The Economics of Open Source Software Development
Jürgen Bitzer and Philipp J. H. Schröder (Editors)

6

A Statistical Analysis of Defects in Debian and Strategies for Improving Quality in Free Software Projects

Martin Michlmayr and Anthony Senyard

ABSTRACT

It has been argued that free software and open source projects benefit from a high level of user involvement. Users influence the direction of a project through feature requests, and their identification of bugs and defects in the software contributes to high levels of quality. In this chapter we present a number of statistical analyses based on over 7,000 defect reports that have been recorded about the free software project Debian. These analyses are used to make observations on the effectivity of the open source development process used by projects such as Debian and lead to recommendations on how to improve this process.

Keywords: Quality improvement, user involvement, software development process, volunteer management.

JEL Classification: O32, L15, L23.

6.1 INTRODUCTION

Free and open source software has established itself as a viable alternative to proprietary software during the last few years. The initial definition of the open source development model was made by Eric S. Raymond in a case study of fetchmail (Raymond, 1999). Raymond concluded that the key aspect of the development model that leads to high quality and feature-rich free software and

open source is due to extensive peer review and user involvement in the development process. Indeed, the distinction between users and developers is fuzzy and the main difference is between users with and without technical expertise.

In the field of software engineering, peer review, such as inspections and walkthroughs, is known to significantly improve the quality of software at a relatively low cost (Fagan, 1976; Doolan, 1992). Free and open source projects utilize this knowledge by allowing all users to be involved in the development process. Obviously, not every user of free software has the technical skills to take part in code review or to carry out development. However, these users can contribute to the project by tracking down and reporting defects or by suggesting new features.

The value of defect detecting is simply enormous. The complexity of even simple software systems makes it impossible to detect all defects using conventional testing methods. However, this is not a problem if the defects which effect the operation of the software can be detected. Herein lies one of the greatest strengths of the open source development process, the recruitment of non-technical users to detect defects which effect the operation of the software system. In order to capture this valuable user input, many free software projects have introduced bug tracking systems that are used to capture defects and which allow users to request new features.

Although some of these bug tracking systems have been used to track and prioritize defects and feature requests for many years, there is little analysis of the recorded data. Analyzing the data is of substantial value since it will reveal how various variables connected to the reported defects, such as the defect removal rate, change over time. It also has the potential to help in the identification of the factors which contribute to a reduction of the reported defects.

In the following, we will analyze data recorded in the bug tracking system of one of the largest free software projects, Debian (González-Barahona et al., 2004). Specifically, we will base our analysis on more than 7,000 defects and feature requests that have been reported about Debian's base system in the 30-month period from April 2001 to October 2003. We will first investigate and summarize changes which have occurred in this time frame. Additionally, we will investigate which mechanisms are most effective at reducing the number of defects. The analysis will give insights about the quality of the defect tracking and removal process and will be used to suggest further improvements to the process.

6.2 BACKGROUND

Free software and open source projects are generally performed by dispersed volunteers, working together via the Internet. While there is an increasing number of projects that are primarily carried out by a company in a traditional environment,

the majority of free software projects are carried out in a distributed way. Due to this distributed nature, developers of free software projects depend on reliable communication and coordination mechanisms to perform their work effectively. Over the years, various software tools have been created to cater for the needs of such distributed development models. One of the most popular tools is the Concurrent Versions System (CVS) (Vesperman, 2003). CVS provides version control facilities and allows multiple developers to work on a piece of code in parallel.

6.2.1 Abundance of Empirical Data

One by-product of the distributed nature of free software projects is that the whole development history is very well documented. While free software developers often do not produce formal documentation, such as specifications, their work leaves abundant traces of the development process on the Internet: mailing lists that are used to communicate with both users and developers are publicly archived, the source code of the software is available for download, and projects often make use of a bug tracking system, such as Bugzilla (Villa, 2003).

This abundance of data, combined with the success of free software and open source projects, is a possible explanation for the high interest of researchers in open source. Various methods have been developed that allow automated analyses such as a counter for physical source lines of code (Wheeler, 2001) and tools to evaluate and interpret CVS and changelog data (German and Mockus, 2003; Capiluppi et al., 2003). Moreover, some quantitative analyses have been published, such as one about GNOME (Koch and Schneider, 2002) and one examining the code quality of various projects (Stamelos et al., 2002).

6.2.2 Defect Tracking in Free Software Projects

Even though there is an increasing number of quantitative studies about free software, little attention has been paid to the rich information stored in projects' bug tracking systems. This may be related to the fact that CVS is still better integrated into the development process than bug tracking systems. A survey of free and open source projects showed that more than 95% of (by their definition) large projects used version control systems such as CVS. At the same time, less than 80% of large projects and roughly 40% of tiny projects used bug tracking systems (Zhao and Elbaum, 2003). Another study showed that a higher percentage of projects use version control than bug tracking systems and that more successful projects incorporate such infrastructure in their development process than those which are unsuccessful (Michlmayr, 2005). Additionally, while SourceForge, the largest hosting site for open source projects at the time of writing, offers CVS as well as an issue tracking system, CVS is described much more exhaustively in their documentation.

Free software projects can greatly benefit from the use of a bug tracking system. Raymond suggested that the high quality and success of open source is connected to user involvement and peer review (Raymond, 1999). Although not every user has the knowledge to participate in the development or code review of an open source project, they can contribute in other ways. A bug tracking system effectively extends the peer review process typically performed on the source code to defect reports. Not only can users report defects and request valuable features, they can also comment on existing defect reports and help in their removal, for example by reproducing them or supplying more information. The importance of user involvement has been emphasized in the literature (e.g. Lerner and Tirole, 2002). While economic incentive theory would predict that most volunteers get involved as programmers – one of the most prestigious roles – Lakhani and von Hippel (2003) have also found a rich community of user-to-user assistance providing technical help and support to each other. Similarly, while work related to bug reports is often not very prestigious, there are various incentives for volunteers to perform such functions. For example, they can steer the project in a particular direction (through emphasis on certain feature requests) and can work with other developers to remove defects, thereby increasing the quality of the product. Such work is also a particularly good way to obtain first hands-on experience in a live project and to get involved in open source.

Villa (2003) confirms this in his observations about the GNOME 2.0 development and release cycle. He found that those users who are not able to code can get involved in quality assurance and thereby help create higher quality products. This clearly demonstrates that the introduction of bug tracking systems does not only allow an effective mechanism for tracking and prioritizing of defects but also promote user involvement by encouraging a new class of volunteers to join a project. This strengthens the *bazaar* surrounding a project and can lead to higher levels of quality and more successful projects (Senyard and Michlmayr, 2004).

6.2.3 Debian's Bug Tracking System

Villa (2003) observed that in order to be effective, the use of the bug tracking system must be tightly integrated with the development process. One project in which this is the case is Debian. The Debian Project is a large project with the aim of creating a complete operating system based on free software (Michlmayr and Hill, 2003; González-Barahona et al., 2004; Robles et al., 2005). Debian introduced a custom-built bug tracking system in 1994 and has since continuously refined the system, making it an increasingly integral part of Debian's development process. All communication between the users and the developers concerning defects and feature requests are carried out via the Debian bug

tracking system. The communication is based on e-mail, with every new input promptly archived in the bug tracking system and the information made available through the World Wide Web. Furthermore, there is a subscription feature through which interested parties can automatically receive all defect reports and communications concerning a specific software package.

The bug tracking system is well integrated with the development process. When a new defect is filed it is assigned a unique defect number and the report is automatically forwarded to the maintainer of the specific software package that is supposedly affected. The maintainer can manipulate the defect status in various ways, such as to make changes to the title or reassign the report to a different software package. If the defect requires a change in the software, the assigned defect number can be referenced in the changelog. Once the new software package becomes available, all defects referenced in the changelog will automatically be closed by the package archive system. In this case, the submitter of the defect report or feature request is automatically informed that the issue has been resolved. They will receive their original report together with the changelog describing how the issue has been addressed (Figure 6.1).

In order to manage the different defect reports effectively, a number of status variables is associated with each report. For example, a defect report can be marked as *pending* to signal that a solution is forthcoming. While there is no variable to indicate the urgency of a report, each defect report has a severity field which reflects the significance of this defect. The severity field is associated with one of six types (critical, grave, serious, important, normal and minor) or one special type known as *wishlist*. The last severity indicates that the report is, in fact, a feature request rather than an actual defect. This distinction becomes important in the following analysis.

```
Subject: Bug#308317: fixed in bins 1.1.28-1
[...]
We believe that the bug you reported is fixed in the
latest version of bins, which is due to be installed
in the Debian FTP archive:
[...]
Closes: 251306 308115 308317 313675 317731
Changes:
  bins (1.1.28-1) unstable; urgency=low
     * New upstream release.
       + Adds support for customized date strings via the
         dateString option. Closes: #308317.
```

Figure 6.1 Defect reports receive the changelog when a fixed software package becomes available.

6.3 DATA ANALYSIS

Since the introduction of Debian's bug tracking system in 1994 to the end of 2003, more than 220,000 defect reports and feature requests associated with more than 13,500 software packages were recorded in Debian's bug tracking system. This chapter focuses only on the subset of defects reported about Debian's base packages. The base packages in Debian form the core software that is part of every installation of the Debian system. Focusing on these packages eliminates the influence of two important variables: popularity and usage. Since all of these packages are installed on every Debian system, there is no difference in the popularity or usage of these software packages. This allows for a better comparison between the software packages considered in this study.

Since the base system changes over time as certain software packages get added and removed, we have limited the analysis to those packages that have formed an integral part of the base system throughout the whole data collection period. Seventy-seven software packages match this criterion. The time period consists of 30 months during which data was collected three times a day. The data collection started in April 2001, ended in October 2003, and comprises more than 7,000 defect reports. For some analyses, this period was divided into three periods of 10 months each.

For the analysis, various statistical methods are employed using the statistics package GNU R, version 1.8.0. For all tests $\alpha = 0.05$ is chosen as the level of significance (the probability of rejecting the statistical hypothesis tested when, in fact, that hypothesis is true). First, we investigate how variables connected to the defect reports change over time. We later analyze effective strategies that have been used for defect removal.

6.3.1 Changes of Variables Over Time

Bug tracking systems play an important function tracking user feedback. The quality of a piece of software can only be improved if defects are identified, reported and corrected. At the same time, free software developers are generally volunteers with a limited amount of time. Dealing with defect reports takes time and can be overwhelming if there are too many of them; in particular, problems may arise when most volunteers focus on prestigious and highly rewarding tasks, such as the creation of new functionality, and little emphasis is put on more mundane tasks such as the removal of defects and maintenance of existing features.

6.3.1.1 Total number of defects

The first question regarding the 30-month period of open (unresolved) reports in Debian's base system is whether the number of open defect reports has changed. Since the number of packages taken into account has stayed constant throughout

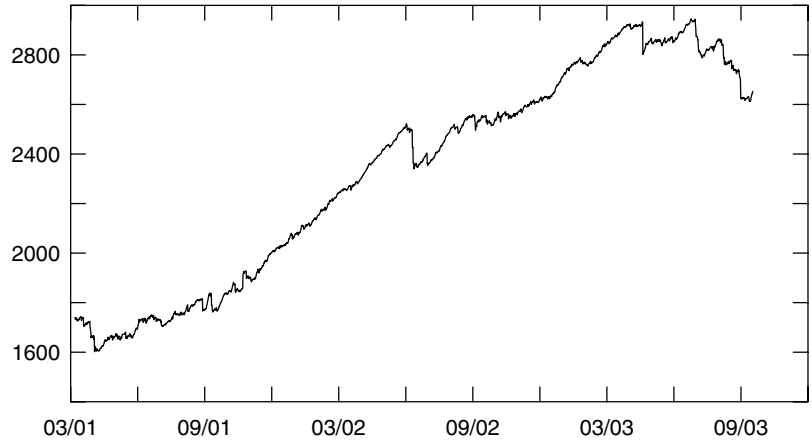

Figure 6.2 Unresolved defects in Debian's base system.

the whole period, the number of open defects should ideally roughly stay the same. However, as Figure 6.2 suggests and an analysis confirms, this is not the case. A paired, two-sided t-test between the number of unresolved defects in all packages together at the beginning and the end of the data collection period clearly shows that there is a significant difference ($t(76) = -3.24$; $p < 0.002$). As it turns out, the number of open defects has grown over the period which was examined. While there were a total of 1689 unresolved reports associated with Debian's base packages at the beginning of April 2001, there were 2615 in October 2003. The same observation can be made about the number of true defect reports ($t(76) = -2.99$; $p < 0.004$) as well as for feature requests ($t(76) = -2.54$; $p \approx 0.013$). It is important to emphasize that the number of software packages considered stayed constant throughout this study. Therefore, an increase of packages in Debian's base package cannot account for the increase of the total number of defects. Similarly, the number of developers working on these packages stayed roughly the same. There is little data on the usage of Debian and free software, but it is possible to study whether the number of defects that are reported is increasing over time.

6.3.1.2 Defect reports per week

One possible hypothesis is that this increase in the total number of defects is due to the success of Linux and open source – this popularity translates to an increased number of users who submit defect reports. A standard analysis of variance (ANOVA) shows that the year has an significant influence on the number of reports that have been submitted each week ($F(3) = 5.68$; $p < 0.001$). However, a closer examination shows that the number of defect reports per week does not increase. Rather, substantially more reports have been submitted per week in 2001 than in the other years in the observation period (Table 6.1).

Table 6.1 Average number of reports opened every week in each year of the observation period

Year	Average Defects per Week	σ
2000	54.42	21.94
2001	64.65	14.94
2002	53.75	12.36
2003	53.56	10.92

Excluding the year 2001 from the ANOVA reveals that the other years do not differ in a significant way ($F(2) = 0.037$; $p = 0.96$). There is currently no obvious explanation why 2001 differs from the other years in such a way. For example, no new version of Debian was released in 2001.

Since there is no obvious increase in the number of defects reported each week, the significant increase of the total number of open bug reports found in the previous section is likely related to the defect removal rate. Our hypothesis is that a certain percentage of defects do not get resolved and as more and more defects are reported (some of which do not get resolved), the number of open reports accumulate over time.

6.3.1.3 Removal rate

The increase of the total number of defect reports combined with a static defect submission rate in the last years suggests that the defect removal rate is decreasing. In order to investigate this hypothesis statistically, the 30-month investigation period was first arbitrarily divided into three periods. For each period, all defect reports that have been submitted in the particular period have been considered. Then, the status of each defect report exactly 10 months after it was submitted was observed. At this point, reports which have been dealt with would be recorded separately from those that still remain to be addressed. The proportion of resolved and open defect reports shows the defect removal rate for the particular period.

Unfortunately, the last period cannot be used for this analysis because it is impossible to forecast the defects in this period what their status will be 10 months after they were submitted. Due to the fact that performing the analysis on only two periods is suboptimal, we decided to go back and take 10 more months into account. Since we did not record any data from this period, a different method of data gathering was needed. Fortunately, the Debian bug tracking system stores its information in a way that allows one to reconstruct the history of a defect report at any point in time. We developed a tool for this reconstruction and verified it with the information we recorded in the 30 months. This tool delivered information for defects reported during the period from June 2000 to April 2001 and can be used in future studies.

Table 6.2 Defect removal rate by period

Period	Removal Rate as the Proportion Removed	σ
2000-06-01	0.76	0.22
2001-04-01	0.68	0.27
2002-02-01	0.61	0.31

Note: The table shows the proportion of defects that have been removed after 10 months.

An ANOVA revealed that the period in which a defect was submitted is a significant factor ($F(2) = 5.66$; $p = 0.004$). Of the defects that have been reported during the first period, 76% were resolved after 10 months. This figure decreases to 68% in the second period and to 61% in the third period (Table 6.2). This clearly shows that there is a decreasing trend in the defect removal rate. If this trend continues, an increasing number of defect reports will accumulate and a large number of defects will never be addressed.

6.3.1.4 Removal time by period

For those defects which were resolved, the time to remove the defect was recorded. An ANOVA shows that the period has a significant influence on the mean removal time ($F(2) = 25.87$; $p < 0.0001$). However, there is no clear trend of how this variable changes over time. While it took about 58 days ($\sigma = 75.63$) on average to resolve a defect reported in the first period, the value changes to 44 days ($\sigma = 70.47$) for the second period and then to 62 days ($\sigma = 83.67$) for the third period (Table 6.3). A preliminary analysis of defects from the fourth period shows that the average is likely close to 44 days. More research, possibly taking additional defect data into account, is needed to show which factors might have an influence on the removal time.

6.3.1.5 Removal time by software category

Finally, we tested whether the category of the software package has an influence on the defect removal time. Raymond argued that high quality in free software and open source projects is achieved through a bazaar of users and developers

Table 6.3 Mean days of the removal time

Period	Removal Time in Days	σ
2000-06-01	58.09	75.63
2001-04-01	43.96	70.47
2002-02-01	61.62	83.67

which forms around a project and contributes to its development (Raymond, 1999). Assuming this hypothesis is true, projects surrounded by a large bazaar should be of higher quality and be more feature-rich than projects which have attracted less users and developers. Some packages in Debian's base system are specific to the Debian distribution and will attract less developers according to this hypothesis than software which is also part of other software distributions. Furthermore, software developed by the GNU Project traditionally has a special status because of its maturity and popularity (Miller et al., 1990, 1995). We have therefore put all 77 software packages in Debian's base system into 3 categories: Debian, GNU or other. Then, we determined the removal time for each resolved defect (Table 6.4). Unlike in the last test, we did not split the whole period of 30 months into three but considered it as one. This explains why the values of the defect removal time presented here are generally higher than in the previous section and is the reason why the tests were not done as one multivariate analysis. An ANOVA performed on this data shows that the category has a major impact on the removal time ($F(2) = 7.58;\ p < 0.0001$). Defects in software from the GNU project are addressed much more quickly than other defects. While there is no significant difference between Debian and other packages ($t(4083) = -1.12; p = 0.26$), defects in Debian packages take almost 6% longer in average to be addressed than defects in other packages. This result supports Raymond's hypothesis that it is very beneficial to have a strong bazaar surrounding a piece of free software.

6.3.2 Impacts on Defect Removal

During the previous analyses we have clearly seen that the defect removal rate goes down and defect reports start to accumulate. This problem has to be addressed because otherwise important user feedback is not incorporated into the software and many opportunities of improving the software are lost. Important defects might also not be resolved in a reasonable time.

6.3.2.1 Bursts of activity

In the following, we are going to look at effective measures for defect removal. An observation of the growth rate of unresolved defect reports shows that there

Table 6.4 Mean removal time by category

Category	Removal Time in Days	σ
Debian	105.60	167.24
GNU	81.95	144.89
Other	99.88	163.96

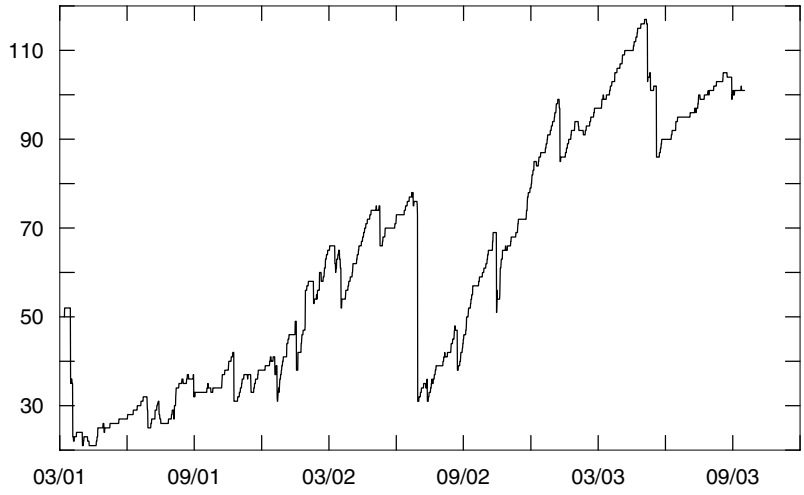

Figure 6.3 Slow increases and steep decreases of defect reports.

is usually a gradual increase. On the other hand, decreases are often much steeper (Figure 6.3). This suggests that defects accumulate over time and are removed in bursts of activity. This would be in line with the volunteer nature of free software projects – a developer may only occasionally find time (e.g., on weekends) to address defect reports and feature requests. There are also some fixed costs associated with each new software release, such as testing and distribution of the software. In order to test the hypothesis that defect reports accumulate slowly and are removed in bursts, we looked at the changes of the number of defect reports per package from one day to the next. There are 2.5 times more increases than decreases and the difference between the upwards and downwards changes in the number of unresolved defect reports is significant ($t(2604.7) = -13.48$; $p < 0.0001$). On average, if the number of reports grows, there is an increase of about 1.19 reports per day. On the other hand, if the number of defect reports goes down, 2.48 defects are removed on average.

As a next step, we analyzed major changes in the number of open defects within a package from one day to the next. For each software package, we checked if there was a difference of more than 5 defect reports. This minimum number ensures that only bursts of activity are taken into account. If this was the case, we additionally looked whether the change was more than 10% of the total number or more than 10 defects. An increase or decrease of more than 10% (assuming 10% involve more than 5 defects) or 10 defects from one day to another is quite a significant burst. In total, we identified 168 downwards and 10 upwards changes. Through an analysis of the changelogs of these packages

Table 6.5 Reasons for major downwards changes

Reason	%
Maintainer modifications	42.26
New version	27.98
New maintainer	14.29
Unknown	15.48

and the Debian bug tracking system, we identified the reasons for these major changes (Table 6.5):

- 42% of the downwards changes were caused by modifications made by the maintainer of the software package.
- 28% are due to a new version of the software released by the upstream author.[1]
- 14% of changes can be associated with a new Debian maintainer taking over or helping the existing maintainer.
- No specific reason could be found for 15%.

Major upwards changes are generally caused by sloppy mistakes that have a severe impact on the usability of the software, such as a typo in the installation scripts or a file conflict between different software packages. There was only one case where a major change was due to the release of a new version made by the upstream author.

6.3.2.2 Upload frequency

The previous analysis shows the importance of modification and new versions both by the Debian maintainer and by the upstream author of the software. Observing open source release styles, Raymond coined the "release early, release often" paradigm (Raymond, 1999). He suggested that frequent releases are vital in attracting new volunteers for a project and forming a bazaar. In order to test whether more feedback is given on active projects, we first gathered the mean time between new uploads for each software package. Then, we checked whether the upload frequency has an impact on the number of defects which are filed. This turned out to be the case ($F(1) = 5.34$; $p \approx 0.024$). While on average 159.1 defect reports ($\sigma = 177.19$) have been submitted on packages which have an average upload frequency of 30 or less days, only 68.58 defects or feature requests ($\sigma = 114.49$) were reported for packages uploaded less frequently. It

[1] In Debian, the maintainer and the developer of a piece of software are generally different people. Debian's goal is not to write software, but to take software from other sources (known as 'upstream') and to integrate them into one system which works together very well. Hence, changes to the software can either be made by the original developer of the software or by the Debian maintainer of this specific software package.

is very difficult to interpret this finding, however, since the connection between upload frequency and number of defect reports can have different causes. While it might be the case that increased feedback is given on more active projects, the finding might just as well indicate that these packages are uploaded more frequently because they have so many defects. More research is needed to answer these issues of causality conclusively.

However, there is one variable connected to upload frequency which can be interpreted without any doubt. An ANOVA revealed that the upload frequency has a significant influence on the removal time ($F(1) = 85.16$; $p < 0.0001$). Defects in packages which have an upload frequency of 30 days or less are removed on average after 80.12 days. On the other hand, packages which are uploaded less frequently show a removal time of 118.2 days. This clearly shows that certain advantages are associated with frequent releases.

6.3.2.3 Team maintainers

Another factor that can be correlated with the removal time is whether the software package is maintained by one person or by a team ($F(1) = 11.41$; $p \approx 0.007$). Traditionally, software packages in Debian were maintained by individuals. However, due to the volunteer nature of Debian, it is often impossible to ensure timely fixes if only one person is responsible for a package (Michlmayr and Hill, 2003; Michlmayr, 2004). Some maintainers have therefore started to jointly maintain a package as a team. The present analysis clearly shows that defects in packages maintained by teams are resolved more quickly. However, out of 77 software packages in Debian's base, only 13 (less than 17%) are currently maintained by a team. Therefore, the present findings should be revisited when more software packages in Debian are being maintained by teams.

6.4 DISCUSSION

An analysis of over 7,000 defect reports associated with 77 core packages in Debian's base system reveals that there is a clear increase in the number of unresolved reports. During the 30-month investigation period, the number increased from 1689 in April 2001 to 2615 in October 2003. This figure is based on all open bug reports in Debian's base, counting both defect reports and feature requests. One possible hypothesis would be that this increase is connected to the growing success of Linux and free software. As more people start to use free software, Raymond's "many eyes" principles come into play: more defects are found and reported and an increasing number of features are requested. However, this hypothesis does not appear to be true. The data shows that there is no increase in the number of reports submitted each week. While more reports have been filed in 2001 for unknown reasons, there is a constant level of about

53 reports per week on Debian's base in 2000, 2002 and 2003. One reason for this constant level might be that users perceive that their reports are not being addressed properly, hence they lose motivation to provide feedback.

While the defect submission rate stays constant, the defect removal rate has decreased. In three periods of 10 months each, the defect removal rate went from 76 to 68%, and from there to 61%. Ironically, one reason for this reduction might be the success of Debian. The project grew from about 3,900 software packages at the end of 2000 to over 13,500 packages in November 2003. A possible hypothesis is that more effort is spent on maintaining these additional packages than on keeping the core packages defect-free. Regardless of the cause of the falling defect removal rate, this reduction in the defect removal rate will have serious effects in the long term if effective countermeasures cannot be found. An increasing number of reports will not be resolved and defect reports will start to accumulate. This will make it much more difficult to determine which defects are relevant. More time needs to be spent on prioritizing the reports. Moreover, as defects become older, reproducing them becomes increasingly more complex because the software continuously changes. Finally, users will perceive that their feedback does not have any impact and will stop providing valuable input. This minimizes the benefits the project can draw from peer review and user involvement, which is so characteristic for free and open source software projects (Raymond, 1999).

An effective strategy to deal with this problem has to be implemented immediately. During the GNOME 2.0 development and release cycle, organized quality assurance efforts were introduced, and defect reports were prioritized (Villa, 2003). A similar organized effort is required by Debian. While Debian has an existing quality assurance team, there is currently little focus on defect reports. It would be very beneficial to have an organized group that helps to reproduce and prioritize defect reports, along with providing possible solutions. Villa (2003) has observed that introducing organized quality assurance attracts a new type of volunteer who previously did not have the chance to participate in free software projects. People who cannot program can help in the Quality Assurance (QA) efforts by reproducing and prioritizing defects. This strengthens the bazaar surrounding a project and at the same time reduces the amount of time the maintainers and upstream authors of the software have to spend on this activity. Finally, in his observations about GNOME quality assurance process, Villa makes a very controversial suggestion and recommends that old defect reports should simply be discarded. He argues that they have a very slim chance of ever getting resolved but take away valuable time from volunteers. He suggests that a new report will be submitted if the defect has truly not been resolved and is of any significance. This procedure is currently not followed by the Debian Project. Villa's suggestion is highly controversial and it is not clear which strategy leads to higher quality software in the long run.

While a QA effort will be able to improve the situation dramatically, the present findings also offer some strategies for the maintainers of software packages. An analysis has revealed that 42% of defects are addressed by maintainer modifications, 28% by a new version of the software provided by the upstream authors and 14% by changing or assisting the maintainer. Additionally, packages with a high upload frequency have been shown to have a shorter defect removal time and possibly receive more feedback from its users.

There are many important lessons to be learned from the present findings. First, frequent releases have certain advantages. They reduce the time a user has to wait for a defect to be resolved and they possibly promote feedback that can be used to increase the quality of the software. Second, it is very important to work together with the upstream author of the software. Debian does not write most of the software it distributes, but obtains it from other sources and integrates it into a system that works together very well. The maintainers of the Debian packages interact with the upstream author of the software closely and forward defect reports and feature requests. It is suggested that upstream authors subscribe to Debian's defect reports about their software so they can rapidly respond to defects and feature requests. The upstream authors benefit from the testing the users perform and from the feedback users give. Third, it is recommended to work on a software package jointly with other maintainers. Given the volunteer nature of free software participants, nobody can ensure that they will have enough time to respond to an issue quickly. Spreading the load across several maintainers leads to more reliability and to a shorter defect removal time. At the same time, working in a team increases the communication and coordination requirements and therefore effective means of splitting the work have to be found (Michlmayr and Hill, 2003; Michlmayr, 2004).

6.5 CONCLUSIONS

Many large, and some small, free software and open source projects use bug tracking systems to capture defect reports and feature requests. In this chapter, a detailed analysis has been performed on data which has accumulated in a free software project over 2.5 years. This chapter comprises several statistical analyses of over 7,000 defect reports submitted on 77 core packages in the Debian system. The findings are of substantial value since they clearly show changes over time and allow the development of effective strategies for defect removal.

The number of defects that still need to be resolved has increased over a period of just under three years. While the defect submission rate stayed constant, the defect removal rate is continuously decreasing. Effective strategies such as an organized quality assurance effort have to be implemented. Furthermore,

three important lessons concerning maintainers of software packages are learned from the study. First, frequent releases lead to shorter defect removal times and possibly to more user feedback. Second, a close interaction with the upstream authors of free software is beneficial, and upstream authors gain from wide testing and more user feedback. Third, working in teams increases the reliability of volunteer maintainers and leads to shorter defect removal times.

Finally, this study shows that many important insights can be gained through the analysis of defect data that has been recorded over time, possibly for years. The tool that was written for this study can be used to investigate roughly 300,000 defect reports stored in Debian's bug tracking system. It remains to be seen whether a similar tool can be created for other popular bug tracking systems used in free software projects, such as Bugzilla. Such a tool would allow the investigation of further projects as well as a cross analysis of the bug-fixing behaviour of different projects. It can be concluded that the availability of high quality data is of substantial value to software engineering and open source researchers and practitioners, and should be used for further studies.

APPENDIX: AVAILABILITY OF DATA AND TOOLS

The complete raw data of Debian's bug tracking system is available publically via the `rsync` protocol from bugs-mirror.debian.org (see http://www.debian.org/Bugs/Access). The following three modules are available:

bts-spool-db: for active bugs. Active bugs are bug reports which are either open (unresolved) or have recently been closed without yet being archived. At the time of writing, this module had about 3.5 GB of data on roughly 60,000 bug reports.

bts-spool-archive: for archived bugs. Resolved bug reports are archived after approximately 30 days. There are about 13 GB of raw data on 235,000 bug reports in this module.

bts-spool-index: for bug index files. This module comprises approximately 30 MB.

The tool developed as part of this study that allows the reconstruction of the status of bug reports on any given date is available on request from the corresponding author.

In addition to the data available from Debian, there is a project called "FLOSSmole" which aims to provide a platform for collaborative collection and analysis of data on free software and open source projects (Howison et al., 2005).

REFERENCES

Capiluppi, A., Lago, P. and Morisio, M., 2003. Evidences in the evolution of OS projects through changelog analyses. In: *Proceedings of the 3rd Workshop on Open Source Software Engineering*, International Conference on Software Engineering (ICSE 2003), Portland, USA, 19–24.

Doolan, E. P., 1992. Experience with Fagan's inspection method. *Software – Practice and Experience*, 22(2), 173–182.

Fagan, M. E., 1976. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3), 182–211.

German, D. and Mockus, A., 2003. Automating the Measurement of Open source projects. In: *Proceedings of the 3rd Workshop on Open Source Software Engineering*, International Conference on Software Engineering (ICSE 2003), Portland, USA, 63–67.

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

Howison, J., Conklin, M. and Crowston, K., 2005. OSSmole: A collaborative repository for FLOSS research data and analyses. In: Scotto, M. and Succi, G. (eds), *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, 54–60.

Koch, S. and Schneider, G., 2002. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal*, 12(1), 27–42.

Lakhani, K. R. and von Hippel, E., 2003. How open source software works: "Free" user-to-user assistance. *Research Policy*, 32(7), 923–943.

Lerner, J. and Tirole, J., 2002. Some simple economics of open source. *Journal of Industrial Economics*, 50(2), 197–234.

Michlmayr, M., 2004. Managing volunteer activity in free software projects. In: *Proceedings of the 2004 USENIX Annual Technical Conference*, FREENIX Track, Boston, USA, 93–102.

Michlmayr, M., 2005. Software process maturity and the success of free software projects. In: Zielinski, K. and Szmuc, T. (eds), *Software Engineering: Evolution and Emerging Technologies*. IOS Press, Amsterdam, 3–14.

Michlmayr, M. and Hill, B. M., 2003. Quality and the reliance on individuals in free software projects. In: *Proceedings of the 3rd Workshop on Open Source Software Engineering*, International Conference on Software Engineering (ICSE 2003), Portland, USA, 105–109.

Miller, B. P., Fredriksen, L. and So, B., 1990. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12), 32–44.

Miller, B. P., Koski, D., Lee, C. P., Maganty, V., Murthy, R., Natarajan, A. and Steidl, J., 1995. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services. *Technical Report*, Computer Sciences Department, University of Wisconsin.

Raymond, E. S., 1999. *The Cathedral and the Bazaar*. O'Reilly & Associates, Sebastopol, USA.

Robles, G., Gonzalez-Barahona, J. M. and Michlmayr, M., 2005. Evolution of volunteer participation in libre software projects: Evidence from Debian. In: Scotto, M. and Succi, G. (eds), *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, 100–107.

Senyard, A. and Michlmayr, M., 2004. How to have a successful free software project. In: *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, Busan, South Korea, 84–91.

Stamelos, I., Angelis, L., Oikonomou, A. and Bleris, G. L., 2002. Code quality analysis in open-source software development. *Information Systems Journal*, 12(1), 43–60.

Vesperman, J., 2003. *Essential CVS*. O'Reilly & Associates, Sebastopol, USA.

Villa, L., 2003. Large free software projects and Bugzilla. In: *Proceedings of the Linux Symposium*, Ottawa, Canada, 471–480.

Wheeler, D. A., 2001. More than a gigabuck: Estimating GNU/Linux's size. http://www.dwheeler.com/sloc.

Zhao, L. and Elbaum, S., 2003. Quality assurance under the Open Source development model. *The Journal of Systems and Software*, 66(1), 65–75.