Subversion 1.5: A Case Study in Open Source Release Mismanagement

Hyrum K. Wright and Dewayne E. Perry
Empirical Software Engineering Laboratory
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
{hwright,perry}@ece.utexas.edu

Abstract

In June 2008, the Subversion development team released Subversion 1.5.0. This release contained a number of new features, but arrived only after a long and difficult development, test and release cycle. This protracted process confused and frustrated both users and developers. In this paper, we discuss the events which led to this breakdown, how the release process is being improved, and what lessons other open source projects can learn from the Subversion community's mistakes.

1 Introduction

Open source projects rely upon timely and quality releases to encourage higher user adoption and attract developers. These releases are the public image of the project, and influence both mindshare and market share. Users adopt projects they see as responsive to their needs, while developers enjoy being part of a vibrant development community. A timely release process can help achieve these goals.

Subversion is a popular version control system whose initial goal was to replace the aging CVS with a more modern design and feature set. With the release of Subversion 1.4.0, these goals were largely accomplished, and the community focused on making additional improvements to Subversion. These included features requested by both open source users and corporate deployments, with the primary one being merge tracking.

As the release manager, one of the authors saw first-hand the frustration and confusion caused by the protracted release cycle for Subversion 1.5.0. This release cycle lasted much longer than anticipated, which negatively impacted both users and developers. This paper offers a retrospective view of an actual release experience, how it could have been better, and what the community is doing to improve the process for future release cycles.

We recount the history of the Subversion 1.5.0 release process, which ultimately ended in the delivery of Subversion 1.5.0 in June 2008, more than a year after planned, and almost two years after Subversion 1.4.0. We discuss the established process of creating a Subversion release, how that process helped and hindered the 1.5.0 release, and ultimately what lessons can be learned from this experience for Subversion and other open source projects. We also discuss the subsequent steps the Subversion community is taking to address these issues in upcoming feature releases.

2 Subversion Release Process

In the early days of the project, Subversion developers established a guiding document known as the "Hacker's Guide to Subversion" [2]. Colloquially referred to as HACKING, this document outlines many aspects of community processes and procedures, including release processes. Although the community allows for circumstantial variation in these processes, HACKING is fairly specific as to how the release process should proceed.

Crafting a release of Subversion involves many individuals in a coordinated effort following established procedures. In the following sections, we describe the roles these individuals fill, the different types of releases, and the version numbering scheme for Subversion releases. We also describe the process used to create a new feature release of Subversion. In Section 3, we compare the ideal described here with what actually happened when releasing Subversion 1.5.0.

2.1 Community Roles

In a large and complex open source community, such as Subversion, different members take on different roles

within the project. Individuals may fill more than one role, (i.e., a person may be both the release manager and a committer), but the roles themselves are distinct [7]. Below we describe the pertinent roles in creating a release of Subversion.

2.1.1 Release manager

The release manager for the Subversion project is a volunteer individual who oversees the entire release process. Typically one of the developers, the release manager coordinates branching dates, signature collection, tarball distribution and release publication and announcement. Rarely does the release manager make unilateral decisions, but his voice is influential in directing the release process and coordinating discussion within the community.

2.1.2 Committers

Committers are individuals with full commit rights to all locations in the Subversion source code repository. How an individual becomes a committer is beyond the scope of this paper, but the primary qualifications for this designation are good judgment and trust within the community. As part of the release process, committers run independent tests of the candidate tarball on a platform of their choosing. Upon successful completion of the tests, they provide cryptographic signatures verifying the integrity of the release.

2.1.3 Third-party distributors

Rarely do users make direct use of Subversion source code as provided, and the project itself does not provide binary packages. Instead, a vibrant community of third-party distributors provides binary packages of Subversion for various platforms.[1] Because of Subversion's well-documented APIs, many third parties build tools on top of the Subversion libraries that integrate with other platforms and environments. While not directly involved in the release process, the feedback from these consumers helps validate API consistency between releases, and provides important testing during the validation of a potential release.

2.2 Versioning Guidelines

Subversion has adopted the "MAJOR.MINOR.PATCH" release numbering strategy, similar to that used for the Apache webserver [8]. The version numbers allow users to know what compatibility guarantees they can expect between different releases.

[1] A recent informal poll at a meeting of Subversion users indicated that not one in a group of over sixty professionals used the source packages as provided by the project. When deploying Subversion, these users all relied on third-party packages.

All releases with the same MAJOR.MINOR numbers are considered part of the same release *series*, with MAJOR.MINOR.0 being the first release in the series. Subsequent releases within the series are considered patch or bug fix releases, and the project guarantees that several important parameters, such as APIs and on-disk working copy database formats will not change. Thus, users and API consumers can know that interfaces will stay consistent between patch releases. New features are never delivered as part of a patch release.

Changes to the MINOR number result in a new feature release. These releases contain new features, and may change database formats on both the client and server. Features releases are promised to be backwards compatible, both in features and APIs, and work with old database formats. Newer releases can read and write older formats, but old releases are not guaranteed to be able to read newer formats (though often are able to).

In addition to code and database compatibility, all releases with the same MAJOR version number are compatible client-to-server. Older clients may not be able to take advantage of more advanced features in newer servers, but they will still be able to communicate. This compatibility is both forward and backward.

2.3 Release Procedure

For several years, Subversion has used a hybrid between feature- and time-based release strategies. Feature-based releases define particular releases to specific features, while time-based releases use strict timetables to determine release dates [9]. In Subversion's hybrid model, the developers would wait some amount of time, usually around six months, determine which features were completed or nearing completion, and use those to define the next release.

Several weeks prior to a new feature release, a release *branch* is created for that release. This branch is a snapshot of the main development branch, *trunk*, and is used for bug fixing and stabilization prior to release. This branch is ideally created at a time when *trunk* is considered stable enough for release, but frequently the need arises to perform additional stabilization on the branch prior to releasing.

To port fixes from *trunk* to the release branch during stabilization, committers nominate and vote on specific changes or groups of changes. A change must receive three positive votes from different committers to be approved for inclusion in the release. Any change may be nominated, but successful nominations are for changes which fix a known bug, increase performance in a non-invasive manner, or fix known API problems. Any committer may veto any change.

When the release branch is considered sufficiently stable, a release candidate (RC) is created from the branch. This release candidate is just that: a candidate for what will

eventually become the official release. The RC enters a period known as the *soak*, a four-week waiting period during which early adopters are encouraged to test the potential release. If no critical errors are found during the soak, a final RC is created, which eventually becomes the new feature release. If a critical error is found, the release manager publishes a new RC with the problems fixed, and restarts the soak period. Table 1 shows the historic times for creating Subversion feature releases.

Each RC, feature release and patch release goes through a validation process before being published. As mentioned before, committers thoroughly test the candidate using the included unit and regression test suites, and upon successful completion, cryptographically sign the release artifacts. In addition, enthusiastic users are invited to test the candidate tarballs and provide feedback, but their testing is not counted toward the required number of signatures.

Committers test on the platform of their choice, but the project requires three signatures from testers on both POSIX and Windows platforms, in addition to that of the release manager. When these signatures have been collected, the release manager uploads the release tarballs to the distribution server and publicly announces the release. For each release, the project distributes source code in `.tar.gz`, `.tar.bz2`, and `.zip` formats, a set of dependencies in the same formats, and the signatures generated as part of the validation process.

After the feature release is published, committers continue to nominate and port candidate changes to the release branch. When a large enough group of bug fixes accrues, a new patch release is issued from this branch, following the same pattern as creating a RC, including committer testing and signature collection. This process may be expedited for serious bugs or regressions.

3 Releasing Subversion 1.5.0

Following the Subversion 1.4.0 release in September 2006, the developers turned their attention to the next feature release, Subversion 1.5.0. Subversion had largely fulfilled it's goal as a replacement for CVS, and the developers started looking for ways to further enhance the feature set. The project needed direction, and found it in merge tracking.

3.1 Merge tracking

Merge tracking was defined within the project as keeping track of which changes occurred on which branches and how these changes have been applied, or *merged* to additional branches. In Subversion 1.4 and earlier, Subversion required users to manually track this information, which proved tedious and error-prone. Individual users, as well as corporate customers wanted Subversion to track this information and use it when performing merges between branches. The developers decided that merge tracking would be the defining feature for Subversion 1.5.0 [3].

Work on the merge tracking feature began on a feature branch, a copy of *trunk* used to implement potentially destabilizing features. Feature branches are useful in isolating incomplete or broken code from unwitting developers, but have the drawback that code on the branch is not as well reviewed or tested. Six months after creation, the merge tracking branch had grown quite complex, but had not yet been merged back to *trunk*.

Several months after merge tracking was started, in March 2007, several developers proposed releasing currently available features in an intermediate feature release, prior to releasing merge tracking. However, the community felt that merge tracking was close to completion, and that any effort spent creating and stabilizing an interim feature release would further delay this feature. Shortly after this decision, the merge tracking branch was merged to *trunk*, and the developers felt that Subversion 1.5.0 would be released by September 2007.

The complexity of merge tracking also hindered development efforts. Only a small percentage of the development community was actively working on the merge tracking feature, and it had grown so complex that additional developers were hesitant to invest the time required to make meaningful contributions. As the release cycle progressed, many individuals knew enough about merge tracking to raise important concerns, but lacked the knowledge to solve them.

As the testing of merge tracking progressed, defect rates failed to stabilize, and the developers continued to work to increase performance. Additionally, the initial design was flawed, which required additional workarounds. Internal and external pressure mounted to create a release, in spite of the chaotic state of the code base.

3.2 From branch to release

Finally, after a couple of abortive attempts, the 1.5 series release branch was created at the end of January 2008. Fixes began to flow into the branch, leading to an initial alpha release on 22 Feb 2008. This release did not pass committer verification, and was quickly followed by a second alpha release on 29 Feb 2008. This was the first time the Subversion project had used the term “alpha” on a release, and both alpha releases contained a number of known issues.

While stability continued to increase, a discussion opened within the project about what to call the next prerelease. One faction wanted to proceed with an RC so the four-week soak period could start, while others, recognizing the bugs that existed were severe enough to prevent an actual feature release, wanted to be more conservative when

Table 1. Dates between feature release branches and releases

Release	Branch date	Release date	Time to first RC (days)	Number of RCs	Time to release (days)	Time from previous release (days)
1.0	19 Dec 2003	23 Feb 2004	63	1	66	N/A
1.1	10 Jul 2004	29 Sep 2004	4	4	81	219
1.2	04 Apr 2005	21 May 2005	1	4	47	234
1.3	28 Sep 2005	30 Dec 2005	7	7	93	223
1.4	05 May 2006	10 Sep 2006	27	5	128	254
1.5	30 Jan 2008	19 Jun 2008	69	11	141	648
1.6	16 Feb 2009	??	0	??	??	??

naming pre-releases. Eventually, the groups reached a compromise, and a beta release was followed by the first true RC on 7 Apr 2008. This was more than two months after the branch was created (see Table 1), an abnormally long time for branch stabilization for a feature release.

Unfortunately, the first RC had critical bugs, and itself was not officially published, nor were the second or third RCs. It was not until RC-4 was announced on 24 Apr 2008, nearly three months after the feature branch was created, that the official soak period began. Additional minor bugs were found and more RCs created, some of which were never published due to the near-immediate discovery of still more problems. As the soak period ended, third-party consumers found additional API bugs which required yet more RCs, often with less than a week of separation between them. Over the course of the process, the release manager created eleven separate RCs, five of which would never be released because they did not pass internal validation.

Subversion 1.5.0 was finally released on 19 Jun 2008. This release came after much debate and struggle within the community, but the developers decided to release even with known issues. The prevailing rationale was that postponing the release would do more harm than good, and existing bugs could be fixed in subsequent patch releases. After experiencing the marathon 1.5.0 release process, developers also felt it was time for a change in release processes.

4 Discussion

In the several months since the release of Subversion 1.5.0, and as the developers work toward the next feature release, they have identified several places where the 1.5.0 release process failed. The developers plan on using these observations to implement improvements when releasing additional feature releases.

4.1 Learn from the past

Despite the transparency of process and free exchange of ideas, open source projects can sometimes be slow to learn from the experiences of others. Sometimes this happens intentionally, but most of the time community members are either unaware of the problems other projects face, being too focused on their own work to notice, or convinced that the same problems are not at play in their own project. A combination of these factors played into the delay in releasing Subversion 1.5.0.

Software projects are notorious for being delivered late [11]. For example, the Emacs text editor released version 22.1 in June 2007, nearly 6 years after the previous feature release. The long development cycle and time between releases frustrated users, who were forced to download and build their own copy of the latest development sources just to have access to features that had already been included in the development branch for years, but were unavailable in the last official release. Developers also felt alienated by the unresponsiveness of the community leadership and frustrated by the long time between when code was written and when it actually shipped [6]. The Subversion community could have seen and worked to avoid frustration by releasing sooner, but did not.

Another project which faced problems similar to Subversion 1.5.0 was the FreeBSD operating system [1]. FreeBSD 5.0 languished in stabilization for several years as new features were added and stabilized in an ambitious development effort. As with Emacs, some of the Subversion developers were aware of the experiences of FreeBSD, but no one thought these same problems could apply to Subversion. Naïveté, ego or both prevented developers from learning from these mistakes in preparing Subversion 1.5.0.

4.2 Follow the process

Projects typically create guidelines to assist with the release process, and Subversion is no different. HACKING exists to bring order to the occasionally chaotic nature of open source development, and to help newcomers become involved in the project. Consistently following established guidelines can help a project create releases which are both timely and of acceptable quality.

In an effort to publish a release, *any* release, the release manager began putting out alpha and beta releases, without any formal definition of what they meant, and how those releases differed from typical RCs. When true RCs did start appearing, there was some question as to their quality; over the course of the release cycle, five of the eleven RCs were never published. The Subversion community did have a process, and attempted to follow it, but the process was not designed for such large features as merge tracking.

It wasn't until the developers neared the end of the soak period and actually started threatening to create the final release that API consumers and third-party distributors started seriously testing the RCs. This led to the discovery of another set of bugs, more RCs, and more schedule slippage in attempting to deliver Subversion 1.5.0.

The community was also unwilling to release code with known issues. No developer wants to release buggy code, but for most users, perfect code is nonexistent code if it has not yet been released. As the number of changes in subsequent patch releases attests, Subversion 1.5.0 did have many bugs, but none were showstoppers for the release, and most have been addressed in subsequent patch releases.

4.3 Time-based releases

Many projects, from complete GNU/Linux distributions to individual software packages, have adopted a time-based release strategy. The theory behind such a strategy is to keep the time between when a feature is implemented and when it is released beneath some known upper bound. We call this the "bus station philosophy": if a feature misses a release, the next will be along shortly, so the release should not be held up for any one feature. This type of process helps keep developers engaged in the project, and gives users the ability to plan upgrade cycles around known dates. It also helps developers plan their efforts to allow *trunk* to be in a branchable state when release dates approach.

This type of strategy should work well for Subversion in particular because of the large number of third-party distributors who rely on releases created by the project. These consumers' products often require extensive development and testing to incorporate features new in a Subversion release. During the Subversion 1.5.0 release process, it wasn't until the developers threatened imminent release that some third-party users started testing thoroughly. Having a well-publicized schedule helps these communities as they build their own products.

Creating "time-based" releases is not a panacea for ensuring a consistent release process. Planning releases becomes difficult in open source projects where resources levels are unknown and constantly shifting. When release deadlines approach, and features are not complete, the community has to make difficult decisions about removing features, letting release dates slip, or shipping partial features. In a consensus-based community, such as Subversion, making these decisions can be difficult, and require resources which detract from further development.

4.4 Defining releases independent of features

Early in the 1.5 release process, developers and other interested parties came to expect that Subversion 1.5.0 would include the much-hyped merge tracking feature. This feature was crucial for a number of potential adopters, and heavily desired by one of Subversion's corporate sponsors, CollabNet. As the development cycle continued, it became evident that merge tracking was a complex problem, and would take much longer than anticipated, but it continued to define the 1.5.0 release.

Since most of the merge tracking development was happening on a branch, *trunk* was still in a releasable state, and an intermediate feature release could have been created in early 2007. A number of developers floated this idea, but the community ultimately rejected it in favor of focusing on delivering a 1.5.0 release which contained merge tracking, estimated to be delivered by September 2007.

Instead of defining Subversion 1.5.0 as the release which would add merge tracking, the community should have examined which features already existed, and been satisfied with creating a release with those features. As a result of a delay in merge tracking, many other desirable features were also delayed, forcing users to run potentially unstable development sources to obtain those new features, much like Emacs users did during the period described in Section 4.1.

By defining features independent of releases, developers not only create the opportunity to release more frequently, they also constrain themselves to more modularly designed software. In the case of Subversion 1.5.0, the the merge tracking feature ended up being much larger than anticipated, and the community ill-equipped to handle it. As a result, instead of dividing and parallelizing the development effort on merge tracking, the developers forced themselves to deliver it as an atomic feature. This also increased testing complexity, further prolonging the release cycle.

5 Aftermath

In the aftermath of the Subversion 1.5.0 release, the developers realized policies and procedures, as well as project mentality, had to change. In response, they instituted a much more rapid cycle for the 1.5.x patch release series, and time-based feature releases. While these have only recently been implemented, response from both users and developers has been positive and the community expects more rapid and better quality releases in the future.

5.1 Patch Releases

In preparing the patch releases for the 1.5.x series, the Subversion community has been much more aggressive than previously, creating more frequent releases. They have noticed several advantages to this approach, some of which are similar to the increased frequency of feature releases, but on a micro scale. For instance, frequent patch releases help get fixes to annoying bugs and regressions into the hands of users faster, just as more frequent feature releases get new features to users quicker. Also, by publishing more patch releases, community testers more frequently test and sign releases, keeping these skills sharp.

We should caution about being *too* aggressive with patch releases, since creating new releases does require some amount of effort for testing, packaging, and deployment. However, the Subversion community feels that the current strategy of 1-2 months between patch releases provides sufficient benefits to balance these drawbacks. The current 1.5 series patch releases have come roughly 6 weeks apart, with the intervals growing slightly as the defect rate decreases.

5.2 Feature Releases

Additionally, at the 2008 Subversion Developer Summit, a group of the core developers committed to producing time-based feature releases. By giving contributors a set date for a release, they can plan development and stabilization efforts around known target dates. This helps keep release and development cycles consistent, which then helps avoid the frustration and confusion which impromptu and drawn out release cycles can lead to. The response has generally been positive; Subversion 1.6 branched in February 2009 with a planned March 2009 release.

5.3 Dividing Features

Subversion continues to improve and incorporate new features which have the scale and complexity of merge tracking. Two of these features, the next-generation working copy (WC-NG) [5] and tree conflict resolution [10, 4], are both being implemented in a modular fashion. This allows them to be released in a partial, but still useful state, creating smaller testing surfaces, and helping to better facilitate time-based releases. This approach also leads to features which are small enough to reduce the learning curve of new developers who wish to become involved.

6 Conclusion

In this paper, we discussed the Subversion 1.5.0 development and release cycle, where it went wrong and how it can be improved. We outlined the established Subversion processes, where the 1.5 series deviated from those processes, and how the Subversion community is working to improve their processes. Specifically, we discussed how time-based releases, learning from past mistakes, and feature modularity can all improve the release cycle and are currently being implemented today.

Releasing quality software in a timely manner is not trivial, but is essential to the continuing health of open source projects. We hope that both proprietary and open source projects can learn from the experience of the Subversion team in releasing Subversion 1.5.0.

7 Acknowledgements

The authors wish to thank the members of the Subversion community who reviewed this paper. C. Michael Pilato, Mark Phippard, Stefan Sperling and Neels J. Hofmeyr all offered constructive comments, as did Justin Erenkrantz. Paul Grisham assisted in proofing and editing the paper, along with many others. We also thank the anonymous reviewers for their constructive comments. This work was partially supported by NASA grant NNX08AC48G.

References

[1] Choosing the FreeBSD Version That Is Right For You. `http://www.freebsd.org/doc/en/articles/version-guide/`.

[2] Hacker's Guide to Subversion. `http://subversion.tigris.org/hacking.html`.

[3] Subversion Merge Tracking Notes. `http://subversion.tigris.org/merge-tracking/`.

[4] Subversion Tree Conflict Notes. `http://svn.collab.net/repos/svn/trunk/notes/tree-conflicts/`.

[5] WC-NG Design. `http://svn.collab.net/repos/svn/trunk/notes/wc-ng-design`.

[6] J. Corbet. Waiting for Emacs 22. `http://lwn.net/Articles/234593/`, 2007.

[7] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[8] J. R. Erenkrantz. Release Management Within Open Source Projects. In *Proceedings of the ICSE 3rd Workshop on Open Source Software Engineering*, May 2003.

[9] M. Michlmayr, F. Hunt, and D. Probert. Release Management in Free Software Projects: Practices and Problems. *International Federation for Information Processing*, 234:295, 2007.

[10] S. Sperling. Investigation of tree conflict handling in selected version control systems. BSc Thesis, 2008.

[11] M. van Genuchten. Why is software late? an empirical study of reasons for delay in software development. *IEEE Transactions on Software Engineering*, 17(6):582–590, 1991.