Vortrag FLOSS-Entwicklungsmodelle: Handout

Andreas Thüring

Open-Source Seminar HS2014 Uni Basel, 30.09.2014

1 Was bedeutet Entwicklung von Open-Source Software?

"Debugging is parallelizable" Zusammenarbeit in potentiell massivem Massstab zwischen lose verbundenen Teams über die ganze Erde verteilt.

Transparenz Nicht nur der Code, sondern auch der Entwicklungs- und Entscheidungsprozess soll offen und dokumentiert sein. Aktive Mitarbeit von Usern ist ausdrücklich erwünscht und soll auch niederschwellig möglich sein. Oft ermöglichen meritokratische und demokratische Systeme eine aktive Zusammenarbeit für Interessierte.

Organisation Zur Koordination eines solchen Prozesses bedarf es Organisationsstrukturen, welche die Entwicklung des Projekts in die richtige Richtung leiten und eine möglichst reibungsfreie Integration von Bugfixes und neuen Features in die Codebasis ermöglichen. Je nach Grösse des Projektes wird die Rolle der Organisation von nur einer Person bis hin zu Stiftungen oder kommerziellen Firmen übernommen.

"Release Early, Release Often" Features werden inkrementell in die Codebasis eingeführt und zumeist in kurzen Releasezyklen veröffentlicht. Währenddessen soll die Funktionstüchtigkeit der momentanen Release-Version immer garantiert werden und neu gefundene Bugfixes werden sofort gepatcht. Dies soll ein möglichst reibungsfreies "hinübergleiten" in eine neue Version ermöglichen.

Benutzte Quellen: [4], [1]

2 Wie ist Open-Source Software-Entwicklung organisiert?

2.1 Wer ist beteiligt?

Im folgenden sind einige Rollen beschrieben, die üblicherweise bei jedem FLOSS-Projekt in der ein oder anderen Form vorkommen. Dabei kann eine Person durchaus verschiedene Rollen gleichzeitig einnehmen!

User Der Endbenutzer der Software.

Contributor Jeder, der Code, Dokumentation, Bugfixes und weiteres in das System einbringt. Oft auf freiwilliger Basis, manchmal auch in bezahlter Form. Contributor können Einzelpersonen oder ganze Organisationen sein.

Maintainer Verantwortlich für die organisatorische Leitung eines (Sub)systems. Überprüft hereinkommenden Code und koordiniert den Entwicklungs- und Kommunikationsprozess in seinem Kompetenzbereich. Bei grösseren Projekten sind die Maintainer oft hierarchisch organisiert.

Kommunikation findet hauptsächlich über das Internet statt. Benutzte Tools sind mannigfaltig und je nach Projekt verschieden: IRC, Mailing-Listen, Bugtracker, git(hub), Wikis etc.

2.2 Release-Management: Beispiel Linux-Kernel

Die Entwicklung erfolgt zu jedem Zeitpunkt an folgenden vier Bäumen:

Prepatch Für Entwickler und Enthusiasten: Hier erfolgt die aktuelle Entwicklung und Integration von neuen Features: Release-Zyklen sind extrem kurz und Inkompatibilitäten und Bugs müssen erwartet werden. Maintainer: Linus Torvalds.

Mainline Hier werden neue Features eingeführt, die sich in der Entwicklerversion bewährt haben. Alle 2-3 Monate wird eine neue Version veröffentlicht. Maintainer: Linus Torvalds.

Stable Sobald eine Mainline-Version für den Release freigegeben wird, wird sie zur stable-Version. Dies ist die empfohlene Version für End-User. Stable erhält Bugfixes aus der aktuellen Mainline-Entwicklung (Backporting), nicht aber die Features.

Longterm Bestimmte Kernel werden für einige Zeit weiter maintained, auch wenn sie nicht mehr aktuell sind. Dies, um Backporting von Sicherheitsupdates für Nutzer dieser Legacy-Kernel zu ermöglichen.

Benutzte Quellen: [2]

2.3 Wie kommt Code ins System? Beispiel Linux-Kernel

1. Diskussion eines neuen Features oder Bugfixes unter Contributorn und Usern z.B. in der Mailing-Liste eines Subsystems.
2. Design und Implementation des Features durch einen oder mehrere Contributor. Während der Entwicklung kontinuierliches Testen.
3. Checkin des Patches durch Contributor in das Repository des Maintainers. Maintainer überprüft den Code. Befindet der Maintainer den Code für gut, gibt er den Patch an den nächsthöheren Maintainer weiter. Wenn nicht, startet der Prozess wieder von vorne.
4. Der Patch wird vom Maintainer der Entwicklungs-Version in diese integriert, wenn dieser als bugfrei befunden wird und keine Kompatibilitäten bricht.
5. Noch aufkommende Fehler werden gefixt in Zusammenarbeit mit der gesamten Entwickler-Community
6. Ist das Feature ausgearbeitet genug für die End-User, wird es schliesslich in die Release-Version übernommen. Weitere Entwicklung von Features und Fixes fängt nun wieder am Anfang des Prozesses an.

Benutzte Quellen: [1]

3 Semantic Versioning

Ein Versuch, Versionsnummern zu standardisieren und mit Bedeutung zu belegen, mit dem Ziel überkomplizierte bis unlösbare Versionsabhängigkeiten zu verhindern. Alle Versionsnummern haben dabei das Format `MAJOR.MINOR.PATCH`.
Der Release-Zyklus beginnt mit der Veröffentlichung einer API mit der Versionsnummer `1.0.0`. In der weiteren Entwicklung wird wie folgt verfahren:

- Erhöhe `MAJOR`-Version um 1 wenn inkompatible API-Änderungen eingeführt werden.
- Erhöhe `MINOR`-Version um 1 wenn rückwärtskompatible Features eingefürt werden.
- Erhöhe `PATCH`-Version um 1 für rückwärtskompatible Bugfixes.

Benutzte Quellen: [3]

References

[1] `http://www.linuxfoundation.org/publications/understanding-the-open-source-development-model`

[2] `http://www.linuxfoundation.org/publications/linux-foundation/who-writes-linux-2013`

[3] `http://semver.org`

[4] `http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/`