Quality and the Reliance on Individuals in Free Software Projects

Martin Michlmayr
Department of Computer Science
and Software Engineering
University of Melbourne
Victoria, 3010, Australia
martin@michlmayr.org

Benjamin Mako Hill
Hampshire College
Amherst, Massachusetts, USA
mako@debian.org

Abstract

It has been suggested that the superior quality of many Free Software projects in comparison to their proprietary counterparts is in part due to the Free Software community's extensive source code peer-review process. While many argue that software is best developed by individuals or small teams, the process of debugging is highly parallizable. This "one and many" model describes a template employed by many Free Software projects. However, reliance on a single developer or maintainer creates a single point of failure that raises a number of serious quality and reliability concerns – especially when considered in the context of the volunteer-based nature of most Free Software projects. This paper will investigate the nature of problems raised by this model within the Debian Project and will explore several possible strategies aimed at removing or deemphasizing the reliance on individual developers.

1. Introduction

In the last few years, the Free Software and Open Source development models have established themselves as viable alternatives to proprietary software development structures. In fact, the suggestion that the quality of Free Software projects meets or exceeds that of proprietary alternatives is heard increasingly frequently and is becoming increasingly difficult to deny [12, 9, 11]. With his now famous mantra, "given enough eyeballs, all bugs are shallow", Eric Raymond was able to highlight the important role that highly parallelized development plays in the creation of high quality Free and Open Source Software [10]. However, software development does not benefit from parallelization in the same way that debugging does. It is a well-known fact that merely adding further programmers to a project will neither improve its quality nor shorten the release cycle [5]. In fact, the reverse appears to be the case as studies have confirmed that most Free Software projects are in fact designed and developed by very small teams or an individual developer [7].

These individuals and small teams benefit from the efficiency that is a direct product of the increased level of intra-project communication ensured by the low level of complexity in decision making structures. However, this centralized development model creates a single point of failure with serious quality assurance implications. These are compounded by the fact that most Free Software development is performed by volunteers who cannot be relied upon for consistent levels of work in the same way that non-volunteers in most commercial projects can [7]. As observed previously, the majority of Free Software projects are directed by a single "lead developer" – usually the software's original author – who assumes the role of "maintainer" and who integrates patches submitted from developers in his or her user community and who organizes and coordinates releases of the software. As a result of the centralized decision making structure, development stalls in the absence of the lead developer. These types of developmental delays and the abandonment of popular Free Software projects with established user communities are remarkably common. Working to predict or prevent these abandonments is difficult. The reasons why a developer might temporarily or permanently cease work on a project are as unpredictable as they are multifarious; they might be overwhelmed with work in their "real life" profession, they might have lost interest in the program or their priorities might simply have changed. To compound matters, distinguishing a temporary sabbatical from a full abandonment is often difficult. Developers often cling to a project they are emotionally involved in by urging their user and developer community to be patient but never find the time.

In the following analysis, we will evaluate the Debian Project's approaches to centralized and individualized project maintenance as a quality assurance issue. We realize that replacing lead developers with teams or expanding the size of existing teams is an unrealistically simplistic solution: it introduces complexity and related intra-project communication issues. It also works to counter the benefits which have made Free Software community's "one and many" development and debugging system so effective. We realize that in many ways, Debian is a unique Free Software project. However, because Debian is so large and pulls developers from across the world and the full spectrum of the Free and Open Source Software community, we believe that lessons learned from Debian can be generalized and applied to Free and Open Source Software development more broadly.

2. Debian and its Package Maintainers

With over 10,000 packages, the Debian Project offers the largest GNU/Linux distribution available [8]. As Debian serves largely as an aggregation of existing software, its developers' primary task is integration. Applications which conform to the Debian Free Software Guidelines [3] are pulled in from the community (from so called "upstream" developers) and integrated as "packages" into the Debian system according to an established set of policy guidelines [4]. In the majority of cases, an individual is responsible for each Debian package – a classification that usually corresponds to one application. The meta-information of each Debian package contains a "Maintainer" field which lists the name and e-mail address of the individual in charge of the software. As a result of this hard-coded dependence on individuals, the problems related to the reliance on individual developers described in the introductory section apply to Debian and upstream developers similarly.

At the moment, a sense of ownership accompanies the maintainership of a package. In many cases, maintainers have a strong attachment to their packages and become upset when another Debian developer intervenes in their job of maintaining the software, frequently resulting in heated exchanges on mailing lists. However, since Debian is a volunteer-based effort and a highly integrated and coordinated system, the project has instituted mechanisms to allow developers to assist each other with their packages when required. Hence, Debian has introduced the concept of Non-Maintainer Uploads (NMUs). As the name implies, an NMU is an upload done by a person who is not the maintainer of the package. The Debian Developer's Reference [2], a document describing the recommended procedures and available resources for Debian developers, has a whole section devoted to NMUs. It does not simply describe how NMUs are done, but also *when* they are appropriate. The Developer's Reference suggests that NMUs are especially justified for security fixes and during the preparation of releases when a packages maintainer appears to be inactive or inaccessible. The reference urges that special care be taken when doing an NMU because the person preparing the NMU might not be as familiar with the package as its maintainer. It is extremely embarrassing and irresponsible to break a package during an NMU.

While the mechanism for performing Non-Maintainer Uploads has been in place for a very long time, the perception of NMUs has changed over time. Currently, many developers interpret NMUs as a sign that they are not performing their work properly. While this is not always the case, this perception is widespread among both users and developers. In the past, this element of stigma was not connected to NMUs. As a result, developers were welcome, even encouraged, to perform NMUs – an attitude that some developers continue to hold. The acceptance of NMUs demonstrates a practical understanding of and approach to Free Software. It is not always possible for a volunteer to perform their duties consistently and thus it should be appreciated when someone lends a needed hand. The cause of shift toward a view of NMUs as more remedial and penalizing is unclear. One plausible analysis connects this shift with the great growth of Debian. While the project plateaued early on at around 200 Debian developers, the group has exploded recently to nearly 1000. Consequently, it is no longer possible for developers to know, or even know of, each of their co-developers. This fact alone has had a radical effect on the nature and self-perception of the Debian community. It stands to reason that many developers will find NMUs by developers that they know more acceptable than those by developers they are unfamiliar with. Even though the person performing an NMU might be competent and exhibiting good technical judgment, the act of a "stranger" making uploads of a developers *own* packages can more easily be perceived as a violent action.

Since Debian's size is only likely to increase, the project has been forced to find a mechanism to allow maintainers to explicitly specify an individual or group they find trustworthy and who can step in as a "backup maintainer". Such a mechanism has been introduced into the project recently in the form of an added field in a packages "control" file. While the meta-information of a package still contains a "Maintainer" field, there is now also an optional "Uploaders" field. While it is still possible for everyone to do an NMU of a specific package, an upload done by a developer listed as an uploader for this package will be treated as a regular upload rather than an NMU. Specifying explicit backup maintainers is useful and increasingly encouraged because the QA or security team has a list of competent or experienced developers to contact when the package's primary maintainer is unreachable.

2.1. Explicit Group Responsibility

In the solution described above, there is still a primary individual maintainer and the "Uploaders" act more in the role of backup. However, its often useful, both in routine package maintenance and for quality and reliability reasons, for these secondary maintainers to take a more active role. That is, there can also be a team of maintainers for a specific package. This type of maintenance is particularly advantageous for larger and more important packages. For example, packages in Debian's central "base" system, which are installed on every Debian machine and which include essential programs like GNU tar or Debian's package maintenance system *dpkg*, are likely to get a high number of more imperative bug reports than less frequently used packages. Consequently, it is often beneficial to have more than one developer working on triaging the bug listing. However, distributing the work among several developers is not constructive for every task of the package maintainer. It has been suggested that QA work requires less coordination than the actual development [12]. Thus, the task of maintaining the bug listing and fixing bugs might be shared between several developers, while an individual might be explicitly responsible for packaging the fixed software and making timely uploads.

While certain obvious advantages flow from team maintenance and development, the model is not without its disadvantages. For example, by challenging conceptions of ownership, groups might decrease maintainers' feeling of responsibility for their packages. While a single developer creates a single point of failure, he or she is also a singly clear responsible party. When a package is maintained by a group, members of the team might postpone important work because they assume that someone else will do it. Furthermore, decreasing the attachment maintainers often have to their packages with the goal of facilitating NMUs tends to decrease the sense of ownership that can act as a useful motivating factor. In these situations, finding an appropriate balance is crucial. In doing so, it is essential to remember that the questions of responsibility and motivation are questions that are tightly linked to issues of personality and vary greatly among developers across and within cultures. This is particularly evident in diverse international projects like Debian.

2.2. Facilitating Group Communication

Unfortunately, moving from individual to team maintenance results in an inevitable increase in communication complexity [5]. Therefore, it is crucial to provide solid mechanisms for increasing the effectiveness of intra-group communication before creating or augmenting teams [6]. One useful method is to carefully and critically limit the type of tasks that will be shared and the types of tasks that will remain in the domain of individuals. For example, QA work, especially reproducing bugs and getting more information from the bug submitters, can be distributed fairly well, while this parallelization is much more difficult to accomplish for development or design. By taking the pressure off of lead maintainers in parallizable areas, maintainers are given more time to devote to design issues and his or her own irreplaceability is tempered.

Toward these ends, there must be established and efficient systems for communication between group members. Since like most high-profile Free Software projects, Debian development is not tied to a particular locality, communication usually occurs through e-mail and IRC. Additionally, Debian Project correspondence in regards to bug reports are handled through Debian's Bug Tracking System [1]. Due to the fact that the Bug Tracking System (BTS) is universally accessible and publicly archived, team members can easily follow work done by other members of the group. A useful mechanism built into Debian's BTS and other popular Free Software bug tracking systems like Bugzilla allows people to subscribe to bug reports for a specific package. In this way, users and team members receive all correspondence by e-mail and can easily track packages without visiting a web site. In addition to team members, this functionality provides a mechanism for interested users and upstream developers to become involved with their software's Debian package and to stay informed of bugs. When an upstream issue (as opposed to a Debian specific issue) is reported, they can follow up directly and provide patches. This serves to facilitate more active collaboration between a package's Debian maintainer and upstream maintainer in a way that is mutually beneficial. The upstream developer benefits from the direct link to users testing their software while the Debian maintainer profits from having the upstream developer's input on resolving difficult bugs.

2.3. Case Studies and Examples

Since the relatively recent introduction of the collaborative mechanisms described above, numerous Debian and upstream maintainers have taken advantage of them. The following three examples are representative of the importance and experiences with these systems.

In the first example, the upstream maintainer of GNU Privacy Guard (GnuPG) subscribed to Debian's GnuPG package in the BTS. As a result, he was better informed of bugs in his software and was pleased to find that he could leverage the Debian infrastructure as a platform for users to test his software. Furthermore, he responded to bug reports when they pertained to upstream issues. He also responded to feature requests through implementing the functionality upstream (therefore also making the features available to non-Debian users). Finally, it is worth noting that this team-maintenance was staged in a manner that allowed for a firm division between the Debian and the upstream maintainers. In only a handful of occasions did both maintainers respond to a bug report simultaneously. As a result, the two maintainers have been able to carry out their tasks more efficiently in the context of an increased communication effort.

A similar second example involves GNU maintainer Paul Eggert who is responsible for a number of important GNU programs including tar, gzip, bison and others. Paul has started to scan the Debian Bug Tracking System for upstream bugs and takes these bug reports into consideration when preparing new upstream releases. In fact, when he prepares a new release of one of his tools, he mails the Debian maintainer with a detailed listing describing which bugs are addressed in the new version. This is a perfect example of how the direct involvement of the upstream maintainer and a concerted effort at team-based work and good intra-project communication leads to a product of much greater quality.

The GNU C library provides a final example. While the GNU C library Debian package has been maintained by a single maintainer in the past, the task was daunting and difficult to perform effectively. Recently, a group of maintainers has been delegated responsibility for the complex and important package. In fact, the "Maintainer" field no longer list a specific maintainer, but a mailing list. This mailing list is open and public so even developers who are not dedicated uploaders of the *glibc* package can following the discussions and provide comments and patches. In the months that glibc has been under team maintenance, dedicated glibc maintainers have continued to independently commit patches to CVS. However, while a handful of people feel empowered to work on the package, no one feels responsible to coordinate or work toward releases. Alluded to above, this issue might be resolved by with the designation of a individual as responsible for coordinating and executing uploads.

3. Application

Many of the insights gained from Debian can be applied to other Free Software projects. While there are many projects with only one developer in a leadership role, the addition of secondary maintainers has advantages that should be clear from Debian's example. These secondary maintainers can act as "backup" primary maintainers and can assist in pushing out new releases when there are security related issues. Additionally, having a specified backup developer provides an obvious successor for a project whose primary developer becomes busy, missing, or otherwise unavailable. While it has been argued that the community has established good procedures for adopting projects which have been abandoned by their primary developer [10], specifying a successor has certain advantages. First of all, the original author can direct the future of the project by picking a successor they approve of. Furthermore, the backup maintainer will have access to the development infrastructure which will make any necessary transition easier. Not only should backup developers be able to make new releases, they should also be able to control the domain and web site of a project. It is not uncommon for a new developer to adopt a project but for the project infrastructure, including the project's domain name, to remain in the control of a previous maintainer, who has disappeared and can not be reached. The effects of such a situation can be disastrous but can easily be avoided by encouraging maintainers to appoint a backup maintainer and empowering them with full access and privileges.

4. Conclusions

The strong reliance on individual developers is a quality assurance consideration in that it is unrealistic to expect complete predictability and reliability from volunteers. Debian's experience has demonstrated that trusted and competent backup maintainers explicitly established by the primary maintainer can provide one successful template of dealing with this problem. Furthermore, these backup maintainers can play an active role in the daily upkeep of the software. For example, QA efforts, such as reproducing bug reports and following up with bug submitters, are highly parallizable and benefit from being dealt with by groups, especially when an important or complex package is involved.

Unfortunately, there are also downsides when moving from individual to team maintainership. The most obvious problem is an increase in complexity and the need for additional communication. Additionally, some maintainers feel a loss in responsibility and motivation and a tendency to wait for other team members to accomplish unpleasant tasks.

Because arguments for group maintenance of software has both recognized benefits and downsides, it is important to study the effects and experiences of group maintenance in Free Software projects. The three short cases analyzed imply largely beneficial results. It remains to be established how representative these packages are within Debian and how applicable their examples are to Free Software projects more generally. In any case, the potential for collaborative and group maintenance in successfully resolving a serious quality assurance issue is obvious and its importance and prominence in successful projects, in one form or another, seems like a good possibility.

References

[1] Debian Bug Tracking System. `http://bugs.debian.org/`.

[2] Debian Developers' Reference. `http://www.debian.org/doc/developers-reference`.

[3] Debian Free Software Guidelines. `http://www.debian.org/social_contract`.

[4] Debian Project Policy. `http://www.debian.org/doc/debian-policy/`.

[5] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, 2nd edition, 2000.

[6] G. N. Dafermos. Management and Virtual Decentralised Networks: The Linux Project. *First Monday*, 6(11), November 2001. `http://www.firstmonday.org/issues/issue6_11/dafermos/`.

[7] R. Ghosh, R. Glott, B. Krieger, and G. Robles. Free/Libre and Open Source Software: Survey and Study. Technical report, International Institute of Infonomics, University of Maastricht, The Netherlands, June 2002. `http://www.infonomics.nl/FLOSS/report/`.

[8] J. M. González-Barahona, M. A. Ortuño Pérez, P. de las Heras Quirós, J. Centeno González, and V. Matellán Olivera. Counting Potatoes: The Size of Debian 2.2. *Upgrade*, II(6):60–66, December 2001. `http://people.debian.org/~jgb/debian-counting/counting-potatoes/`.

[9] T. J. Halloran and W. L. Scherlis. High quality and open source software practices. In *2nd Workshop on Open Source Software Engineering*. ICSE, 2002.

[10] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly, 1999.

[11] Reasoning. How open-source and commercial software compare.

[12] D. C. Schmidt and A. Porter. Leveraging open-source communities to improve the quality & performance of open-source software. In *1st Workshop on Open Source Software Engineering*. ICSE, 2001.