Jörg Schilling

OSS Softwareentwicklung

mal anders Denken...

Erstes Projekt veröffentlichen

Entsteht OSS eigentlich aus dem Nichts?

- **Jeder hat sicherlich schonmal OSS geladen, kompiliert und dann verwendet, doch wie entstehen die Pakete?**
- **Planen einzelne OSS Autoren Projekte?**
  - **Das dürfte selten der Fall sein**
  - **Es sei denn der Autor arbeitet für eine OSS Firma**
- **Sind OSS Projekte verteilte private Programmierprojekte?**
  - **Das hängt vom Projekt ab**
- **Ich will eigene Software verteilen...**
  - **Soll ich mein Projekt als OSS verteilen?**
  - **Sollte ich einfach ein TAR von meinem Projekt verteilen wie es gerade ist?**

OSS Projekte von mehreren Autoren

- **Wenn mehrere Autoren an einem Projekt arbeiten muß geplant werden**
- **Meist steckt hinter solchen Projekten eine Firma**
- **Ohne Firma dahinter fangen OSS Projekte jedoch meist als „One Man Show“ an...**

Der Fall der OSS Firma

- Eine OSS Firma plant die Entwicklung ihrer Projekte
  - Das ist meist für private Projekte nicht der Fall
- Die folgenden Ausführungen sind für das One Man Show Projekt, könnten dennoch auch für Firmen interessant sein

Dieser Vortrag

- Kann nicht auf alle Einzelheiten eingehen
- Das würde den Rahmen von einer Stunde Sprengen
- Wir gehen daher nur auf alle Dinge ein, die wichtig sind
- Der Vortrag ist daher als Denkanstoß gedacht
- Wer ein eigenes Projekt veröffentlichen will, kann den Vortrag als Checkliste sehen die er eigenständig abarbeitet

Eigenes Programm als OSS veröffentlichen

- **Mal angenommen....**
  - **Ich habe ein Programm geschrieben das interessant für Andere sein könnte**
  - **Ich möchte das Programm daher als OSS veröffentlichen**
  - **Nur worauf muß ich dabei achten?**
  - **Wird sich jemand dafür interessieren?**
  - **Woher wissen Andere daß es das Programm gibt?**
  - **Wird mein Programm beachtet, oder untergehen?**
  - **Wann ist das Programm für Andere interessant?**

Zuerst entscheiden was man will

- Was soll das Ziel einer Veröffentlichung sein?
- Will man seine Software nur „loswerden“?
  - Dann könnte man es irgendwo abkippen...
- Will man daraus ein Projekt mit einer gewissen Lebensdauer machen?
  - Dann sollte man sich um die nötigen Randbedingungen kümmern
  - Dann sollte man dafür sorgen daß das Projekt nicht untergeht

Wann ist ein Programm interessant?

- Es löst ein Problem das mehr als eine Person hat
- Es ist für Suchende auffindbar
  - Ist es das nicht, wird es dennoch niemand nutzen
- Es ist auf dem System des potentiellen Nutzers lauffähig
  - Nicht alle potentiellen Nutzer verwenden die gleiche Plattform... also ist portable SW im Vorteil
- Es ist ausreichend gut dokumentiert
- Es hat keine offensichtlichen Bugs...
- ...oder es ist gut gewartet (Bugs werden schnell gefixt)

Was benötigt jemand für die Entwicklung

- Die bisherige Entwicklung sollte nachvollziehbar sein
- Die Funktionalität sollte dokumentiert sein
- Bei der Weiterentwicklung neue Fehler vermeiden
- Wie kann man das am Besten erreichen?

Worauf sollte man achten?

- Eine Versionsverwaltung erlaubt die Entstehung nachzuvollziehen und herauszufinden wann ein Bug entstand
- Eine Man Page liefern die immer aktuell ist
- Code Review machen auch bei Ein-Mann Projekten
- Automatische Regressionstests schreiben/nutzen
- Release-Notes schreiben
- Bei größeren Projekten hilft ein Bug-Tracking System
- Neue Versionen öffentlich ankündigen
- Häufig neue Versionen herausgeben

- Ich benötige keine Versionsverwaltung!
  - So etwa glaubte es Linus Torvalds von 1991 bis 2001
  - Das Resultat: Ein Chaos von TAR Archiven
  - Dann kam BitKeeper SCCS... und half beim Konvertieren
- Fazit: Eine Versionsverwaltung wird immer gebraucht
  - z.B. um Änderungen nachvollziehen zu können
  - Um herauszufinden wie ein Bug entstanden ist
  - Um zu dokumentieren welcher Code von wem ist
  - Um die Änderungskommentare aufzulisten
  - ...

Welche Versionsverwaltung ist sinnvoll?

- Am besten ein System das verteiltes Arbeiten erlaubt
  - Verteilt heißt z.B. auch Lokal ohne Netz....
  - Damit sind zentralisierte Systeme ungeeignet
  - Zentralisiert sind z.B. CVS oder SVN
  - CVS ermöglicht allerdings eine lokale Kopie..
- Was also ist sinnvoll?
  - z.B. Mercurial, SCCS, BitKeeper oder GIT
  - SCCS zur Zeit nur lokal, in einem Jahr auch remote

Wie oft sollte man Code einchecken?

- Große Projekte erwarten von ihren Mitarbeitern seltenes Einchecken
  - Das liegt meist an der sonst hohen Gesamtfrequenz
  - Aber: seltenes Einchecken kann die Nachvollziehbarkeit verhindern
- Was passiert bei sehr häufigem Einchecken?
  - Das kann kurzfristig persönlich helfen
  - Langfristig verringert das die Nachvollziehbarkeit
- Mittelweg: private Branches für die Zeit der Entwicklung
  - Diese Branches können später beseitigt werden

Aktuelle Man Pages liefern

- Undokumentierte Software will niemand verwenden
- Software, die nicht alles dokumentiert auch nicht
- Auch wenn das Schreiben von Dokumentation aufwendig ist, das ist eine der wichtigsten Aufgaben!
- Wenn Nutzer um Hilfe bitten ist meist die Man Page zu schlecht
  - Daher bei Nachfragen überlegen wie es besser geht
- Beispiel für gute Man Pages: Der POSIX Standard
- Beipiel für schlechte Man Pages: viele GNU Man Pages
  - Ausnahme: z.B. bash, hier schreibt der Autor selbst

- Code Review lohnt immer
  - Selbst wenn man alleine ist, hilft es das Projekt für einige Tage liegen zu lassen und dann anzusehen
  - Dann erst ein Commit in die Versionsverwaltung
  - Wenn man jemanden kennt, sollte der den Code begutachten
- Der Code Review sollte aber immer erst nach dem systematischen Testen erfolgen
- Wünschenswert wären Code-Review Portale

Automatische Regressionstests

- Automatische Tests sollte man eigentlich zur Abdeckung des gesamten dokumentierten Verhaltens haben
  - Nur ist der Aufwand der Erstellung dafür groß
- In jedem Fall aber sollte für jeden neu bekannten Bug ein Test dazu gebaut werden
  - Damit verhindert man, diesen Bug später wieder einzuführen
  - Wiederkehrende Bugs sind gar nicht so selten...
- Wenn man immer mal einige neue Tests hinzufügt kommt man irgendwann zu ausreichender Abdeckung

Releasenotes

- **Releasenotes dokumentieren die Entwicklung des Projekts**
- **Sie helfen Nutzern zu erkennen, ob die Installation einer neue Release wichtig ist**
- **Die informieren Nutzer über Bugfixes und Erweiterungen**
- **Sie informieren Nutzer damit indirekt auch über die Ernsthaftigkeit der Entwicklung...**
- **Zur Erzeugung kann man die Delta-Kommentare der Versionsverwaltung heranziehen**

Das Bug-Tracking System

- Kleinere Projekte benötigen kein Bug-Tracking
- Ab einer gewissen Größe ist es aber unabdingbar
- Handelt es sich um ein sehr großes Projekt wird ein Bug-Tracking System benötigt, daß eine Unterteilung in mehrere Kategorien unterstützt
  - So ein System ist z.B. Mantis

Neue Versionen ankündigen

- **z.B. in Ankündigungsportalen wie http://freshcode.club einstellen**
- **Oder auf einer eignen projektspezifischen Webseite**
- **Die Nutzer benötigen ein festes System, an das sie sich anpassen können**

Häufig Publizieren

- Keine Software ist ohne Fehler
- Wer zu selten neue Versionen herausgibt erweckt den Eindruck fehlender Wartung
- Meine Software wird meist 14-Tägig aktualisiert
  - Dazu gibt es die Schilytools mit allen Projekten
  - Wichtig sind nach Außen erkennbare Zyklen
- Wer seltener als ca. alle 6 Monate eine neue Version baut, erweckt den Eindruck das Projekt lebt nicht mehr

Portabilität verifizieren

- In regelmäßigen Abständen sollte man die Software auf allen unterstützten Plattformen kompilieren und testen
  - Empfohlen wird mindestens einmal pro Jahr
  - Das kann heute vielfach in Emulatoren geschehen

Die richtige Versionsverwaltung aussuchen

- **Welche Versionsverwaltung verwendet wird kann durchaus nach eigenem Geschmack entschieden werden**
  - **Vermieden werden sollten aber Systeme, die kein Offline-Arbeiten ermöglichen**
  - **GIT ist aktuell sehr beliebt, aber nicht zwangsläufig die beste Wahl**
  - **Es kann helfen mal mehrere Systems auszuprobieren, oder Freunde nach Erfahrungen zu befragen**

System zum automatischen Testen finden

- Es wird in jedem Fall ein System für Unit-Tests benötigt
- FreeBSD enthält ein System für Unit-Tests
- Die Schilytools enthalten ein auf Shell Skripten basiertes System für Unit-Tests
- Das Schreiben von Unit-Tests ist eine aufwendige Sache
  - Da kann man auch etwas Zeit in die Auswahl stecken

In jedem Fall…

- Ein Bug-Tracking System für das Projekt einrichten
- Eine Mailing Liste für das Projekt einrichten
  - Die Nutzer wollen eine einfach Ansprechstelle
- Freundlich auf Anfragen reagieren
  - Die Kommunikation ist das Aushängeschild
- Regelmäßig TAR Archive erstellen und verteilen
  - Gut gebaute Sourcecode Archive kompilieren durch Aufruf von *make* und benötigen sonst nichts
  - Das Klonen der Versionsverwaltung erzeugt typischerweise keine Umgebung die dem genügt

Das Erzeugen des verteilten TAR Archives

- Niemals das TAR Archiv direkt aus dem eigenen Entwicklungs-Baum erzeugen
- Stattdessen Skriptgesteuert eine Kopie als Variante des Entwicklungsbaums bauen
- Sicherstellen, daß in der Kopie die Ergebnisse auch von Programmen wie „autoconf“ enthalten sind.
- Dann aus dieser Kopie das TAR Archiv erzeugen
- Das TAR Archiv auspacken und kompilieren
- Diesen Test möglichst auch auf einem „jungfräulichen“ Rechner durchführen

Zu Guterletzt

- Die Rechner der Anwender sehen anders aus als der Eigene
- Häufig fehlen Programme oder Dateien die vom eigenen Projekt benötigt werden
- Daher auf jungfräulichem Rechner testen und Abhängigkeiten dokumentieren
- So vermeidet man enttäuschte Nutzer

Danke!