Quality Practices and Problems in Free Software Projects

Martin Michlmayr, Francis Hunt, David Probert
Centre for Technology Management
University of Cambridge
Cambridge, CB2 1RX, UK
martin@michlmayr.org

***Abstract*— Free software and open source projects are often perceived to be of high quality. It has been suggested that the high level of quality found in some free software projects is related to the open development model which promotes peer review. While the quality of some free software projects is comparable to, if not better than, that of closed source software, not all free software projects are successful and of high quality. Even mature and successful projects face quality problems; some of these are related to the unique characteristics of free software and open source as a distributed development model led primarily by volunteers. In exploratory interviews performed with free software and open source developers, several common quality practices as well as actual quality problems have been identified. The results of these interviews are presented in this paper in order to take stock of the current status of quality in free software projects and to act as a starting point for the implementation of quality process improvement strategies.**

***Index Terms*— quality practices, quality assurance, quality improvement, free software, open source.**

I. Introduction

In recent years, the free software and open source model has established itself as a viable alternative to other development models. Many popular products, such as Linux, Apache and Samba, have been created according to open source practices, and the number of free software applications is increasing steadily. An empirical study of a large collection of free software conducted in 2001 found more than 55,000,000 lines of code, with an estimated value of almost US$1.9 billion according to the COCOMO model [4]; moreover, the system has more than doubled in size since then [5].

The free software model has not only led to the creation of significant pieces of software, but many of these applications show levels of quality comparable to or exceeding that of software developed in a closed and proprietary fashion [6], [14]. In his paper "The Cathedral and the Bazaar", Raymond suggests that this high level of quality is partly due to the high degree of peer review and user involvement often found in free software projects [13]. This hypothesis is in line with findings from traditional software engineering research, which have shown that peer review has a significant impact on the quality of software [3].

While Raymond's hypothesis sounds plausible, his model has yet to be validated and tested rigorously through empirical data analysis. In general, much of what has been said about quality in free software is based on anecdotal rather than empirical evidence. Given the recent success of many free software applications it can be assumed that, to some extent, the software fulfils the requirements and quality standards of users. Nevertheless, empirical work is necessary to study quality in free software and to contrast it with that of proprietary software. In particular, work on the identification of problems related to the free software model would be beneficial. Insights from such research can subsequently be used to propose process improvement strategies.

II. Background

Free software projects usually exhibit two characteristics that have important implications on quality assurance. First, they are distributed, in many cases all over the world. Second, the participants of free software projects are usually unpaid volunteers. The recent interest in free software, along with its deployment, clearly shows that successful and mature products for end-users can be built by volunteers in a distributed fashion. The free software development model offers certain advantages over proprietary models; these include the potential for substantial peer review and the attraction of excellent programmers from across the world. However, free software also faces certain challenges that are unique to this model. For example, due to the voluntary nature of free software projects, it is impossible to fully rely on project participants [11]. This issue is further complicated by the distributed nature because it makes it difficult to identify volunteers who are neglecting their duties, and to decide where more resources are needed [10].

While most research on open source has focused on popular and successful projects such as Apache [12] and GNOME [9], there is an increasing awareness that not all free software projects reach maturity, success or high quality. SourceForge, which is currently the most popular hosting site for free software and open source projects with over 95,000 projects, is not only a good resource to find well maintained free software applications – there are also a large number of abandoned projects and software with low quality [7]. Some of these abandoned projects may be explained in terms of a selection process given that more interesting projects with a higher potential will probably attract a larger number of volunteers, but it has also been suggested that project failures might be related to the lack of project management skills [15]. Nevertheless, large and successful projects also face important problems related to quality, such as the volatility of volunteer contributions [11], [10] and the inability of developers to keep up with bug reports [17].

In order to ensure that free software remains a viable model for the creation of mature and high quality software

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 24-28

suitable for corporate and mission-critical use, free software quality assurance and project management have to take these challenges and other quality problems into account and find solutions to them. As a first step in the development of practices and processes to assure and further improve quality in free software projects, existing quality practices and quality problems have to be clearly identified. To date however, only a few initial surveys on quality related activities in free software projects have been conducted [19], [20]. The rest of this paper will present results from exploratory interviews conducted with free software developers from a diverse range of projects. The insights gained from these interviews lead to a more complete account of the current status of quality practices and problems.

III. Methodology

For this study, exploratory interviews with seven free software and open source developers have been conducted. The projects were of a very diverse nature and it is believed that despite the relatively small sample size a fair representation of various kinds of free software projects is given. One project started at a university was aimed at giving undergraduate students the opportunity to get experience of a software development project. This project was later transformed into a community project in which students of this university participated with external contributors. Another project was part of research activities carried out by a large corporation, and one resulted from a university research project. The remaining projects originated as community projects. The size of the projects varied quite widely, from large projects such as Apache, GNOME and Debian, to smaller efforts such as VideoLAN and Dasher.

The interviews were of a fairly unstructured nature so as to allow a thorough exploration of quality in the projects in which the interviewees were involved. Questions, such as the following, were asked directly but the interviews remained open for the exploration of other topics:

- Are there any quality issues in your project? If so, what are they and how do you deal with them?
- What techniques can be applied in free software and open source projects to ensure quality?
- What kind of facilities does your project have to ensure quality?
- How would you compare the quality of open source and proprietary software? When might the quality in one be higher than in the other?

IV. Results

The results from the interviews will be presented in two sections; current development and quality practices identified in the interviews will be discussed, followed by a summary of quality related problems facing open source projects. Before these two areas are covered, the findings from the question concerning quality in open versus proprietary software will be discussed since it provides an overview of how quality in free software projects is perceived by their participants. While it is not expected that interviewees necessarily have an unbiased view regarding this question, it can shed light on the subjective perception of quality by free software and open source developers.

Few interviewees said that quality in either free software or closed proprietary software is necessarily higher. However, many participants felt that free software had a higher potential to achieve greater quality and can react faster to critical issues such as security bugs. There are various reasons for this. Firstly, its open nature promotes more feedback, which can be used to improve the software. Feedback can either be given in the form of bug reports or feature requests. Secondly, many participants felt that motivation was higher in free software projects because volunteers could work on whatever they wanted. Open collaboration with other developers and input from others also increase the motivation to work on a piece of software. Some interviewees thought that this increased motivation had positive effects on the quality of the software. Third, an interviewee suggested that free software can attract better human resources because of its distributed nature. He argued that free software "can benefit from a wider range of expertise and knowledge than a traditional software company can usually bring to bear on a problem". A downside of community projects compared to commercial development was the lack of resources and infrastructure. Finally, one interviewee responded that it is very hard to compare open and closed development models because of the opposing philosophy of these models. Since closed source companies often hide their defects and source code, it is hard to make a comparison. This is certainly an area where academic research can fill a gap.

A. Development and Quality Practices

One of the surprising insights gained was how greatly development practices and processes employed by differ across projects. In this section, a summary of practices found in the interviews will be given. While most projects employ some of these processes, few projects follow all of them. It is not clear why this is the case or whether projects would benefit from the introduction of more of these practices. An interesting observation is that everyone who was interviewed stressed the importance of processes and, in particular, of good infrastructure.

The identified practices can be categorized broadly into three areas:

1) Infrastructure
   Free software projects rely heavily on infrastructure that allows distributed development and collaboration. Important parts of infrastructure are:
   - Bug tracking systems (e.g. Bugzilla): these are used to capture feedback from users. They are often used to store both actual bug reports as well as feature requests.
   - Version control systems (CVS, SVN, Arch): these allow multiple people to work on the same code basis concurrently and keep track of who makes which changes.
   - Automatic builds (e.g. tinderbox): these make sure that the newest code in the version control system still builds. The test builds can be done on a number of different hardware or software environments.
   - Mailing lists: used for communication, both between developers and users.

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 24-28

While many projects make use of such infrastructure, the way they are used differs significantly from project to project. The different use of this infrastructure (i.e. the practices) may have important implications on project quality. For example, some projects employ a very rigorous policy through their automatic build system, halting further development until the defect has been removed. The practices related to the use of version control systems are also very diverse. In some projects, commit access is given immediately after the first contribution. In others, contributors have to work for months to gain respect from others, and there are also projects where only the lead developer has the right to commit code. The implications of these different practices on project quality need to be studied in further detail.

2) Processes
Free software development follows many processes but a large number of them are not documented anywhere – developers adhere to them implicitly.

- Joining: projects require prospective members to follow specific, mostly undocumented, procedures in order to join a project. These procedures vary considerably across projects. Some explicitly make sure to admit only contributors from whom high quality submissions can be expected while other projects are more liberal. One specific joining script has been described in the literature [18].
- Release: different release policies are employed by projects but many follow freeze stages. A feature freeze is the point when no new features are to be incorporated into the code base but there is sufficient time to fix bugs. More recently, string freezes have also gained popularity. They give translators of the software an opportunity to update their translations before a release is made.
- Branches: these are used to differentiate between versions of a program, for example by having a stable and a development branch. New development tools which make branches easier to deal with, such as Arch, have had a significant impact on the development process.
- Peer review: typically, changes made to the version control system are reviewed by members of the projects; though in most cases, this form of peer review is not very well formalized. Developers hope that other project participants will look at their changes but there is often no assurance that this is the case in reality. On the other hand, some projects have introduced very rigid procedures and expect code to be reviewed before it is committed.
- Testing: in order to ensure that a new release fulfils the standards of a project and that it has no major regressions, some projects have testing check lists. These check lists contain the most important functions and briefly describe how they can be tested. A release is only made when testers on different platforms have gone through the check list and have confirmed that the new version does not display major show-stoppers or regressions. Since few projects have automatic test suites, having a check list is a good solution to guarantee that at least major components and functions operate.
- Quality assurance: some projects organize bug days or bug squashing parties to triage their outstanding bugs. During this work, duplicate bug reports are marked as such, old bugs are reproduced, and bugs are also fixed.

3) Documentation
Many interviewees criticized their projects for a lack of documentation regarding development practices. Few projects have explicit documentation describing ways of contributing to and joining a project. In some cases contributed source code was checked by the lead developer of a project and then rejected because it did not conform to the coding style, a style that was not clearly documented anywhere. However, some projects, mainly those who have a large number of contributors, have good documentation.

- Coding styles: documentation aimed at developers which describes the style which should be used for the source code.
- Code commit: documentation which describes when and who can make changes in a project's version control system.

B. Quality Problems

The quality problems that have been identified in the interviews are of particular interest since they are largely issues which have yet to be solved in the free software community. While many of the practices described in the previous section have simply not been adopted by certain projects, good solutions for some of the following quality problems have as yet to be developed.

- Unsupported code: one of the unsolved problems is how to handle code that has previously been contributed but which is now unmaintained. A contributor might submit source code to implement a specific feature or a port to an obscure hardware architecture. As changes are made by other developers, this particular feature or port has to be updated so that it will continue to work. Unfortunately, some of the original contributors may disappear and the code is left unmaintained and unsupported. Lead developers face the difficult decision of how to handle this situation.
- Configuration management: many free software and open source projects offer a high level of customization. While this gives users much flexibility, it also creates testing problems. It is very difficult or impossible for the lead developer to test all combinations so only the most popular configurations tend to be tested. It is quite common that, when a new release is made, users report that the new version broke their configuration.
- Security updates: in many cases they are made in a timely manner but sometimes fixes are not available for weeks or months.
- Users do not know how to report bugs: as more users with few technical skills use free software, developers

see an increase in useless bug reports. In many cases, users do not include enough information in a bug report or they file duplicate bug reports. Such reports take unnecessary time away from actual development work. Some projects have tried to write better documentation about reporting bugs but they found that users often do not read the instructions before reporting a bug.

- Attracting volunteers: a problem some projects face, especially those that are not very popular, is attracting volunteers. There are usually many ways of contributing to a project, such as coding, testing or triaging bugs. However, many projects only find prospective members who are interested in developing new source code. Few contributors are interested in helping with testing or triaging bugs. As a result, developers have to use a large portion of their time for tasks other people could easily handle.
- Lack of documentation: it is possible that the previous problem is related to the lack of documentation. Volunteers may like to contribute in an area but they might not know how to start. Little help is given to prospective contributors and almost no documentation exists. The lack of developer documentation also implies that there is no assurance that everyone follows the same techniques and procedures.
- Problems with coordination and communication: in some projects, there are problems with coordination and communication which can have a negative impact on project quality. Sometimes it is not clear who is responsible for a particular area and therefore bugs cannot be communicated properly. There may also be duplication of effort and a lack of coordination related to the removal of critical bugs.

In summary, the interviews have identified a number of outstanding issues in some free software projects that have potential implication on project quality.

V. Discussion and Further Work

The interviews have given rise to a number of interesting insights about quality in free software. One striking observation is the diversity of practices that appear to be employed by free software practices. Based on the interviews, it would be hard to make the claim that there is one free software model that all projects follow. While projects share common attributes, there are clear differences in their practices. Since practices might have a different impact on project quality, the relationship between these practices and the quality of the resulting products needs further empirical investigation. This may allow the identification of best practices or show factors influencing the applicability of practices to particular projects.

The interviews have also shown that projects clearly face a number of problems. While all of the projects which have been taken into account for this study can to some degree be considered successful free software projects, there is definitely room for quality improvement. The problem of unsupported code is of high significance since the removal of existing functionality may impact users. On the other hand, it is not clear whether a volunteer project can guarantee that a particular functionality – or even the project itself – will continue to be maintained. This raises the more general question whether projects based on unpaid volunteers can not only deliver a high quality product but also ensure that high levels of quality can be maintained.

ISO 8402 defines quality assurance as all "planned and systematic activities" directed towards fulfilling the requirements for quality. Given the volatility of volunteer contributions and the unplanned nature of many free software activities, it is not clear whether free software projects can have a planned and systematic approach to quality, and as a consequence, whether they can ensure high quality. This question will become of increasing interest as free software is deployed in more corporate and mission-critical settings. In fact, some problems due to the lack of a systematic approach to quality have already been observed. While many important defects, especially security related issues, are fixed very quickly, often within hours, it was pointed out during the interviews that some remain unresolved after weeks or months. There is a great variance in defect removal time and more systematic quality assurance activities can contribute to timely fixes.

Some free software projects have recently realized the importance of quality assurance and have created dedicated teams. GNOME has performed various quality assurance tasks for a long time [17], Debian has a well established QA team [2], [10], and KDE started a quality effort in March 2004 [8]. These efforts largely focus on basic levels of quality assurance, such the removal of defects or components which are no longer supported. The next step is to move from ensuring minimum standards of quality towards the prevention of defects and the implementation of quality improvement strategies.

In order to study quality and quality improvement, tools and metrics are needed to actually *measure* quality. It is only possible to evaluate quality improvement techniques based on repeated quality measurements. Unfortunately, quality is a concept which is very hard to define precisely. Many people have an intuitive feeling of what quality entails, but when it comes to defining quality in a precise manner it turns out that the concept is very hard to pin down [1].

Quality is a concept which is made up of many different components [16]. What is a high quality product for some is not necessarily perceived the same way by others. For example, the user perception of quality may be different from that of software developers. This point has a high significance in free software since volunteer developers often write the software for themselves and they might therefore not take the requirements and quality standards of other, less technical users into account. In many volunteer projects, there is little intermediation between users and developers while commercial projects are clearly driven by the demands of paying customers. A good approach to quality will adopt a holistic perspective and take different requirements and quality attributes into account.

This study has offered some initial insights into where improvements are possible. Further research with larger samples is needed to investigate other quality related problems in free software projects. The implementation of quality improvement strategies also requires tools and metrics to measure and assess different aspects of quality. This is an area where synergies between the academic world and the

free software community may be possible. Further empirical studies investigating quality depend on the development of tools and metrics to measure quality. These tools could in turn be made available to free software developers so that they can assess the quality of their project and use the results as the starting point for the development of effective quality improvement strategies.

VI. Conclusions

This paper has presented insights into quality aspects of free software projects gained through exploratory interviews which have been conducted with free software developers. In addition to an identification of software development practices related to quality, a number of quality problems have been described facing free software projects. These insights can be used as the starting point for further investigations regarding quality in free software projects, along with the implementation of quality improvement strategies.

The significance of further research about quality aspects of free software has been shown. In particular, the question has been posed as to whether projects consisting exclusively of unpaid volunteers can ensure high levels of quality. Further research is suggested to identify other quality problems not found in this study and to evaluate the impact of different practices on project quality. Finally, it has been argued that tools are needed to measure and assess quality, both for academics to perform further studies and for free software developers to improve project quality, and that synergies might be possible in this area.

VII. Acknowledgements

This work has been funded in part by Fotango, the NUUG Foundation and the EPSRC.

References

[1] B. Dale and H. Bunney, *Total Quality Management Blueprint*. Oxford, UK: Blackwell Business, 1999.

[2] "Debian quality assurance team," http://qa.debian.org/, accessed March 31, 2005.

[3] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976.

[4] 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*, vol. II, no. 6, pp. 60–66, December 2001.

[5] J. M. González-Barahona, G. Robles, M. Ortuño Pérez, L. Rodero-Merino, J. Centeno González, V. Matellan-Olivera, E. Castro-Barbero, and P. de-las Heras-Quirós, "Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian)," in *Free/Open Source Software Development*, S. Koch, Ed. Hershey, PA, USA: Idea Group Publishing, 2004, pp. 27–58.

[6] T. J. Halloran and W. L. Scherlis, "High quality and open source software practices," in *Proceedings of the 2nd Workshop on Open Source Software Engineering*. Orlando, FL, USA: ICSE, 2002.

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

[8] "KDE quality team," http://quality.kde.org/, accessed March 31, 2005.

[9] S. Koch and G. Schneider, "Effort, cooperation and coordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.

[10] M. Michlmayr, "Managing volunteer activity in free software projects," in *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, Boston, USA, 2004, pp. 93–102.

[11] M. Michlmayr and B. M. Hill, "Quality and the reliance on individuals in free software projects," in *Proceedings of the 3rd Workshop on Open Source Software Engineering*. Portland, OR, USA: ICSE, 2003, pp. 105–109.

[12] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[13] E. S. Raymond, *The Cathedral and the Bazaar*. Sebastopol, CA: O'Reilly & Associates, 1999.

[14] D. C. Schmidt and A. Porter, "Leveraging open-source communities to improve the quality & performance of open-source software," in *Proceedings of the 1st Workshop on Open Source Software Engineering*. Toronto, Canda: ICSE, 2001.

[15] A. Senyard and M. Michlmayr, "How to have a successful free software project," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference*. Busan, Korea: IEEE Computer Society, 2004, pp. 84–91.

[16] I. Sommerville, *Software Engineering*. Essex, England: Pearson Education Limited, 2001.

[17] L. Villa, "Large free software projects and Bugzilla," in *Proceedings of the Linux Symposium*, Ottawa, Canada, 2003.

[18] G. von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Research Policy*, vol. 32, no. 7, pp. 1217–1241, 2003.

[19] L. Zhao and S. Elbaum, "A survey on quality related activities in open source," *Software Engineering Notes*, vol. 25, no. 3, pp. 54–57, 2000.

[20] ——, "Quality assurance under the open source development model," *The Journal of Systems and Software*, vol. 66, pp. 65–75, 2003.