USENIX Association

Proceedings of the FREENIX Track: 2004 USENIX Annual Technical Conference

Boston, MA, USA
June 27–July 2, 2004

Managing Volunteer Activity in Free Software Projects

Martin Michlmayr
Department of Computer Science and Software Engineering
University of Melbourne
Victoria, 3010, Australia

Centre for Technology Management
University of Cambridge
Mill Lane
Cambridge, CB2 1RX, UK

martin@michlmayr.org

Abstract

During the last few years, thousands of volunteers have created a large body of free software. Even though this accomplishment shows that the free software development model works, there are some drawbacks associated with this model. Due to the volunteer nature of most free software projects, it is impossible to fully rely on participants. Volunteers may become busy and neglect their duties. This may lead to a steady decrease of quality as work is not being carried out. The problem of inactive volunteers is intensified by the fact that most free software projects are distributed, which makes it hard to quickly identify volunteers who neglect their duties. This paper shows Debian's approach to inactive volunteers. Insights presented here can be applied to other free software projects in order to implement effective quality assurance strategies.

1 Introduction

The success of free software in the last few years, including projects such as Linux and Apache, has clearly demonstrated that the free software development model is a viable alternative to proprietary software development. Free software is a matter of liberty, rather than price, since it offers the user more freedom than proprietary software, such as the freedom to run, copy, distribute, study, change and improve the software. One reason for the success of the free software development model, which is tightly connected to the Unix philosophy [19], is related to the open nature of free software: a large number of developers inspect the code and get involved in a project. Eric S. Raymond studied the free software development model in detail and published his observations in a widely acclaimed paper [18]. During his case study, Raymond found that there is a high amount of parallelization in the debugging process. Due to the open nature of the source code in free software projects, anyone can review the code, find defects and contribute bug fixes. Raymond suggested that this 'bazaar' model, in which a large number of volunteers review the code and contribute feedback and patches, is the reason for the success and high quality of many free software projects. This suggestion meshes well with findings in software engineering which show that peer review significantly contributes to quality [9].

Free software development is typically characterized by two factors. First, free software projects are carried out by volunteers. Second, the volunteers of a project are distributed. However, not all projects exhibit these two characteristics. Due to the success of free software, an increasing number of companies get involved in free software projects. Some companies allow their paid developers to get involved in distributed free software projects. Also, some software is being developed in a traditional fashion by paid programmers working in the same building and is then released as free software. This example shows that it is important to make a clear distinction between the way software is developed and the license under which it is distributed. A piece of software can be developed in a traditional way and then be licensed as free software. However, this paper covers a certain aspect of software developed according to the free software development model, that is, projects which are distributed and largely carried out by volunteers.

This free software development model has generated a large amount of very popular and successful software in the last few years. Due to the recent popularity of free software, there is an increasing number of academic studies and papers which concentrate on successful projects, such as Apache [16], GNOME [14] and Linux [20]. However, there are also a large number

of unsuccessful free software projects which have not yet attracted the attention of researchers. SourceForge hosts over 70,000 projects, but a large number of them are not actively developed. Since free software projects do not have budgets, there is usually not a specific point when unsuccessful projects get wrapped up. It is therefore hard to identify projects which have been failures. Nevertheless, it is clear that some free software projects fail or have severe quality problems.

The distributed and volunteer nature of free software projects make them unique in software development, and is related to certain advantages, such as the high amount of peer review mentioned previously. However, there are several drawbacks and challenges associated with the free software development model as well. Due to the volunteer nature of free software projects it is impossible to fully rely on participants [15]. Also, coordination and management can be very complex in distributed projects with volunteer participants. For example, it is impossible to put all participants of a distributed free software project in a room, give out tasks, and only leave the room when everyone agrees to perform the tasks they have been given.

Free software projects have to deal with these challenges and find solid solutions. The success of free software shows that many challenges have been overcome, and the development model is continuously refined as more experience is gained. One major challenge, which is becoming an increasing problem, is volunteers who suddenly stop carrying out their duties without informing others. In many free software projects, volunteers have specific responsibilities. While a large number of participants make infrequent contributions, there are some volunteers who play crucial roles in a project, and who therefore have to be constantly involved. For example, the Linux kernel has a number of trusted lieutenants through whom code submissions of specific parts of the kernel are carried out [17]. They are central to the development, since other volunteers cannot contribute if the trusted lieutenants do not carry out their work. Similarly, if the main developer of a very small project becomes inactive, the whole project may come to a halt. If such volunteers become inactive, especially without informing others so a backup can be arranged, projects face important problems. There are therefore two problems: a project can stop completely if a core developer becomes inactive, or the quality of a project may suffer.

It is therefore very important to observe and investigate the problem of inactive volunteers from a quality assurance perspective and to discuss possible solutions. In the following, I will approach this problem from the perspective of the Debian Project, and discuss mechanisms Debian has implemented to deal with this problem. While some solutions described in this paper are specific to Debian, many lessons can be learned from Debian's experience of dealing with inactive volunteers. It is my hope that members of other projects will gain a better understanding of the problem through this paper and map Debian's strategies to their respective projects.

2 Background

As discussed in the previous section, volunteers who are not performing their duties can have a significant impact on the quality of a project. Therefore, it is important to take this problem into consideration in a project's quality assurance effort. In this section, I will discuss and summarize the problem, and describe why it is such a big problem especially for Debian.

2.1 Inactive Volunteers

The motivation of volunteers in a free software project is different than that of paid developers involved in the development of a commercial application. Raymond has observed that developers get involved in free software projects to "scratch an itch" [18]. This explains why certain tasks are not carried out in free software projects. For example, most free software projects do not have a written specification, simply because writing one is a tedious job most volunteers are not interested in. Similarly, users of free software are not in a position to demand new features from the developers of a project. If they require functionality nobody else is interested in developing, they can get involved themselves and contribute to the project, or pay someone to implement the missing features.

There are two schools of thought about the responsibilities of a volunteer. One school maintains that a volunteer is free to do whatever they wish at any time and does not have any obligation at all. The other school of thought claims that once a volunteer has agreed to perform a specific function they have a responsibility to fulfill it and to tell others when they cannot perform their duties anymore. From a quality assurance perspective, it does not really matter what a particular volunteer thinks – certain measures have to be taken in *any* case. However, it is clear that it is easier to work with volunteers who have a certain diligence in their responsibilities. Ideally, a volunteer will realize when they can no longer perform their function, for example because they are too busy or have lost interest, and will arrange for a replacement or backup. Unfortunately, experience from Debian and other projects shows that this is often not the case.

The problem of inactive volunteers is tightly connected to the nature of the free software development model. The fact that most free software projects are performed by volunteers makes solutions fairly complex. Due to the volunteer nature of free software projects, it

is inevitable that participants become busy from time to time. Student volunteers have exams and participants who earn their living in companies might have less time sporadically when a project is due. For all participants it is true that personal circumstances may change and leave less time to contribute to free software projects. Furthermore, participants in free software projects can sometimes experience 'burnout' [12]. There are many variables that may change and affect the participation in a free software project. However, it is clear that earning a living and 'real life' take precedence when time is short. Hence, it is important that a free software project does not rely on a specific volunteer for a crucial task but instead implements redundant structures [15].

The problem is aggravated by the second characteristic of free software projects: its participants are distributed. This fact makes it very hard to identify the personal circumstances of a developer. In a traditional, commercial software project, people would notice very quickly if something happened to one of their developers. Their colleagues would wonder why a developer is not coming in to work. They would easily *see* that something is wrong because the developer is not present. Unfortunately, it is not as easy in free software projects where coordination and communication mainly relies on e-mail – there is not much one can do when a volunteer does not respond to inquiries by e-mail.

The combination of these two factors makes it very hard to identify inactive volunteers in free software projects. Moreover, the question whether a participant is active is of a gradual rather than a binary nature. If a volunteer has not performed their duties for a month, it is possible that they are just temporarily busy and will come back later. This makes it very hard to draw the line and make a decision when to take action. A volunteer might just be on holidays for three weeks, but one does not know that because it was not explicitly announced.

Raymond covered the problem of free software maintainers who do not perform their duties anymore briefly in his paper "Homesteading the Noosphere" [18]. It is a well-known problem and some guidelines have been discussed, such as when to fork or take over an abandoned project. There are also two terms which have been associated with inactive volunteers. *AWOL*, which stands for "Absent Without Leave", and *MIA*, short for "Missing In Action", are both used to refer to volunteers who become inactive without informing others.

While the problem is known, no solutions have been developed which are generally applicable. It is also a very delicate problem because one has to maintain the balance between respecting a volunteer for what they have done while at the same time pointing out that they are not performing their duties anymore and suggesting that it is time to move on. Before describing Debian's approach to the problem, an explanation of why inactive volunteers are a severe problem specifically for Debian is in order.

2.2 Debian

Debian is one of the most popular and certainly the largest Linux distribution available to date [11]. The mission of Debian is to provide a complete operating system comprising free software. While Debian develops some software itself, such as Debian's package manager `dpkg`, the main task of Debian is not to develop software. Instead, Debian obtains software from other sources (the so-called "upstream" developers) and integrates the different pieces of software into one system. The integration is done through the creation of a Debian package which complies to certain guidelines which are outlined in a document known as the Debian Policy manual [5]. Every package is under the control of one or more maintainers, and most volunteers in Debian maintain one or more packages. While there are some volunteers who help out with other tasks, such as maintaining the web site or the software archive, publishing security updates, or performing quality assurance functions, the majority of volunteers in Debian are package maintainers.

Although there are some packages which are being maintained as a team, the majority of packages are currently maintained by one person. As the sole responsibility for keeping a package up-to-date and free of defects rests on that package's maintainer, there exists a heavy reliance on those volunteers. Achieving more redundancy by having backup maintainers and maintainer teams has been suggested [15]. While there is a slow trend away from having a single maintainer for a package, the majority of packages are still maintained by one person. Therefore, there is a high liability to packages becoming unmaintained as volunteers become busy or entirely inactive. This mapping of one package to one single maintainer is the reason why inactive maintainers are such a huge problem for Debian. In some respects, Debian is like a bazaar of cathedrals: a project comprising a large number of individual projects of varying size, each with its own owner.

There is one additional factor which makes the situation very complex: Debian's growth. Debian has grown in size over the years, and especially in recent years, quite drastically (see figure 1). The enormous size of Debian makes it very hard to keep track of every maintainer. With over 800 maintainers, it is impossible to know each volunteer in Debian, and it is very hard to spot if one out of over 8,000 packages in the archive is not maintained properly. Also, while Debian as a whole has grown in size, the quality assurance team has remained about the same size.

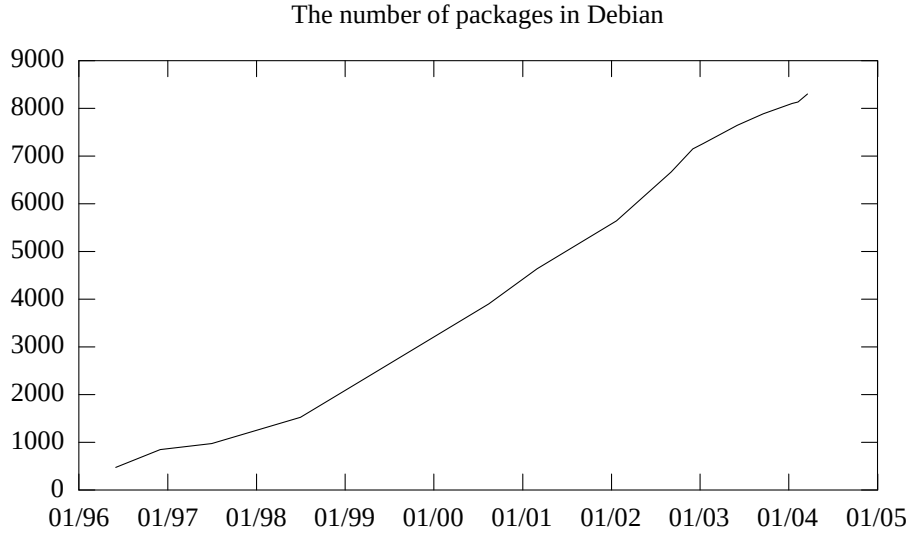

Figure 1: The size of Debian has steadily increased over the years. At the beginning of 2004, the project featured over 8,000 packages.

Taken together, these factors constitute a large quality assurance problem which has to be solved. In the following sections, I will describe ways to locate and track inactive maintainers, and show tools which have been developed to assist in these tasks. Before covering inactive maintainers, it is important to give some attention to the full life cycle of a maintainer and to discuss how one actually becomes a package maintainer in Debian.

3 Debian's New Maintainer Process

Over the ten years of Debian's existence, many processes have been adapted and refined in adherence to the ways in which Debian has changed. Debian's admission process was very informal in the early days. Everyone knew each other at that time. Prospective members could simply send an e-mail stating their interest and they would be added to the project. It was also very common to help out with the packages of another maintainer when they were busy. As Debian mushroomed, the processes slowly became more formal in order to deal with the large number of developers and the size of the Debian system.

As Debian reached a size where it was impossible to know every other volunteer, a new admission process was needed. The New Maintainer process [4] was introduced to handle new volunteers. Debian's New Maintainer process is much more elaborate than the admission processes of most free software projects. One reason for this is that every Debian developer has to be fully trusted. Since Debian packages are installed as root on a user's machine (that is, with full permissions), it is necessary to strictly control who can upload packages to the Debian archive. Each software package uploaded to the archive must be digitally signed with the GPG (GNU Privacy Guard) or PGP (Pretty Good Privacy) key of a Debian developer. When a volunteer fulfils all steps in the New Maintainer process and gets accepted as a Debian developer, their GPG key is added to the Debian keyring against which signatures of every package upload to the archive is verified.

The New Maintainer process is composed of three steps. First, every volunteer needs a GPG key which is digitally signed by an existing Debian developer. This involves a face-to-face meeting which allows performs a social function and intends to get new members better integrated. Second, the volunteer's philosophy with regards to free software and understanding of Debian's Social Contract [7] is checked. Third, the volunteer's technical skills are tested in various ways. Once an applicant has passed all three steps, a thorough report is sent to the Debian Account Manager (DAM) who decides whether an account is created. Debian's constitution grants the Debian Account Manager the authority to admit new people to the project or to remove existing developers [2]. Traditionally, the DAM has exercised the former right, but as inactive maintainers get identified it also becomes important to remove volunteers from the project.

The New Maintainer process plays an important role in the problem of inactive maintainers because it selects which volunteers are admitted to the project. In many large free software projects, it seems fairly typical to make a one-time contribution of source code. The project benefits from this contribution as the code

gets integrated in the project and is then maintained by others. The original contributor does not have to show a high level of commitment, as the code has been integrated and the contribution is therefor useful. However, in Debian few people are interested in contributing patches for old bugs or assisting other maintainers. Most volunteers are interested in maintaining packages on their own. For that, however, you need a constant level of contribution. Otherwise, the package is not maintained well.

The New Maintainer process is vital in ensuring that only volunteers who show a high level of commitment get admitted to the project. If large numbers of new volunteers who do not understand the problem they cause by neglecting their commitments are admitted, the problem is intensified. It is therefore important to tackle the problem at its root and educate new maintainers. It is important to take into account that the problem is largely a social one. In many cases, volunteers realize that they are busy, but they do not admit to themselves that they are actually *too* busy, and are harming the project. They often think they will find the time "tomorrow", but in reality it never happens. Hence, it is important to discuss this problem from the beginning, and the New Maintainer process is a good venue for this.

Over the last years, the New Maintainer process has become more elaborate and longer than in the past. While this change was not a conscious decision, I think this is partly because of inactive maintainers. If the admission process takes a long time and is quite complex, only those volunteers who are really interested will go through the process. Once they have cleared this hurdle, it is likely that they will stick around for a long time. The New Maintainer process therefore acts as a screening process not only to select volunteers with good technical skills, but also those who are likely to show a high level of commitment to the project in the future. However, this is only speculation on my part, and it is too early to tell whether this strategy really works out. It will be interesting to conduct a study in a few years to see whether the duration of the admission process is linked to the likelihood of a volunteer neglecting their duties. For this study, you would first investigate how long or complex the joining process was for someone, and then check whether they are still active a certain number of years after joining. Even without an elaborate study, it is clear, though, that the admission process has to take the problem of inactive maintainers into account.

4 Locating Inactive Maintainers

As mentioned before, finding volunteers who are neglecting their duties is much harder in a distributed volunteer project than in a company where everyone comes to work every day. In this section, I will describe the sources of information Debian takes into account when searching for inactive maintainers.

Due to the volunteer nature of free software projects, it is quite likely that some volunteers will neglect their commitments, especially in large projects. It is therefore important to take this possibility into account and to mention it in the developer's documentation. For example, the Debian Developer's Reference explicitly asks volunteers to inform other members of the project when they go on holiday and provides a section describing how to retire from the project properly [3]. This way, arrangements can be made so other volunteers take over their tasks, either temporarily while they are on holiday or permanently when a volunteer retires. Most communication in free software projects is based on e-mail or IRC (Internet Relay Chat, a form of real-time communication) and many participants have never met in person. While e-mail is usually a very effective means of communication, it completely fails if someone does not answer their e-mail – unfortunately, this is likely the case if someone is busy or something serious has happened. Contacting the person by phone is often a more effective alternative in this case. Hence, having the phone number of each volunteer and additionally also an emergency contact, either a relative or a friend, are very helpful in finding out what happened to a volunteer if they do not respond to e-mail. While Debian currently does not require either a phone number or an emergency contact in its developer database due to privacy concerns, participants will be able to enter this information on a voluntary basis in the future.

4.1 Sources of Information

In most free software projects, many sources of information are available which can be used in order to form a judgement whether a volunteer is inactive. In the following, I will describe sources typically used to locate inactive maintainers in Debian.

4.1.1 Echelon

Like most free software projects, Debian relies on electric means of communication. While IRC is used by a large number of volunteers in Debian, the primary infrastructure is based on e-mail. Debian has an automatic system which monitors every mailing list and looks for postings sent by a Debian developer. The system, known as Echelon, uses various means to determine whether a mail is sent by a Debian developer, such as recognition of the e-mail address or verification of the signature of a GPG or PGP signed message. When a message has been recognized as one posted by a Debian developer, an activity value is updated in Debian's internal LDAP database. This database is used to store various bits of information about every official Debian developer.

In addition to fields for name, login and other personal information, there is a field which indicates the time a message has been last posted on the Debian lists by a developer. This field also lists the Message-ID of the e-mail so it can be found easily. In addition to the activity value, there is a field which indicates when the last digitally signed message has been posted. Since all Debian uploads have to be signed, this activity field gives an indication when the maintainer has last uploaded one of their packages.

Debian's Echelon system is a tremendous resource during the search for inactive maintainers. While it is not 100% reliable, since it sometimes does not recognize a message to be from a Debian developer, it gives a very good first indication of whether a participant has been around recently or not. This system is very useful to see whether it is worth looking at other sources of information.

4.1.2 Package Information

Many sources of information are directly related to the job of a Debian maintainer – maintaining packages. If the packages of a specific maintainer appear unmaintained, this is a clear indication from a quality assurance point of view that certain measures have to be taken. While unmaintained packages do not necessarily imply that a maintainer is inactive, the situation has to be dealt with. It is sometimes the case that some packages are not maintained well because a maintainer is overwhelmed by the amount of work they have. In that case, it may be that they maintain their important packages well but neglect others which they deem to be less important. However, in some cases, unmaintained package are due to a maintainer simply neglecting all their duties.

There is a wide range of information which can be used to judge how well a package in the Debian archive is being maintained.

Release Critical Bugs: Debian has a Bug Tracking System through which bugs can be reported [1]. Each bug has a severity, ranging from minor to critical. Debian defines bugs of the severities serious, grave and critical to be "release critical", meaning that a package with bugs of that class is not considered to be ready for release. The Bug Tracking System provides an easy overview of all bug reports of a specific maintainer. This information can be used to see how well their packages are being maintained. If a maintainer has release critical bugs which have been outstanding for a while, this is a good indication that they are not performing their duties. This is especially the case if no activity can be found in the bug history, which is the case, for example, if the maintainer has never responded to the bug submitter asking for more information or clarifying the bug.

FTBFS Bugs: Debian supports a large number of architectures, and each package which is uploaded to the Debian archive is built automatically on all architectures. If a package has previously built on an architecture but no longer builds, a release critical bug is filed saying that the package "fails to build from source" (FTBFS). As with other release critical bugs, these give a good indication of the activity of a Debian maintainer.

In Debian's Bug Tracking system, bugs can be marked as closed in two ways. If the maintainer of a Debian package closes the bug, it is marked as done and the bug submitter is informed. On the other hand, if someone else closes a bug, it is merely marked as fixed and the bug submitter is not notified. It is the responsibility of the maintainer to confirm that the bug has been dealt with and to close the bug for good. This distinction is very helpful in the search for inactive maintainers: a large number of bugs marked as fixed might indicate that someone else is performing the maintainer's duties, maybe because they are not doing it themselves. Fixed bugs are especially interesting in the case of release critical and FTBFS bugs.

Old Standards-Version: The Debian Policy manual describes what Debian compliant packages have to look like, and this document is continuously updated to reflect new procedures. Each Debian package has a field which indicates which version of the Policy it complies with. This field, known as the Standards-Version, is very helpful to find packages which are out of date with regards to current Debian procedures. Similarly, if a package has not been updated a year after a stable release of Debian has been made, this is a good sign that a package is not maintained well. Even if a piece of software is not developed anymore, the Debian package has to be updated regularly as Debian procedures change. Hence, packages have to be updated regularly regardless of how fast the upstream software is developed (or whether it is still in development at all).

New upstream version: It is a good sign of an inactive maintainer if a new upstream version of the software has been available for a while which has not yet been packaged for Debian. While there are sometimes good reasons not to package the new version immediately (such as when a release is not deemed to be stable or of release quality yet), in most cases it hints at an inactive or busy maintainer.

4.1.3 Hints From Developers

With the current size of Debian, it is impossible for the fairly small quality assurance team to monitor each package to find inactive maintainers. It is therefore very beneficial to have a well-documented contact address where other developers and users can report maintainers who they think are not active anymore. The quality assurance team which has experience in tracking down

inactive maintainers can then use this pointer to investigate further whether a maintainer is really neglecting their duties.

5 Contacting Maintainers

Once a maintainer has been identified who is believed to be neglecting their duties, they have to be contacted before any measure is taken. There might be a good reason why they are momentarily not active and it is recommended to discuss the situation with them before anything else is done.

When contacting a maintainer who is believed to be inactive, it is important to be polite. After all, due to the distributed nature of free software projects, one may not be aware what happened to a volunteer. One always has to keep in mind that there might be a good reason why they are not performing their duties. For example, something might have happened to a relative or close friend. What they need the least at this point is a hostile e-mail asking why they are not spending time on their volunteer activities. It is even possible that something has happened to the volunteer and that a relative will read their e-mail.

Additionally, when contacting a maintainer, it is important to keep in mind that we are all volunteers. While I firmly believe that a participant in a free software project has certain responsibilities once they have volunteered, it is not consistent with voluntary work to expect someone to respond to your inquiry within a day. Similarly, one cannot demand from volunteers that they perform their duties. All you can do is politely ask that they do so, or ask them to officially let go so other volunteers can take over their tasks.

With this in mind, it is time to contact the maintainer. A short message summarizing one's findings, for example stating which packages are unmaintained, is in order. In the e-mail, one can politely ask what their situation is, and whether they still have the time and interest to carry out their duties. If the maintainer does not respond within two or three weeks, another message can be sent. This should refer to the first message, note that nothing has since been done and suggest that the packages should be given away so other maintainers can take over. In this mail, it should also be mentioned that action will be taken if the maintainer does not respond or does something soon. Again, if the maintainer does not react after two or three weeks, it can be assumed that the maintainer is really not active anymore. In this case, their packages should be given away so other maintainers can take over. In Debian, there is a listing for this known as "Work-Needing and Prospective Packages" [8]. Through this system, Debian developers can indicate to other maintainers that they are no longer interested in a particular package and that they are looking for a new maintainer.

At this point, it is also important to think about authority in the project. In Debian, there is not a written document which explicitly describes the authority of the quality assurance team. From this point of view, it is not clear whether they possess the authority to take packages away from a maintainer and make them available to others. However, this is a very important task which has to be done in order to maintain the quality of the whole system.

Hence, the quality assurance team started looking for inactive maintainers and took their packages away. The task was performed very carefully so no active maintainer was mistaken as inactive. Over time, other members of the project acknowledge that the quality assurance team had the authority to perform this function. While the authority was firmly established over time in the case of Debian, it is important to discuss this matter before taking action.

6 Tracking Inactive Maintainers

In large projects, such as Debian, there are many volunteers and it is possible that a large number are inactive. All of them have to be contacted and records have to be kept of who was contacted at what time and what the current status is. At some point, the number of people who have been contacted reaches a number where it is no longer feasible to keep the information in one's head. Therefore, a system is needed which allows one to easily keep track of what has happened so far. Such a system additionally allows multiple people to work on this task and to coordinate their efforts.

To make this possible for Debian, I have implemented a simple system which aims to be an effective means of keeping track of developer activity. New information can be added by e-mail, and the status of a maintainer or package can be queried on the command line. This collection of utilities, collectively known as `mia`, consists of three core tools:

mia-record: This tool processes e-mail and stores the information. When an inactive maintainer is contacted, a blind copy of the message can be sent to a mail alias which pipes the message into `mia-record`. The tool will store the e-mail and ask for a summary of this message by e-mail. Once the e-mail is answered, a summary of the message is stored. Alternatively, an `X-MIA-Summary` header can be added to the original message and that summary is stored automatically.

mia-history: This tool shows the previously added summaries of a specific volunteer or of all volunteers who have been recorded so far. This tool is hence very useful to easily see what has happened already. For this tool to be helpful, it is crucial that the summaries supplied to `mia-record` are content-rich. Additionally,

it is useful if the summaries correspond to a specific schema so they can be easily parsed automatically. For Debian, I have introduced a set of arbitrary keywords which summarize different states. As an example, the output of `mia-history` might look like this:

```
foo:
    2002-01-06: Hint: bar
    2002-04-02: lost interest
    2002-04-03: Orphaned: bar
```

The person `foo` was contacted because someone gave a hint that the package `bar` was in a bad shape. The maintained responded saying that they are no longer interested in maintaining the package, and therefore the package was given away ("orphaned" in Debian's terminology).

mia-check: If proper keywords are used in the summaries, `mia-check` can automatically show which volunteer has any tasks outstanding. For this to work, different keywords belong to certain classes. The keywords "hint" adds an item to the TODO list of a volunteer. There are other keywords, such as "RC" (release critical bugs) or "S-V" (the Standards-Version is very old, i.e. the package does not conform to current policies), which add items. On the other hand, keywords such as "orphaned" or "removed" take an item from the TODO list. `mia-check` goes through all keywords, and if anything is left on the TODO list at the end and the volunteer has not been contacted recently, `mia-check` shows that this volunteer has to be contacted again.

While these tools are very simple, they make the work tremendously easier. They allow the quality assurance team to keep track of what has been done already. Data is stored with `mia-record`, queried with `mia-history`, and `mia-check` is a useful reminder showing outstanding tasks. Once a maintainer is contacted several times without response, it is time to act and make the packages available to other developers.

While these tools allow easy track keeping, I firmly believe that finding and contacting inactive maintainers cannot be fully automated in a sensible way. Some tasks, such as obtaining an overview of the packages of a maintainer, can be automated or at least semi-automated, but the whole process of identifying and dealing with inactive maintainers relies on human judgement. This makes the whole process very time consuming. However, in order to maintain the quality in a system inactive volunteers who play important roles have to be identified and solutions found.

7 Debian and Inactive Maintainers

Debian has recognized the problem of maintainers who neglect their packages and began approaching the issue systemically in 2001. Since then, various activities have been carried out. In particular, two different kind of activities can be distinguished. First, there is a continuous effort to track down inactive maintainers and to sort out solutions for their packages. Second, all maintainers without packages were contacted once to see whether they are still interested in volunteering for Debian.

7.1 Continuous Activities

Since 2001, there have been continuous activities to find and contact inactive package maintainers, and in case they do not respond to make their packages available to other developers or remove them if they are obsolete. From 2001 to the end of 2003, around 180 package maintainers were contacted, more than 320 packages were given away to other maintainers and around 25 packages were removed. These figures suggest that finding and dealing with inactive maintainers might have a real impact on quality.

7.2 The MIA Ping

In addition to the continuous activities carried out by the quality assurance team, the Debian Account Manager, who has the authority to add and remove volunteers from the project, conducted a 'MIA ping'. He contacted every Debian developer without a package in the archive in March 2003 to determine whether they are still interested in volunteering for Debian. If they did not respond to the mail after two months, their account would be deactivated and they would be put in an "emeritus" class. Recent compromises of GNU [10] and Debian [6], both involving local root exploits, show the importance of purging old users from project machines.

This MIA ping revealed that the e-mail of 34 volunteers was bouncing, 94 volunteers did not respond at all, 28 wanted to retire officially, 10 said they were still active, and 26 expressed their continued interest but it was not clear whether they would really do any work. It turned out later that the majority of those who were not sure about their involvement remained inactive. These numbers are very interesting because it was the first time in Debian's ten year history that anything like this was carried out. It helps to get a better grasp of the actual number of volunteers in a project, and improves security as unused accounts can be removed or locked.

8 Proposed Remedies

There are many factors which contribute to the severity of the problem of inactive volunteers. In addition to the distributed and volunteer nature of free software projects, tracking inactive volunteers is a complicated task because it cannot be fully automated. It requires human judgement to distinguish whether a volunteer is temporarily busy but likely to return or whether they are really inactive and neglecting their duties. Some

mechanisms for finding inactive volunteers can be automatated, but the whole tasks remains very time intensive. Furthermore, there is usually a long time span between realizing that a volunteer is neglecting their duties and something is done about it. As I argued before, it is polite so send several messages and to wait for two or three weeks for a reply to each message. Therefore, it can take months until the problem is resolved and the situation improves. During that time, the quality of the software suffers.

In order to keep quality high at all times, it is vital to take preventive measures. For example, the admission process should take the problem of inactive volunteers into account and make sure that prospective volunteers understand the problem. Another recommendation is to limit the introduction of low-interest packages into the distribution. A package that only one developer is even potentially interested in maintaining carries a large risk of being neglected or abandoned. Another possible way to assure quality is to have a dedicated group of developers who can temporarily maintain a package when its maintainer is busy. Finally, another recommendation for Debian is to move away from having a single maintainer per package to having teams who are responsible for a package [15]. This way, more redundancy is created and the reliance on a specific volunteer is smaller.

Fortunately, an increasing number of volunteer projects, including GNOME [21] and KDE [13], are paying more attention to quality assurance, and it is my hope that mature QA processes for free software projects will be developed in the next few years. It would also be beneficial if free software hosting sites, such as SourceForge, would actively classify inactive projects as such and had quality assurance teams.

9 Conclusions

Volunteers who neglect their duties are a potential problem in any free software project. In large projects, it is often difficult to recognize immediately when a volunteer does not carry out the tasks they are responsible for. Unlike in commercial companies where it is obvious when an employee does not come in to work, it requires much effort in distributed, volunteer projects to find who is inactive. While there are various sources of information which can be taken into account, the process of finding inactive maintainers and contacting them is time intensive and requires human judgement. It is therefore impossible to fully automate the task. The process of moving from realizing that a volunteer might be inactive to resolving the situation also takes a long time as it is polite to first contact the volunteer to see whether they can explain the situation. During this time, quality in the project decreases as the functions the volunteer is responsible for are not carried out. Due to this, it is important to take preventive measures and to consider this problem up front.

In this paper, I have described Debian's approach to the problem of package maintainers who do not perform their duties. Several sources of information have been introduced which are used in tracking down inactive maintainers. While these sources of information are specific to Debian, they help other projects gain a better understanding of the problem and this allows them to develop strategies which apply to their respective projects. Various tools have been described which are used to keep track of inactive maintainers in Debian and which allow multiple members of the quality assurance team to work on the problem together.

Free software projects have to acknowledge their volunteer nature and introduce more redundancy. They have to realize that certain volunteers will become inactive at some point, and take precautions. The problem of volunteers neglecting their duties has to be recognized and dealt with in order to maintain the high quality found in many free software projects as well as an effective development process.

10 Availability

The tools mentioned in this paper which are used in Debian to track inactive maintainers are available under the GNU General Public License (GPL) from `http://cvs.debian.org/mia/?cvsroot=qa`. Debian consists completely of free software as per the Debian Free Software Guidelines and is available from `http://www.debian.org/`.

11 Acknowledgements

This work was in part funded by the NUUG Foundation. I would like to thank Bart Massey for his valuable comments and suggestions.

References

[1] Debian Bug Tracking System. `http://bugs.debian.org/`, accessed April 6, 2004.

[2] Debian constitution. `http://www.debian.org/devel/constitution`, accessed April 6, 2004.

[3] Debian Developer's Reference. `http://www.debian.org/doc/developers-reference/`, accessed April 6, 2004.

[4] Debian New Maintainer process. `http://www.debian.org/devel/join/newmaint`, accessed April 6, 2004.

[5] Debian Policy. http://www.debian.org/doc/debian-policy/, accessed April 6, 2004.

[6] Debian Project machines compromised. http://www.debian.org/News/2003/20031121, accessed April 6, 2004.

[7] Debian Social Contract. http://www.debian.org/social_contract, accessed April 6, 2004.

[8] Debian Work-Needing and Prospective Packages. http://www.debian.org/devel/wnpp/, accessed April 6, 2004.

[9] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3), 1976.

[10] GNU Project FTP server compromised. http://lwn.net/Articles/44310/, accessed April 6, 2004.

[11] Jesús M. González-Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, and Vicente Matellán Olivera. Counting Potatoes: The Size of Debian 2.2. *Upgrade*, II(6):60–66, December 2001.

[12] Guido Hertel, Sven Niedner, and Stefanie Herrmann. Motivation of software developers in open source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7):1159–1177, 2003.

[13] KDE Quality Team. http://quality.kde.org/, accessed April 6, 2004.

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

[15] Martin Michlmayr and Benjamin Mako Hill. Quality and the reliance on individuals in free software projects. In *3rd Workshop on Open Source Software Engineering*, pages 105–109. ICSE, 2003.

[16] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[17] Glyn Moody. The greatest OS that (n)ever was. *Wired*, 4 August 1997.

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

[19] Eric S. Raymond. *The Art Of Unix Programming*. Addison-Wesley, 2003.

[20] Stephen R. Schach, Bo Jin, David R. Wright, Gillian Z. Heller, and A. Jefferson Offutt. Maintainability of the Linux kernel. *IEE Proceedings - Software*, 149(1):18–23, 2002.

[21] Luis Villa. Large free software projects and Bugzilla. In *Proceedings of the Linux Symposium*, Ottawa, Canada, 2003.