Release Management in Large Free Software Projects

Martin Michlmayr
University of Cambridge
tbm@cyrius.com

Agenda

- Background of this research
- Projects: selection criteria; problems and solutions
- Why time-based releases work
- Implementing time-based releases
- Conclusions

Background

- Investigating free software from a quality perspective
- Approach: issues of coordination and management
- Process improvement
- Problematic areas? Release management

Project selection

- Large and complex
- Voluntary
- Distributed
- Time-based

Projects

Project	Interval	Introduction
Debian	15-18 months	middle of 2005
GCC	6 months	2001
GNOME	6 months	beginning of 2003
Linux kernel	2 week merge	middle of 2005
OpenOffice.org	3 months	beginning of 2005
Plone	6 months	beginning of 2006
X.org	6 months	end of 2005

Debian

Version	Release Date	Months
1.1	1996-06-17	
1.2	1996-12-12	6
1.3	1997-06-02	6
2.0	1998-07-24	14
2.1	1999-03-09	7
2.2	2000-08-14	17
3.0	2002-07-19	23
3.1	2005-06-06	35
4.0	2007-04-08	22

Debian

Past problems

- Release management was not very organized; infrequent release updates
- Blockers found late during the release
- Delays: out of date software
- Bad image for the project

Solutions

- Implementation of better release management structures
- A release date was set well in advance
- Regular release announcements and updates
- Definition of release targets
- Clarification of responsibilities

Outstanding problems

- Developers need to see that targets can be met

GCC

Version	Release Date	Months
3.0	2001-06-18	
3.1	2002-05-15	11
3.2	2002-08-14	3
3.3	2003-05-13	9
3.4.0	2004-04-18	11
4.0.0	2005-04-20	12
4.1.0	2006-02-28	10

GCC

Past problems

- Closed development
- Long time between releases, no public snapshots
- When development picked up, changes often broke development tree

Solutions

- Introduction of open development style, steering committee
- Division of development phase into 3 stages
- Patches are peer reviewed

Outstanding problems

- The release manager is busy

GNOME

Version	Release Date	Months
1.0	1999-03-03	
1.2	2000-05-25	15
1.4	2001-04-02	10
2.0	2002-06-27	15
2.2	2003-02-06	7
2.4	2003-09-11	7
2.6	2004-03-31	7
2.8	2004-09-15	6
2.10	2005-03-09	6
2.12	2005-09-07	6
2.14	2006-03-15	6
2.16	2006-09-06	6
2.18	2007-03-14	6

GNOME

Past problems

- Version 2.0 was supposed to mainly change internal interfaces. Delays. Developers frustration
- It was not clear what was going on
- Freezes often came unexpectedly, did not lead to a release
- Vendors had deadlines but GNOME's schedule was unpredictable

Solutions

- Introduction of a rigorous schedule and policies
- Introduction of the idea of reverting
- The project gained credibility because releases were actually performed on time

Outstanding problems

- Concerns whether this release cycle makes the project less innovative

Linux

Version	Release Date	Months
1.0	1994-03-14	
1.2	1995-03-07	12
2.0	1996-06-09	15
2.2	1999-01-25	31
2.4	2001-01-04	23
2.6	2003-12-17	35

Linux

Past problems

- Due to the long release cycle, many changes accumulated
- Features got out very slowly
- Vendors backported many features to their own releases

Solutions

- New versions are now released every two or three months
- Steady flow of code into production and many people get to test the new code
- Features get out more quickly
- Vendors can directly work with current releases and the community

Outstanding problems

- There is no long-term stable version
- Regressions between versions are often introduced

OpenOffice.org

Version	Release Date	Months
1.0	2002-05-01	
1.1	2003-09-02	16
2.0	2005-10-20	26
2.0.1	2005-12-21	2
2.0.2	2006-03-08	3
2.0.3	2006-06-29	4
2.0.4	2006-10-13	3
2.1.0	2006-12-12	2
2.2.0	2007-03-29	4

OpenOffice.org

Past problems

- Due to the long release cycle little testing occurred
- Many changes accumulated
- Features were put in very late, even during the beta cycle
- Vendors shipped unreleased versions

Solutions

- The project moved to a 3 month release interval, creating a tight feedback loop with users
- Better planning allows more collaboration between vendors
- Motivation in the project has increased
- The release process has become more transparent

Outstanding problems

- Move from 3 to 6 months: too much pressure on QA, and users don't want to upgrade

Plone

Version	Release Date	Months
1.0	2003-02-06	
2.0	2004-03-23	13
2.1	2005-09-06	17
2.5	2006-06-16	9

Plone

Past problems

- Releases took a long time to get out
- Releases had many changes and caused migration problems
- Unpredictability of Plone is bad for web developers

Solutions

- Implementation of better development structures
- Deadlines have motivated developers to finish features
- Web developers can decide in advance which version to use

Outstanding problems

- Can they release on time?

X.org

Version	Release Date	Months
7.0	2005-12-21	
7.1	2006-05-22	5
7.2	2007-02-15	9

X.org

Past problems

- XFree86: infrequent releases, no plan, and rigid structure
- The code base was huge and monolithic: hard to test and attract new volunteers

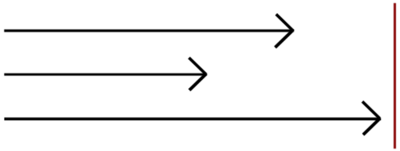
Solutions

- X.org moved from a monolithic to a modular system
- Introduction of two release mechanisms: releases of individual components, and roll-up releases of all components
- Creation of a fall back mechanism in case components are not ready for release

Outstanding problems

- Get experience with time-based releases

The fundamental problem



- Independent development, little coordination
- Release: requires alignment of all work
- Sudden, unexpected call of alignment leads to problems

What are time-based releases?

- Instead of releasing when a certain set of features has been achieved, you release according to time
- You don't have to release on the specific release date if there are issues
- You can still plan to have features, just not wait for them

What conditions are necessary?

- Enough work gets done
- Distribution is cheap
- Releases don’t require specific functionality
- The project is modular

Coordination mechanisms

- Regularity
  - Reference point
  - Discipline and self-restraint
  - Familiarity
- Schedule
  - Gives people information to work independently
  - Reduces coordination

Incentives

- Organizations: predictability
- Users: periodical fixes, smooth upgrades
- Developers: know when they have to get code in, contributions get out to users quickly
- Vendors: can plan and work with the community

Release Interval

- Regularity and predictability
- User requirements
- Commercial interests: e.g. book authors
- Cost factors related to releasing
  - Support for old releases
  - Fixed costs of releases
  - Confusion among users
  - Fragmentation of users
  - Upgrade costs
- Network effects: working with other projects and distributions

Conclusions

- Some free software projects have successfully reacted to change (growth, users, etc.)
- Time-based releases are effective because they introduce two coordination mechanisms: regularity and the use of schedules.
- Time-based releases are an effective mechanism to establish better planning in projects with little control over voluntary contributors.
- What does this mean for other volunteer projects?
- More information: http://www.cyrius.com/research/