspricht dem Maintainer eines Software-Projekts. Was eigentlich in einem heillosen Durcheinander enden müsste, wächst durch Selbstorganisation zu einer großen Software heran.

Man kann dabei niemals sagen, die Software sei „fertig“. Raymond spricht deshalb davon, dass die Softwareindustrie keine Fertigungs-, sondern eine Dienstleistungsindustrie sei.

Im Essay sind 19 Richtlinien enthalten, wie gute OpenSource Software programmiert werden kann:

1. Jede gute Software wird von einem Entwickler geschrieben, der ein persönliches Problem lösen will.
2. Gute Programmierer wissen, was sie schreiben müssen. Brillante wissen, was sie neuschreiben müssen (und was sie wiederverwenden können).
3. Plane, eins wegzuwerfen; du wirst es sowieso tun.
4. Wenn du die richtige Einstellung hast, werden dich interessante Probleme finden.
5. Wenn du das Interesse an einem Programm verlierst, ist es deine Pflicht, dieses einem kompetenten Nachfolger zu übergeben.
6. Wenn du deine Benutzer als Mitprogrammierer betrachtest, ist dies der einfachste Weg zu schneller Verbesserung und effizientem Debugging.
7. Veröffentliche früh. Veröffentliche häufig. Und höre auf die Benutzer.
8. Mit einer hinreichend großen Gruppe von Betatestern und Mitentwicklern, wird fast jedes Problem schnell erkannt und die Lösung von jemandem gefunden.
9. Schlaue Datenstrukturen und einfacher Code funktioniert viel besser als anders herum.
10. Wenn du deine Betatester wie deine wertvollste Ressource behandelst, werden sie dies auch werden.
11. Fast so gut wie eigene gute Ideen zu haben, ist es, gute Ideen von den Benutzern zu erkennen. Manchmal ist letzteres besser.
12. Meist entstehen die brillanten Lösungen aus der Erkenntnis, dass das Problem falsch verstanden wurde.
13. Perfektion (im Design) ist nicht erreicht, wenn man nichts mehr hinzufügen kann, sondern wenn nichts mehr entfernt werden kann.
14. Jedes Tool soll so funktionieren, wie erwartet. Aber ein wirklich gutes Tool führt zu Verwendungszwecken, an die du niemals gedacht hättest.
15. Wenn du Schnittstellencode schreibst, verhindere um jeden Preis, den Datenstrom zu verändern – und verwirf nur etwas, wenn dies der Empfänger verlangt.
16. Wenn deine Programmiersprache überhaupt nicht Turing-vollständig ist, kann syntaktischer Zucker dein Freund sein.
17. Ein Sicherheitssystem ist nur so sicher wie sein Geheimnis. Vermeide Pseudogeheimnisse.
18. Um ein interessantes Problem zu lösen, suche eines.
19. Mit genügend guter Kommunikation, wie über das Internet, und Führung ohne Zwang sind viele Köpfe immer besser als einer.

10 Free Software vs. Open Source

- *Free Software* klingt nach: billig, schlecht, wertlos bzw. wie ein ideologischer Generalangriff auf Urheberrechte im allgemeinen
- *Open Source* hingegen ist das, was Unternehmer lieber hören wollen
- Free Software wird leicht mißverstanden als Free Beer, also etwas womit man kein Geld verdienen kann oder darf
- Open Source: der Fokus liegt auf dem praktischen Aspekt, um eine Community um eine Software herum zu entwickeln
- Free Software: noch wichtiger als der praktische Aspekt ist der gesellschaftliche Aspekt dieser Kultur der Offenheit
- Free Software: alle Software sollte frei sein
- Open Source: freie und unfreie Software sollten nebeneinander existieren

11 Open Source Definition (Bruce Perens)

In der OSD werden neun Freiheiten garantiert:

1. Free Redistribution (Free as in Freedom, Free Price is a Side Effect)
2. Source Code Must Be Available (to maintain or port the software)
3. Derived Works Are Pemitted (improvement must be possible)
4. Integrity of the Author's Source Code (bei größeren Änderungen entweder diese deutlich markieren oder Namen des Produkts verändern)
5. No Discrimination Against Persons or Groups (z.B. auf Computern von Abtreibungsgegnern wie -befürwortern)
6. No Discrimination Against Fields of Endeavor (Business, School, Military)
7. Distribution of License (Übertragbarkeit/weitergabe der Lizenz an andere Personen)
8. License ust Not be Specific to a Product (Keine Eischränkung z.B. auf Verbreitung nur innerhalb einer Distro)
9. License Must Not Contaminate Other Software (Kein Zwang die Software nur mit anderer freier Software verbreiten zu dürfen)

Am Ende des Dokuments steht eine Liste von OS-Lizenzen: GPL, BSD u.a.

12 Free Software vs. Communism

- Lehrer erzogen früher zum Teilen (z.B. von Süßigkeiten),
- heutzutage: Lizenzen sollen respektiert werden, Software darf nicht geteilt werden
- Gesellschaftspolitisches Ziel: Good Will Society statt of Dog-Eat-Dog-Jungle
- Wichtig: Freie Software zerstört nicht die Grundlage zum Wirtschaften, sondern ermöglicht sie, weil man auch mit ihr Geld verdienen kann
- verschiedene unverschämte Lizenzklagen von Microsoft enthüllten der amerikanischen Öffentlichkeit: Microsoft ist offenbar nicht der American Dream

Kommunismus: Man muss teilen, wer nicht teilt wird bestraft oder gar umgebracht

Neoliberalismus: Es ist verboten zu teilen, weil dadurch Urheberrechte/Monopolrechte verletzt werden

Open Source: Jeder kann frei entscheiden, ob er teilen möchte oder nicht