Experiences Mining Open Source Release Histories

Jason Tsay
Institute for Software Research
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
jtsay@andrew.cmu.edu

Hyrum K. Wright Dewayne E. Perry
Empirical Software Engineering Laboratory
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas, USA
hyrum_wright@mail.utexas.edu,
perry@mail.utexas.edu

ABSTRACT

Software releases form a critical part of the life cycle of a software project. Typically, each project produces releases in its own way, using various methods of versioning, archiving, announcing and publishing the release. Understanding the release history of a software project can shed light on the project history, as well as the release process used by that project, and how those processes change. However, many factors make automating the retrieval of release history information difficult, such as the many sources of data, a lack of relevant standards and a disparity of tools used to create releases.

In spite of the large amount of raw data available, no attempt has been made to create a release history database of a large number of projects in the open source ecosystem. This paper presents our experiences, including the tools, techniques and pitfalls, in our early work to create a software release history database which will be of use to future researchers who want to study and model the release engineering process in greater depth.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Management, Human Factors

Keywords

Release engineering, data mining

1. INTRODUCTION

Releases are a critical part of a software project, and are often the primary interaction an end-user has with the software. Creating useful releases is valuable, but also difficult, which leads many software projects to create, record and distribute release artifacts using a wide variety of methods and tools, which may evolve over time.

The history of an open source project can often be defined by the history of its releases, and a change in the release process often mirrors a change in the project itself. As projects grow and mature, their release processes change to reflect the changing needs of their user base. In large part, the process reflects the community [2]; by studying releases, researchers can gain insights into not only the release process, but also the history of an open source project itself.

The nature of open source software lends itself to data mining and analysis. Even though comprehensive datasets of open source projects exist [14, 7], they generally do not include release history information. These datasets contain source control, bug tracker, and mailing list statistics, but not release information. Release information is usually much more difficult to obtain, partly because it is not stored in any standardized format between projects, or even within projects. Additionally, as the release process changes, the method of recording this data may also change.

Most studies of open source release process [4, 18, 10] only examine a small number of projects, largely due to the lack of usable and normalized aggregate release data for a large number of projects. In planning a study of our own, we realized a much wider survey was required, and embarked on a project to survey and categorize the release histories of a much broader collection of open source projects. This paper presents our early experiences in collecting this data, the pitfalls we encountered and our further plans to develop and use this multi-project dataset to study the release process.

1.1 Release Engineering

Boardly speaking, *Release engineering* is the part of the software engineering process during which the release artifact(s) are produced. Many software organizations of sufficient size have release engineers or release engineering teams. Although the nomenclature may be common, the roles fulfilled by these groups, as well as the artifacts produced, are as varied as the groups themselves.

The artifacts created by release engineering may vary. Traditionally they have included executables, installers, libraries or source code packages. Newer service-oriented-software delivery paradigms have changed this model. Instead of publishing an artifact for end user consumption, users may interact with the software in a hosted environment, which then changes the way this software is released. These perspectives are not even discrete: artifacts and their corresponding release processes may exist anywhere along this continuum [15].

Whatever the artifact, the software must eventually be released, and this release process should be treated as part of the software development process. In traditional software development methodologies, such as the spiral or waterfall models, release engineering is usually considered part of the deployment and maintenance phases [1, 11].

Organizationally, many proprietary and open source software projects employ dedicated release teams which are tasked with building the final shipping product, very literally "engineering the release." The handoff between development and release teams may be a discrete step, or the separation between the two contexts may be more nebulous. As is the case with the types of artifacts produced, team composition exists along a continuum, rather than discrete categories.

2. MOTIVATION

Release engineering is an often-neglected part of the software development process, though an important part of the software life cycle. The overarching goal of our research program is to study the release engineering process and provide insight into release processes, particularly their faults and failures. In doing so, we aim to provide means to better detect and prevent failures, as well as recover from them. Our end goal being to significantly improve industrial release engineering practices.

Although the authors have significant practical experience with release engineering, our experience is limited to a small number of organisations and projects. To accomplish our research goals, we need a depth of understanding that only comes from studying a wide variety of release processes. By using a large number of open source projects, we ensure that we have a substantial basis for our insights and proposals. We can also apply the techniques learned from creating our release history database to data collection in proprietary and industrial systems.

2.1 Previous Work

In spite of the large corpus of open source data, no attempt, as yet, has been made to collect a comprehensive database of project release history. Fischer, et al. demonstrated the ability to automate release history collection for a single project [5], but for our research needs, we desire a generic approach to collect and record release history from a wide variety of projects. Due to its automated approach, the above study also had no method of recording certain metadata about the release, such as the type of release (beta, release candidate, general availability, etc.) or the label (i.e., version number). Many projects record and use this information in a variety of ways, and a comprehensive release history dataset should contain it.

The *Description of a Project* (DOAP) [3] standard contains facilities for recording and describing release information, but has seen limited acceptance by the open source community. Those projects that do publish DOAP information rarely include historical information about the project's past releases. Additionally, although we initially targeted open source projects, we want our framework to be extensible to proprietary projects as well, and DOAP appears unlikely to gain much traction in that domain.

This lack of standardization contributes another problem in mining release histories: how does a release mining tool automatically determine the metadata associated with a release? The approach used by Fischer et al. correlated version control activity and bug tracker history, but this does not reveal a large subset of external information about the release. Additionally, their approach was tailored to the Mozilla dataset they used, and is difficult to generalize across a large number of projects.

3. DATA COLLECTION PROCESS

The steps for building our release history database were three-fold. First, we selected the projects to initially target, using several criteria to get a broad picture of the open source landscape. Second, we collected the actual data, using a framework of parsers and some manual inspection. Third, we standardized and inserted the data into a database for later use. This section describes our method.

3.1 Project Selection

We began our survey by selecting a wide cross section of open source projects and categorizing their current status according to several criteria. As our goal is to focus on release history data, and not do a comprehensive analysis of open source projects in general, we made these categorizations quite broad. In classifying projects, we took a "common sense" approach, though our techniques were understandably subjective. Our aim was to create a release history database, not study how to classify open source projects.

We observe that the type of a project influences its release process. Large open source operating system distributions, such as Ubuntu Linux or Fedora likely have different release processes than stand alone projects or suites of developer tools. A relatively new project has different needs, and hence different release processes, than an established and mature software project. Choosing a wide cross-section of open source project types and sizes allows us to create a dataset useful for studying release processes generally, while observing trends on a macro scale.

Due to the large number of open source projects and the pitfalls associated with choosing a good dataset [8], we felt it best to only focus on existing active open source projects. These projects have demonstrated an ability to create meaningful releases, and have also existed for sufficient time to create meaningful histories.

Using only these successful projects, however, taints the data by survivorship bias. To overcome this bias, we may expand our release history database in the future to include failed projects, but at present feel that our efforts are best focused on successful projects.

3.2 Data Collected

In choosing which data to collect, both about projects and their releases, we explicitly chose factors which we believe have an impact on the release process, such as the age and type of a project. We did not collect extensive data on the projects themselves, as such information already exists in the research community. Such information as issue tracker statistics, mailing list information, and version control history is not directly pertinent to our study, and can be readily obtained elsewhere.

We characterized each project according to its maturity, the type of project it is and the version control system. This

Metric	Notes
Date	Date the release was published
Type	The type of the release (e.g., *stable* or *testing*)
Label	Release identifier, often a version number
Source	Source of the foregoing information

Table 1: Release data collected

may appear like a small number of degrees of freedom, but we plan to eventually cross reference our list of projects with existing open source project information (such as FLOSSmole) to take advantage of the work already done by other researchers. For the project maturity, we had three values (*mature*, *adolescent*, and *young*), while the other axes contain a variety of values.

For each release, we collected the following data: the project it belonged to, the date the release was published, the type of release, the release label (version number) and the source of the data (see Table 1). Even though this is a small subset of possible data for each release, our initial work focuses on the release histories of projects, not a comprehensive look at all information available for each release. We have included sufficient information in our data collected to allow us to easily find and record additional release information, if desired.

3.3 Collection Mechanism

We chose to store our data in an SQLite database [12], with only two tables, one for the projects and one for the releases, indexed by project. SQLite is a light-weight database tool which allows our data be stored in a structured format, but also transported to other SQL-based systems. SQLite readily interfaces with a number of programming languages, which enables us to import the data into other software for more detailed analysis.

To simplify the data collection and aggregation, we created a set of Python scripts to first collect and then record the information. The information gathering scripts are specific to each project and are implemented as a set of Python modules, which feed into a central module to normalize and store the data.

Our scripts that collect the information follow a three-step process of parsing information sources for data, formatting and categorizing the data, and inserting the formatted release history into the database. Due to the lack of standardization in recording release information, the parsing and formatting scripts are specific to each project.

The first step for a project is to find adequate information sources for a project's release history and then to parse these sources. In general, we were interested in obtaining the following information for each of a project's releases: the release label, the corresponding release date, and any notes regarding the release itself, like changelogs. These notes are used later to help determine release types.

When we started looking for release history, we mostly used changelogs and web pages as our primary sources of release history. These sources, while useful, also tend to wildly vary in format, requiring different parsing mechanisms for each project. We were able to follow a general pattern: by using regular expressions to identify patterns such as HTML tags, we looked for release labels and the corresponding release date. The release notes collected during this process are also collected and aggregated for later use.

Eventually, we discovered some sources that were more standardized, like mailing list archives, source code repositories, and SourceForge download listings. The parsing techniques for these sources needed to be a bit more sophisticated. Relevant emails in the mailing list archives were identified by using Python's HTML modules to traverse the archive, looking for email headers with certain keywords. By using libraries to mine source code repositories, we were able to automatically traverse code repositories and find artifacts such as tags of older releases. Many projects upload all or most of their releases to their hosting service, such as SourceForge, so scraping a project's complete file listing web page yields a fairly complete release history. These three sources have the advantage of being fairly standardized, allowing for the same script (with some tweaks) to be used across multiple projects.

The drawback of this technique becomes apparent when trying to obtain older release history information. Unfortunately, the aforementioned sources are not always complete in regards to release history. Code repositories, especially if the project has moved from one type of repository to another (such as CVS to Subversion), will often omit early releases. Sometimes, early releases were simply never stored in the repository in the first place. Likewise, some projects switch hosting services well into a project's development, and in the move to the new hosting provider, many projects may neglect to upload their earlier releases. In some cases, the "final" release is the only artifact uploaded to the hosting service, but we were interested in pre-release history as well. For instance, if a release contains a serious bug or vulnerability, the resulting bugfix release will be the only one uploaded.

After parsing the data, the next step is to standardize the data acquired and to determine the metadata, including type, of every individual release. Primarily, the release dates needed to be standardized for the database. Because version numbering schemes vary across projects, there is little in the way of formatting that can be done for the release labels.

One important step in the formatting process is to categorize each of the releases in terms of what kind of release it is. We decided early on that we would divide releases into three categories: *prerelease*, *feature*, and *bugfix*. While prereleases such as beta and release candidate releases are fairly easy to identify due to their release labels, determining whether a release is a feature or a bugfix is often surprisingly difficult. To do so, we generally fell back on the release notes or the version number associated with the release.

Some projects, such as Ubuntu, have a version numbering scheme that makes it very easy to determine the purpose of a release. Other projects, such as the Linux kernel, are also very well-documented in terms of what the version number actually means. These kinds of projects were very easy to programmatically determine the types of their releases.

Many projects, however, especially smaller or less mature projects, do not use a version numbering scheme that reflects the type of release. Due to this lack of a consistent scheme, manual analysis of the release notes was often required in order to determine release type. This required a bit of subjectivity in order to determine what exactly is or is not a feature. These two issues make programmatically determining release types very difficult. One automated process used to assist in determining release type was to look for certain keywords in the release notes, but this process still required a manual check afterwards to verify accuracy.

Due to the Python SQLite module, inserting the final data into the database was an easy task. The database insertion module proved to be the only module that we were able to use across multiple projects, since the proper normalization occurred in the previous step of the process.

Project release history recording mechanisms change over time, but for mature projects, we anticipate the method of recording various releases will remain stable. Thus, updating the release database for future project releases should be possible with our existing scripting infrastructure.

4. LESSONS LEARNED

As we started our survey, some of the problems in recovering the data became apparent, and we began to understand why such a survey had not yet been attempted. In particular, the lack of standardization, as well as incomplete project release information, was problematic.

4.1 Lack of Standardization

In the course of doing our own data collection, one of the reasons for the lack of release history information became obvious: traditional project information such as bug and source histories are stored in databases which are inherently structured. The structure of these systems allows automated tools to easily parse and store the information.

In contrast, release information is scattered across a wide variety of sources, and largely varies between projects. During the course of our analysis, we found release history information in mailing list archives, source code repository histories and structures, web pages, or simply release artifact listings. Even within the same project, older history and newer history was frequently stored in separate locations.

As stated earlier, many projects do not use a standard version numbering scheme. More specifically, many projects do not label their releases in a way that facilitates in determining the purpose of the release. This makes it difficult to determine programatically the type of a release, often requiring manual and oftentimes subjective analysis.

One place in which release history information is standardized is project package management systems. Systems such as RPM and the FreeBSD Ports system contain a large number of software releases [16], but the majority of these are focused on distributing the latest version of the software, not maintaining a complete history of the upstream software system. Additionally, the release presented through these systems often has additional patches or modifications, which do not faithfully reflect the upstream release artifact.

4.2 Incomplete Histories

The fluid nature of open source development also adds confusion. As projects fork and merge, such as gcc [6], it becomes difficult to track the complete release history of the project. In instances where no definitive source was found, we were forced to make subjective decisions about what artifacts are part of the true release history of a project.

Although many projects maintain a release history for their more mature releases, oftentimes earlier releases are not as well-documented. Often the only artifact that still exists of older releases is the release label, important information about the release itself such as the date or release notes about the release are not documented. In some cases, older releases are simply lost such as many of the older Linux kernel releases [9]. In other cases, when a project moves to a new code repository or to new hosting service, such as SourceForge, earlier releases will often be excluded from the migration.

Some projects also fail to completely document their prereleases, especially after the “official” release. For instance, the Ubuntu wiki, although generally well-documented in terms of release history, leaves out information for prereleases in certain releases, such as in the Ubuntu 6.06 (Dapper Drake) release schedule [13]. The “Flight” prereleases have been left out of the release schedule, unlike in the more recent releases. There are other cases in where a feature release may have a serious vulnerability or bug, then the resulting bugfix release is the only release well-documented.

4.3 Generalizability

The current work focuses on open source software, due to the ease of gathering the initial data, but we hope the techniques can be applied to proprietary software releases as well. Doing so will provide a more balanced view of release history and processes, helping to remove some of the biases prevalent in current software engineering research [17].

We fear, though, that such a task will present several difficulties. Proprietary can have many more non-technical influences that open source projects lack, such as marketing and branding requirements. These influences may change the external label of a release, while internal (e.g., non-public) versioning may progress differently. While these issues are not unique to proprietary software releases, these extra branding factors are more prevalent in that domain, and can make automatically mining release history information very difficult.

For example, throughout its 25-year history Microsoft Windows progressed through a series of version numbers, followed by year-based labels, and then brand names, eventually returning to version numbers. The latest release of Microsoft Windows is external version 7, but internally labeled version 6.1. Deciding which version label to use, and then automatically retrieving these labels would be a difficult task for our release history miner.

The reclusive nature of proprietary software projects can also make them difficult sources to pull data from. While this problem is not necessarily new, it does make automatic release history gathering difficult, because some, but not all, of the information may be public. This leads to a subtle data biasing problem, which is difficult to overcome. Even defining what a is a “release” can be difficult, as artifacts may span the continuum between automatically-generated daily builds for internal use to semi-public prereleases intended for a limited audience, to the final publicly-consumed release.

In general, our framework for sorting and categorizing releases should be workable with proprietary systems, but we fear that the actual data collection may prove infeasible.

5. CONCLUSION

Releases form a critical part of the overall history of a project, and represent the cumulative efforts of the project developers. This information is simply too useful to overlook in developing a holistic view of an open source project. However, no comprehensive release history database yet exists in the research community, and while undertaking the task to create one, we discovered the reasons why this has not yet been attempted.

We conclude that programmatically creating a release history database from existing open source data is not trivial, though with proper per-project tooling specialization, we were able to accomplish it for a reasonable and useful number of projects. Furthermore, standardizing the tools and methods by which projects create and record their releases would make analyzing historical records much easier.

We have currently collected 1579 distinct releases from 22 different open source projects, though we have not yet analyzed our data completely, we expect that such a wide range of releases is a good start toward better understanding the release engineering process and how it can be improved. As our dataset matures and becomes more comprehensive, we anticipate releasing it, as well as the collection mechanism to the research community for further use.

5.1 Future Work

This paper briefly outlines our experiences in collecting data to create a release history database from open source projects. This data has many possible uses, but our immediate goal is to use it to learn where release process faults and failures occur, and improve the ways used to predict, prevent, and recover from these faults.

6. REFERENCES

[1] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[2] M. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.

[3] Description of a Project. `http://trac.usefulinc.com/doap`.

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

[5] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*, page 23, Washington, DC, USA, 2003. IEEE Computer Society.

[6] A Brief History of GCC. `http://gcc.gnu.org/wiki/History`, 2008.

[7] J. Howison, M. Conklin, and K. Crowston. FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, 2006.

[8] J. Howison and K. Crowston. The perils and pitfalls of mining SourceForge. *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*, pages 7–11, 2004.

[9] Linux Kernel Version History: Consolidated list. `http://www.oldlinux.org/Linux.old/docs/history/Master.html`, 2002.

[10] M. Michlmayr. Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management. In *Proceedings of the First International Conference on Open Source Systems*, pages 309–10, 2005.

[11] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, pages 328–338. IEEE Computer Society Press Los Alamitos, CA, USA, 1987.

[12] SQLite Home Page. `http://sqlite.org/`, 2010.

[13] DapperReleaseSchedule. `https://wiki.ubuntu.com/DapperReleaseSchedule`, 2008.

[14] M. Van Antwerp and G. Madey. Advances in the sourceforge research data archive (srda). In *Fourth International Conference on Open Source Systems, IFIP 2.13 (WoPDaSD 2008)*, Milan, Italy, September 2008.

[15] A. van der Hoek, R. S. Hall, D. Heimbigner, and A. L. Wolf. Software release management. *ACM SIGSOFT Software Engineering Notes*, 22(6):159–175, 1997.

[16] A. van der Hoek, R. S. Hall, D. Heimbigner, and A. L. Wolf. Software release management. *SIGSOFT Softw. Eng. Notes*, 22(6):159–175, 1997.

[17] H. K. Wright, M. Kim, and D. E. Perry. Validity Concerns in Software Engineering Research. In *Proceedings of the Workshop on the Future of Software Engineering Research*, November 2010.

[18] H. K. Wright and D. E. Perry. Subversion 1.5: A Case Study in Open Source Release Mismanagement. In *Proceedings of the ICSE 2nd Emerging Trends in FLOSS Research and Development Workshop*, May 2009.