Open Source Software Release Management in der Gegenwart

oder: Wann wird die Release fertig?!?

Peter Eisentraut

Chemnitzer Linux-Tage 2009

Zeitleiste Open Source Software

1991 Linux-Kernel 0.01

1993 erste Debian-Release

1995 erste MySQL-Release

1997 “The Cathedral and the Bazaar”

1998 Netscape denkt laut über Open Source nach

2003 Linux 2.6.0 veröffentlicht

2004 Firefox 1.0 veröffentlicht

2005 Debian 3.1 veröffentlicht, nach 3 Jahren

2008 MySQL 5.1 veröffentlicht, nach 3 Jahren; Monty protestiert

Zeitleiste Peter Eisentraut

1993	erster Kontakt mit Linux
seit 1999	PostgreSQL-Entwickler
seit 2004	Debian-Entwickler
2004	erste Teilnahme Chemnitzer Linux-Tage
2008–	Sun Microsystems

Bonmots Open Source Software

Zitate aus “The Cathedral and the Bazaar”:

- “Given enough eyeballs, all bugs are shallow.”
- “Release early. Release often.”
- “If you treat your beta-testers as if they’re your most valuable resource, they will respond by becoming your most valuable resource.”

Wie sieht die Realität aus?

Release early. Release often.

“Release early. Release often. And listen to your customers.”

1. Release early – vor Abschluss aller Tests
2. Release often – sobald neue Entwicklungen vorliegen
3. Listen – Benutzerrückmeldungen einbauen

Typische Release-Zyklen

Typische Release-Zyklen

Name	Major	Minor
Linux	(2–3 Jahre)	3 Monate
Debian	2 Jahre	3–4 Monate
MySQL	2–3 Jahre	1 Monat
PostgreSQL	1 Jahr	2–3 Monate
Firefox	2 Jahre	1 Monat
OpenOffice.org	6 Monate	3 Monate
Windows	2–5 Jahre	1 Monat
Oracle	3–5 Jahre	?
Opera	2 Jahre	2 Monate

Major = neue Features, Minor = Bugfixes

Release-Ablauf

Normaler OSS-Release-Ablauf:

1. Entwickeln
2. Beta
3. Release

Linux-Kernel:

1. Entwickeln (kurz)
2. Testen (kurz)
3. Release
4. Distro-Integration
5. Distro-Beta
6. Distro-Release

Release-Ablauf

Normaler OSS-Release-Ablauf:

1. Entwickeln
2. Beta
3. Release

Linux-Kernel:

1. Entwickeln (kurz)
2. Testen (kurz)
3. Release
4. Distro-Integration
5. Distro-Beta
6. Distro-Release

Release-Ablauf/Beispiel

Release-Ablauf OpenOffice.org:

1. Entwickeln
2. Beta
3. Release

Release-Ablauf Go-OO.org:

1. Entwickeln (kurz)
2. Testen (kurz)
3. Release
4. Distro-Integration
5. Distro-Beta
6. Distro-Release

Release-Ablauf/Beispiel

Release-Ablauf OpenOffice.org:

1. Entwickeln
2. Beta
3. Release

Release-Ablauf Go-OO.org:

1. Entwickeln (kurz)
2. Testen (kurz)
3. Release
4. Distro-Integration
5. Distro-Beta
6. Distro-Release

Große Software, längerer Release-Prozess

Große Software – längerer Release-Prozess:

- mehr Features
- mehr Entwickler
- exponentielle Komplexität
- längere Beta-Phasen
- Dokumentation
- Übersetzungen
- Binärpakete
- Release-Notes
- Marketing

Release-Realität

- Offenes VCS – “Release always”
- Offener Beta-Prozess
- Häufigere Releases oft wünschenswert aber nicht machbar
- Absolute Stabilität in Minor Releases
- “RE/RO” geht für kleine Pakete und für Linus Torvalds

All Bugs are shallow

“Given enough eyeballs, all bugs are shallow.”
– oder –
“Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.”

Bugs-Tiefe

proprietäre Software:

- Alle Bugs sind “deep” aus Anwendersicht.
- Analyse Hersteller-intern je nach Bedarf
- Reparatur durch bezahlten Programmiererstab

Open Source Software:

- Die meisten Bugs sind “shallow”.
- Analyse durch Community
- Reparatur durch überlastete Haupt-Maintainer
  - Flaschenhals
  - Quelle für Enttäuschung beim Umgang mit Open Source

“One key to understanding is to realize exactly why it is that the kind of bug report non–source-aware users normally turn in tends not to be very useful. Non–source-aware users tend to report only surface symptoms; they take their environment for granted, so they (a) omit critical background data, and (b) seldom include a reliable recipe for reproducing the bug.”

Brook's Law

Brook's Law: "Adding more programmers to a late project makes it later."

—

C&B: "But on open-source projects, the halo developers work on what are in effect separable parallel subtasks and interact with each other very little; code changes and bug reports stream through the core group, and only within that small core group do we pay the full Brooksian overhead."

Brook's Law

Brook's Law: "Adding more programmers to a late project makes it later."

—

C&B: "But on open-source projects, the halo developers work on what are in effect separable parallel subtasks and interact with each other very little; code changes and bug reports stream through the core group, and only within that small core group do we pay the full Brooksian overhead."

Brook's Law und OSS

Der Schlüssel ist Modularisierung und Schnittstellen: "... going open will not necessarily save an existing project that suffers from ill-defined goals or spaghetti code ..."
Einfache Schnittstellen:

- Linux
- Debian

Komplexe Schnittstellen:

- PostgreSQL
- Firefox

Beta-Tester

“If you treat your beta-testers as if they’re your most valuable resource, they will respond by becoming your most valuable resource.”

—

“[...] we find pretty consistently that the only really limiting resource is skilled attention.”

Beta-Tester

“If you treat your beta-testers as if they’re your most valuable resource, they will respond by becoming your most valuable resource.”

—

“[...] we find pretty consistently that the only really limiting resource is skilled attention.”

Woher Beta-Tester nehmen?

Beispiel PostgreSQL:

- Release-Cycle: 1 Jahr
- ca. 9 Monate Entwicklung, 3 Monate Beta
- also: Beta-Tester nur 25% der Zeit involviert
- Wenig Zeit für Feedback

Woher Beta-Tester nehmen?

Gegenbeispiel MySQL:

- Release-Cycle: 3 Jahre
- MySQL 5.1.30 “GA” 2008
- aber: MySQL 5.1.3 “Alpha” schon 2005

Woher Beta-Tester nehmen?

Beispiel Debian: dreistufige Veröffentlichung

- Unstable – für Entwickler
- Testing – für den Desktop?
- Stable – für den Server

Also: große Anzahl Tester

Woher Beta-Tester nehmen?

Überlegungen: Was verhindert die Involvierung von Testern?

- Verlangen nach Stabilität
- Schlechte Upgrade-Möglichkeiten
- Keine Anwendungsmöglichkeiten
- Kurze Beta-Phase

Aber: Zu lange Entwicklungsphasen vergraulen Entwickler.

Release when it's ready

"Release when it's ready." ("Good things come to those who wait.")
Aber wann ist es denn "fertig"?

- alle Bugs weg?
- keine Lust mehr?
- persönliche Ziele erreicht?
- Zeitplan

Release-Zeitpläne

Der Trend geht zu festen Zeitplänen:

- OpenBSD (6 Monate)
- GNOME (6 Monate)
- Ubuntu (6 Monate)
- Debian (18 Monate)
- OpenOffice.org (6 Monate)
- PostgreSQL (12 Monate)
- (MySQL)
- ...

Support-Zeiträume

Aber: Support-Zeiträume

- Ubuntu LTS: 5 Jahre
- Debian: nächste Release + 1 Jahr (= 30-36 Monate)
- PostgreSQL: ca. 5 Jahre
- MySQL: 5 Jahre
- Linux-Kernel: 0

Zum Vergleich:

- RHEL: 5 Jahre
- Solaris: 10 Jahre

Release: Wie? Wann? Wo?

Alles muss passen:

- Support-Zeitraum
- parallel unterstützte Releases
- Release-Zyklus
- Entwicklungs-Länge
- Test-Komplexität
- Beta-Länge
- parallel entwickelte Releases
- Release-Aufwand
- interessierte Mitwirkende
- -> Management

Man kann nicht alles haben.

Entwickeln heute

- Komplexe Software hat komplexe Herausforderungen, OSS oder nicht.
- Konkrete Ziele und Zeitpläne funktionieren besser für große Projekte.
- Modularisierung und Schnittstellen sind wichtig.
- Unterschiedliche OSS-Projekte haben unterschiedliche Anforderungen und Verfahren.
- Linux-Kernel kann sich gelegentlich eine Ausnahme leisten.
- C&B ist nicht widerlegt — muss aber weitergedacht werden.
- “Human Resources” sind entscheidend.

Entwickeln ab morgen

“To solve an interesting problem, start by finding a problem that is interesting to you.”