Release Management Within Open Source Projects

Justin R. Erenkrantz

Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
jerenkra@ics.uci.edu

Abstract

A simple classification system for release management practices is presented. When applied to a set of projects, in this case a set of open source projects, distinctive practices are highlighted and relative strengths can be assessed. Three projects are studied, the Linux kernel, Subversion, and the Apache HTTP server. Their release practices, as portrayed by the classification system, emerge as a complex combination of subprocesses and tools chosen to support specific project goals and properties. Through application of this classification, we identify areas of weaknesses in the projects' release management processes and conclude with an overview of potential improvements to the processes and tools that can be made.

1. Introduction

An important aspect of open source projects is conducting a release. While the source code is freely available and usually developed in an open manner, the ratio of users may greatly exceed the number of useful developers[3]. Therefore, an open source project may need to package their software in a way that is conducive to attracting non-technical users.

We will identify release management by the processes and tools used to package the software for public consumption. Some projects may place expectations on developers to have certain tools available or understand how the development process works. However, when creating a release for a greater audience, these same expectations may not apply. For example, users of a web server may not be interested in how the web server functions. They may only be concerned with whether it meets their requirements.

The state of release management guidelines in open source projects has been remarkably informal so far. It may be difficult to get a clear picture of how a project conducts its releases. The processes and tools may only be evident by their side effects rather than a direct statement of the process. This presents a significant challenge to understanding release management and improving the processes and tools for open source projects.

2. Taxonomy

To help understand release management policies, we introduce a taxonomy for identifying common properties of release management across open source projects. The characteristics as presented in this paper are: release authority, versioning, pre-release testing, approval of releases, distribution, and formats.

2.1. Release Authority

A release is usually accompanied by the presence of a coordinating authority. This coordinator is commonly identified as the release manager. The release manager has the responsibility for conducting the release. The release manager may also have final authority over what makes it into a release.

Depending upon the organization of the project, the release manager may be a dedicated person. This person will be the focal point for all releases. Another common strategy is to enlist different people for a release depending upon participant availability. Therefore, the specifics of the release policy may change over time as different people handle the release responsibilities. The presence of documented release guidelines may assist in helping new release managers understand what they should do.

2.2. Versioning

A release is identified by its associated version number. The version number of a release signifies some property of the release to the community. Typical purposes of versioning is to identify the stability of the release.

Since open source projects often support concurrent development, there may be parallel branches of development. Each branch may be released independently and at different stages of maturity. Therefore, it could be crucial to identify from which branch a release originates by its version number.

Some projects denote the expected quality by placing a label such as alpha or beta on releases. A project may then define the expectations that go along with this label. Another quality often identified by a version number is whether it is a release candidate. A release candidate may be produced before a release to try to obtain feedback before an actual release is conducted. These release candidates may be a good mechanism for increasing the expectations of release quality.

2.3. Pre-release Testing

Oftentimes, a project has a set of criteria that a release must satisfy before it can be called a release. These criteria may be that the code passes an acceptance test, new features may need to be present, or particular issues are resolved. This stage verifies that the criteria has been met satisfactorily. Pre-release testing should be conducted in a manner that promotes independent parties to verify the results of pre-release testing.

2.4. Approval of Releases

A policy may be in place for approving releases before the release is official. The management structure of the project may agree that the requirements for a release have been met. This may be strongly correlated with the amount of authority the

	Release Authority	Versioning	Pre-release Testing	Approval of Releases	Distribution	Formats
Linux Kernel	Designated	Stable/unstable	Release Candidates	Release Manager	Mirrors	No official contributions
Subversion	Subset of committers	Pre-1.0	Regression	Release Manager	Single site	User contributed
Apache HTTP Server	Committer	Stable/unstable	Real-world, Automated	Consensus	Mirrors	Committer contributed

Table 1: Release Management Strategies

release manager has. Release managers may be able to approve a release independently. Or, the release may need final approval from another group.

2.5. Distribution

Distribution consists of two aspects: visibility and accessibility. After the release has been approved, the community needs to be made aware of its presence. Furthermore, a viable system needs to be in place to handle the demand for the new release. Additionally, integrity of the release needs to be ensured so that the community is confident that the release has not been tampered before delivery.

2.6. Formats

A project may have guidelines that govern how packaging of the release is accepted and generated. Ideally, enough formats should be covered so that all users can deal with the release in their preferred packaging format with a minimal degree of hassle. In addition to providing the source to the release, a release may also offer binaries. Depending upon the implementation language, these binaries may be system-dependent or system-independent.

2.7. Projects

For this paper, three open source projects were examined to determine their release management practices. Each of these projects is in a different domain and has varying process and tools in place to support releases. A summary of the strategies taken by each project is listed in Table 1.

3. Linux Kernel

The Linux kernel is at the heart of the GNU/Linux operating system. It was initially written by Linus Torvalds in 1991 and he still remains as the central authority figure. However, Linus delegates his authority across many participants. There are kernel maintainers who are responsible for a section of the kernel. The other position of authority is the release manager for a branch.

The Linux kernel has multiple active branches open at any one time. Due to Linux's versioning system, each branch can conduct a release independently. These branches are usually at different stages of the software life cycle. Once a release is made, it is then distributed across a set of mirrors for the public to download.

3.1. Release Authority

The Linux kernel has a single designated release manager for each open branch. The release manager has final authority on all kernel releases. As denoted in Table 2, there are currently three open release trees with different release managers. A release managers for a tree does not have to receive approval from any other release manager.

Version	Release Manager
2.2	Alan Cox
2.4	Marcelo Tosatti
2.5	Linus Torvalds

Table 2: Current Linux Kernel Release Managers

Usually, the release manager accepts patches from a variety of maintainers. Each maintainer is responsible for a particular component of the kernel. The general public should submit patches to the maintainer rather than the release manager. The changes are then integrated into the official tree by the release manager.

Linux currently uses a decentralized repository system utilizing BitKeeper. Previously, there was no way to maintain the current copy of the tree as viewed by the release manager. Now, it is possible for a release manager to integrate entire changesets from maintainers in an automated fashion. This has reduced the amount of work that a release maintainer spends on integrating patches.

Additionally, people are free to release kernels on their own without the approval of the release manager. However, these versions are not distributed in the same fashion as official releases. Furthermore, these custom releases are usually denoted by appending the author's initials to the version number from which the release originated.

3.2. Versioning

The Linux kernel uses a versioning scheme consisting of three integers arranged in the following pattern: x.y.z. The integer x in this pattern represents the major release number; y denotes the minor release number; and z indicates the patch level of the release.

Due to the high level of activity in the Linux kernel, there are usually at least two streams of development open at any time. The stable releases are indicated by an even minor release number, while unstable releases are denoted by an odd minor release number.

A challenge is presented in this scheme as to when a release should be handed off to the next release manager. Linus Torvalds is usually in charge of the unstable trees. In the past, when an unstable tree becomes stable, Linus has maintained release management for the first few stable releases of this new tree. After he is confident in the stability of the tree, he will name the new release manager and will start on the next unstable tree.

3.3. Pre-release Testing

Rather than performing extensive testing on a release, the Linux release model typically relies upon its user community to test a release before it is made official. Testing by the release manager may be impractical since it is often hard to obtain access to esoteric hardware configurations. These release candidates are usually denoted with a -pre tag appended to their version number. After the release candidate is distributed, feedback is solicited from the user community as to the stability of this release. This allows preliminary feedback as to the portability and stability of the release candidate.

3.4. Approval of Releases

Since the release manager has total authority over the release, the Linux kernel release process does not require approval by the community. However, it is usually desirable to receive positive feedback from the users of a release candidate. If a problem is discovered in a release candidate, it can usually be resolved before the final release.

3.5. Distribution

Official releases are indicated by an email to the linux-kernel mailing list. Usually, a list of changes from the previous release are included in the email. These releases will be made available via the kernel.org mirroring system[5]. Currently, there are 130 mirrors spread across the globe that assist in the distribution of Linux kernel releases. These mirrors are accessible via round-robin DNS entries keyed off the ISO country code.

In order to ensure the integrity of the downloads, every file available on the official mirrors are digitally signed using an OpenPGP key[6]. This allows the end users to verify that the download was received correctly without tampering.

3.6. Formats

For an official release, complete tarfiles of the kernel source compressed with either gzip and bzip2 are available. Also, incremental patches versus the last official release are made available. For release candidates, patches are available in a compressed format versus the prior official release. Incremental to the last release candidate and changeset differences from prior releases are usually available as well.

4. Subversion

Subversion is designed as a compelling replacement for CVS[2]. The infrastructure and core developers are funded by CollabNet. While many other developers contribute to Subversion, the CollabNet developers act as a controlling authority over the release process.

4.1. Release Authority

Due to the social organization of the Subversion project, the developers from CollabNet act as the central authority in the project. Therefore, all releases are conducted by the CollabNet developers. They have final say as to what makes it into a release. The specific release manager for a release is decided internally.

4.2. Versioning

Subversion is currently only available in alpha releases. Due to its low maturity level, Subversion does not currently have a strong versioning scheme. Issues are currently distinguished between being fixed before 1.0 and those that can be completed after 1.0. However, the meaning of a 1.0 version is not yet concrete.

4.3. Pre-release Testing

Subversion has a thorough automated test suite that performs regression testing. Therefore, the developers can run the test against a release candidate. If the candidate does not pass the test suite, the release may be held until it successfully passes.

4.4. Approval of Releases

Since releases are conducted by a controlling authority, the rest of the community does not get final approval for a release. However, the CollabNet developers rely upon an issue tracking system[1]. All issues that are slated to be fixed in a release milestone must be closed before the corresponding release can be made.

4.5. Distribution

Currently, Subversion does not utilize mirroring for release distribution. All Subversion releases are served by a single CollabNet server. Since access to the server is controlled, the releases are not digitally signed by the developers. Announcement emails are sent to a special announcements list as well as the regular development list.

4.6. Formats

Initially, a Subversion release is only accompanied by a compressed gzip tar file corresponding to the source for a particular release. However, the Subversion development community encourages packaging contributions by its users to increase the number of formats that are available for download. Once the new format has been submitted and approved for distribution, it is made publicly available. Table 3 indicates current release formats that are available for Subversion.

5. Apache HTTP Server

The Apache HTTP Server is currently the most popular

GZip Tar files (source)	RedHat 7.x/8.x RPMS
Mandrake 8.x RPMS	Source RPMs
Windows Installer	Windows Binaries

Table 3: Subversion Release Formats

HTTP server[7]. The project is organized as a meritocracy[4]. The Apache HTTP Server project guidelines define three groups of participants: Apache HTTP Server Project Management Committee members, Apache HTTP Server Committers, and Apache Developers[9]. A member of the Project Management Committee is responsible for setting the direction of the project. A committer can make changes to the master repository. And, a developer is defined as anyone who contributes to the project.

5.1. Release Authority

Due to the decentralized organization of the Apache HTTP Server Project, there is no designated authority that will conduct every release. Each member of the Apache HTTP Server Project Management Committee has the authority to conduct a release. However, a volunteer from this group may come forward and act as release manager.

Releases do not have predefined objectives or timeframes. A release only occurs when there is enough interest and a person is willing to take on the responsibility. Consequently, this person has complete authority over what makes it into a release. This volunteer is responsible for deciding what patches and changes will be included. They are also responsible for creating the release artifacts.

When a release manager starts to conduct a release, feedback from the community is usually solicited. Developers may indicate that they have some changes that they wish to see make it in to the release. The release manager can decide if the release should be held for these changes.

5.2. Versioning

The Apache HTTP Server Project currently has two versioning models: stable and unstable[8]. A recent shift has been made to follow a similar versioning scheme as the Linux kernel (see 3.2): consisting of a major, minor, and patch number.

A stable release is denoted by an even minor version component. Stable releases are expected to be backwards-compatible with prior stable releases with the same minor version component. These releases should be expected to be able to run in production without significant faults.

An unstable release is indicated by an odd minor version. Generally, an unstable release may not have received the same degree of testing as a stable release. These releases are primarily meant for allowing third-party module developers to provide early feedback, use new interfaces, and suggest improvements.

5.3. Pre-release Testing

The release manager will usually issue a release candidate and announce it on the development mailing list and to a specific testers list. Then, the other developers can test the release and provide preliminary feedback. Each developer is free to conduct their own tests on their preferred platforms. The developers will report back with their quality interpretation of the release candidate.

The release manager may also choose to run the automated test suite located in the httpd-test framework. This repository is a set of Perl-based tests that are designed to allow for regression testing. When an issue is resolved, a developer may decide to also commit a regression test check into the repository. Adding new test cases is not mandatory, therefore the regression test suite is not as thorough as it might otherwise be.

A stable release is usually preceded by a run of the release candidate on the main apache.org web server. By running the code in production for a period of time on a high-traffic site, it allows real-world feedback to be obtained. After a release has satisfactorily handled requests for a few days, the community may feel more confident about the stability of the release.

5.4. Approval of Releases

While the release manager has full authority over what makes it into a release, the rest of the group must approve the release for publication. A release can only be made public after at least three committers have positively approved the release on the development mailing list and more positive than negative votes are received. The release approval voting system is in contrast to the code policy where negative votes can not be overriden[10],

5.5. Distribution

The Apache HTTP Server utilizes a system of volunteer mirrors spread across the globe. Each mirror provides releases for all Apache Software Foundation projects. The releases are digitally signed by the release manager to ensure the integrity of the files. A MD5 hash of each file is also provided. After a release has been approved and the release has been propagated to all of the mirrors, an email is sent to a dedicated announcement list.

5.6. Formats

Initially, the releases are available as a tar file compressed in either gzip or compress formats. This is achieved by an automated script[11]. Additional formats are not required to be available before a release is made public, but other formats (such as Windows binaries and source formats) are usually available before the announcement is sent.

In order to ensure the integrity of the releases, binaries and other packaging contributions will not be accepted for official distribution by any one that is not an active developer. Other parties may distribute the code in their other formats on their own site, but these formats will not be available officially.

6. Discussion

We have examined three open source projects and their release management procedures. In the case of the release

authority, the extent of the decentralization seems to have a direct impact on how the release authority is handled. On projects with a controlling authority, the releases are handled by a controlling authority. If the project has no controlling authority, it may be difficult to obtain authority. However, a project with a decentralized organizational structure may still be willing to have a dedicated release authority.

For Linux, one of the underdeveloped areas of its release management is its handling of pre-release testing. It is currently done in an informal manner. This may lead to problems being identified shortly after a release. Developing an automated system where reports can be reported and examined may help.

Due to Subversion's organizational structure, it currently does not support geographic dispersal of distribution load. As Subversion grows in popularity, it may encounter problems with scalability of distributing releases. However, since Subversion is backed by a single organization, it bears the responsibility to handle the distribution load. Another issue with Subversion is that its versioning policies are not as well defined as the other projects examined in this paper. However, as Subversion approaches a 1.0 release, a more formal definition of versioning may occur.

The largest problem in the Apache HTTP Server is the decentralized release authority. This stems from the decentralized organizational structure of the project. Releases may not occur as frequently as the time commitment of particular volunteers may be sporadic. Assigning a dedicated release manager like Linux does may not be realistic for Apache HTTP Server. A potential side-effect of not having a dedicated release manager is that releases only occur when enough people have the time to produce a release.

We have applied this suggested taxonomy to three open source projects and their release management practices. We believe that this taxonomy may be beneficial to other projects - both commercial and open source. This taxonomy allows the breaking down of a typically confusing area of software development in order to promote better understanding. By identifying areas of potential weakness, it should be possible to improve the practices of a project.

7. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 0205724.

8. References

[1] CollabNet. *IssueZilla*. <http://subversion.tigris.org/servlets/ProjectIssues>, HTML, 2003.

[2] CollabNet. *Subversion.* <http://subversion.tigris.org/>, HTML, 2003.

[3] Cox, A. *Cathedrals, Bazaars, and the Town Council*. <http://slashdot.org/features/98/10/13/1423253.shtml>, Slashdot, HTML, 1998.

[4] Fielding, R.T. Shared Leadership in the Apache Project. *Communications of the ACM*. 42(4), p. 42-3, 1999.

[5] Kernel.Org Organization. *The Linux Kernel Archive Mirror System.* <http://www.kernel.org/mirrors/>, HTML, 2002.

[6] Kernel.Org Organization. *The Linux Kernel Archive OpenPGP Signature*. <http://www.kernel.org/signature.html>, HTML, 2002.

[7] Netcraft. *Netcraft Web Server Survey.* <http://www.netcraft.com/survey/>, HTML, 2003.

[8] The Apache HTTP Server Project. *Apache 2.x Versioning.* <http://cvs.apache.org/viewcvs.cgi/httpd-2.0/VERSIONING?rev=HEAD>, HTML, 2002.

[9] The Apache HTTP Server Project. *Apache HTTP Server Project Guidelines and Voting Rules.* <http://httpd.apache.org/dev/guidelines.html>, HTML, 2003.

[10] The Apache HTTP Server Project. *Apache HTTP Server Release Guidelines.* <http://httpd.apache.org/dev/release.html>, HTML, 2003.

[11] The Apache HTTP Server Project. *Apache HTTP Server Release Script.* <http://cvs.apache.org/viewcvs.cgi/httpd-dist/tools/release.sh?rev=HEAD>, Shell Script, 2003.