Ph.D. proposal

submitted to Prof. Brian Fitzgerald
Department of Computer Science and Information Systems
University of Limerick, Ireland

Workflow in distributed volunteer projects – Intuitive approaches to modern Debian package development

Martin F. Krafft <mail@martin-krafft.net>

13 Oct 2005

Abstract

This research paper/project details the technical challenges the Debian project faces as it continues its tremendous growth in size and popularity. It describes a research endeavour designed to increase the use of version control within the project for improved coordination of globally distributed teams of volunteers working on the software packages that make up the system. The research primarily focuses on the integration and consolidation of the involved processes. With tools already available for some parts of these processes as well as the coordination of teams, the goal is not to reinvent the wheel, but rather to "reuse and improve" these tools, to better integrate them, and to make them more accessible by providing abstraction wrappers with interfaces intuitive to Debian developers. It is further the intent for these tools to be optional and fully compatible to existing practices, thus not forcing developers to adapt. The research starts with process analysis and studies of the work habits on the side of the developers, and targets the final output of tools, which implement improved workflow in Debian package management through meaningful integration of existing (and proven) methods.

1 Introduction

1.1 Overview of the problem domain

The Debian project [2] is perhaps the largest, globally distributed computer project [27]. While its development over the past decade tells an astonishing success story, the project is beginning to groan under its sheer size [40]. Large parts of the Debian development process are accomplished with tools which have iteratively improved to meet new needs of the project. The immense speed

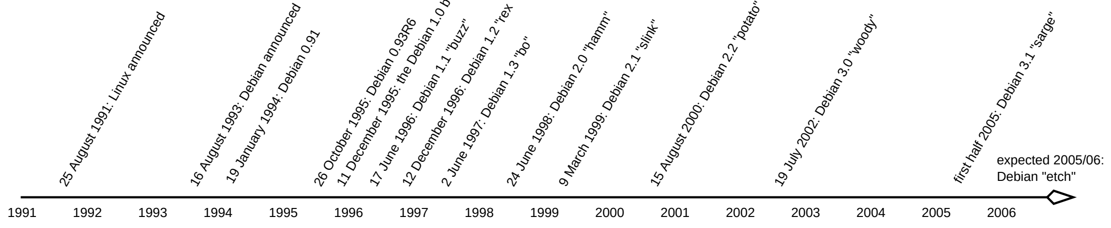

Figure 1: Debian has had difficulty dealing with its own growth, a problem most apparent in the increasingly long cycles between stable releases.

at which new developments take place, the size of the project, and the ever increasing popularity of the Debian operating system frequently make these tools obsolete and cause severe delays in the project. Figure 1 displays how the time span between subsequent releases has been increasing for the past seven years [41].

With scaling problems surfacing on a regular basis, Debian developers started to explore new techniques and approaches to cope with the challenge of growth. In a project where maintainers enjoy near absolute control over their software package or components of the infrastructure [21, ch. 6] infrastructure, issues frequently arise over bottleneck situations, when a single developer is unreachable, overloaded, or not willing to consider suggestions by others. For this reason, the Debian project has been identified as a "bazaar of cathedrals" [35] (*c.f.* [49]). One of the most notable trends in recent years is for developers to team up to co-maintain important software packages [52], and endeavours exist to formalise this development model [53].

The concern that a move from a single developer to team maintenance model is an "unrealistically simplistic solution" to bottlenecks [43] is a valid one: such a move brings with it a sudden increase in complexity of communication and coordination [19]. The classic response to such an increase in coordination complexity is the deployment of tools from the wide domain of computer-supported cooperative work [28]. Apart from the extensive and multifarious communication media in active use [21, ch. 6], the open source community relies heavily on software configuration management (SCM) and version control systems (VCS)[1] [10, 42, 33], such as CVS [26], which are part of the large set of CSCW tools. Nevertheless, as shall be argued further down, the available tools, that are applicable to Debian package maintenance, suffer from usability problems themselves, generally adding complexity for the individual and making it harder to contribute.

1.2 Statement of the problem

The tendency for team development in Debian calls for tools that help eliminate bottlenecks, rather than create new ones. Additionally, reducing the complexity and standardising a new, flexible approach to package management should make it easier for one-time/infrequent contributors to donate bits of their time here and there, without having to first acquaint themselves with a

[1]Even though technically a subset of software configuration management (SCM), VCS is often used interchangeably with SCM in the F/OSS world. Other synonyms include source code management, source control, and revision control systems (RCS) [60].

different set of tools for each package.

From a technical point of view, modern VCS are capable of coordinating even the largest F/OSS projects [32, 44]. In fact, several developers are already maintaining their packages under source control, a practice backed up by the existence of tools such as `cvs-buildpackage`, which strives to automate most aspects of building a package maintained in a repository. Because the choice of VCS can be religious to some users, every other major source control system has a corresponding package with tools to aid the building of packages maintained in the system (*e.g.* `svn-buildpackage`).

The usage of the aforementioned building tools requires the developer to strictly adhere to a structure laid on top of the VCS mechanisms and generally leaves the maintainers exposed to the intricacies of each system, rather than to keep out of the way and allow them to maintain their packages. Furthermore, with four of five of these build tools available, a developer familiar with one may not be able to easily contribute or help out with maintenance of a package using another one of these tools.

As will be illustrated below, the act of building packages is only a small component of Debian package management. Other pertinent tasks include patch management, change log keeping, correlating changes with entries in the bug tracking system, unit and regression testing, checking for policy compliance, and uploading the package. At present, these tasks are separate processes with little or no integration. Each process may further be governed by preferred practices by the primary package maintainer. As a result, package maintenance leaves a lot of room for human error, and requires a strict set of guidelines to be followed by all contributors in the case of team-managed packages.

As Debian continues to grow, the project will have to deal with the problem of bottlenecks and rely more on distributed contribution [24]. At present time, several approaches to Debian package management in teams exit. However, none can do without a set of guidelines or best practices, and none integrate the separate sub-processes in a way to eliminate redundancy or reduce the manual (and thus error-prone) work required from the maintainers in a meaningful way. Moreover, having different, incompatible approaches will limit domain of potential contributions for each volunteer as few will go through the trouble of learning different methods to accomplish the same task.

The existing Debian package management techniques have grown with the project for over a decade. The project has now reached a complexity that calls for a reassessment of the approaches. As part of the research I propose, I plan to take a step back and approach Debian package management — possibly the most important collection of processes within the project — with the goal of integration. However, instead of drafting a high-level model and designing the tools accordingly, I want to emerge them from existing habits and processes that seem intuitive to developers. The tools must be optional and compatible with existing approaches, because it is not possible to force volunteer contributors to adopt new methods [40].

2 Version control and Debian

2.1 An overview of source control tools

The VCS market is filled with several dozens of tools [60, 45], which all differ in terms of origin, motivation, underlying model, and target user group. One rough division can be made between centralised and distributed systems. In the class of the centralised VCS reside `CVS` , the established standard in VCS in the open source community, despite its many defects [61], and `Subversion` [22], which was designed to replace `CVS` and addresses many of its problems while trying to adhere to its interface where possible. Both, `CVS` and `Subversion`, rely on a central server instance to coordinate the cooperation between developers. Developers obtain the source code from this server, commit changes to it, and update their local trees to pull in changes committed by others. While read-only access is possible anonymously, most regular operations on the repository require an account. Thus, each additional contributor calls for administrative intervention, and a certain level of trust has to exist. The entrance barrier for contributors to projects using centralised VCSis can therefore be high [55]. A frequently raised criticism of centralised VCSs is the difficulty for contributors to effectively work on the code when offline.

In a distributed system, a central server is not needed. Instead, a repository is published on *e.g.* a web server, and a developer interested in hacking the code creates a branch of the repository, which is stored on the local hard disk. A branch is a fork from a code base at a certain point in time, but it differs from a fork in that branches are designed to facilitate merging changes from one branch into another. Differently from a centralised system, the branched tree is a repository of its own, and the developer can now edit the code, commit, revert, and inspect, until a milestone is reached. Then, the developer makes the repository publicly available, and announces the location to *e.g.* a mailing list. Interested developers can merge the changes from the branch, while the VCS keeps track of which changes have been merged, and when. The classic approach to distributed source control relies on communication media for announcements of new branches. Most distributed systems, however, are also designed to allow their use in a centralised fashion.

Distributed VCSs are usually centred around branching and merging, and exceed centralised VCSs in terms of capabilities and robustness of these two methods. Possibly the best known distributed tool is `GNU Arch` with its command-line interface `tla` [39]. `tla` suffers from a fair share of problems related to performance, technical soundness, and usability. The `bazaar` project forked from `tla` to address these problems, but it was soon found that `Arch`'s model was inherently flawed in ways that would make a user-friendly interface next to impossible [13]. As a consequence, the `Bazaar-NG` project was born. Of further interest may be `git`, the VCS tool developed by Linus Torvalds for the Linux kernel, and its descendant `mercurial`, as well as `darcs`, a VCS optimised for small projects; all three are distributed systems.

2.2 A brief overview of Debian package management

Before inspecting the different approaches to source control, it helps to take a brief look at the processes behind Debian package management. The nature of the work performed by Debian developers is different from conventional software engineering. The task of a Debian developer is that of integration rather than the development of software *per se*. Debian developers take public releases by software projects, such as Apache or KDE, compile them, and assemble them into packages according to a strict set of rules to ensure the coexistence of thousands of software packages on a single system [8].

When a software has been successfully packaged, it is uploaded to the Debian archive and globally distributed to hundreds of mirror servers. The upload marks the beginning of the regular package lifecycle. The package will first spend some time in the `unstable` archive, and when it meets a set of basic consistency criteria, it propagates to the `testing` archive, from where the set of packages included in a `stable` release is sampled. Users are able to install software from any of these three archives, and usually, a number of bugs are found, which are reported to the Debian bug tracking system [1]. The package maintainer then coordinates with the upstream author to fix the reported problem(s) and uploads a new package to the `unstable` archive, which eventually replaces the buggy version. This process is detailed in [35, ch. 4].

Debian developers usually depend on the software they package and thus have special interests to maintain a high level of quality. In addition, many developers add features to the software for their own use, or integrate improvements, enhancements, and bug fixes from other contributors. Largely to lessen their own maintenance load, the Debian developers make all such changes available to the upstream author, in the hope that they might be integrated into the original source code and thus need not be maintained specifically for Debian. This practice is described in Debian's Social Contract as a service to the F/OSS community at large, who can profit from improvements made by Debian without having to use the Debian operating system itself [6].

The requirements of a VCS for Debian packaging differ substantially from the requirements for the development of an executable application or library. For simple packages, it suffices to commit the Debian-specific control files to a versioned repository. However, as soon as changes to the upstream source are required to fix bugs, or new features are to be implemented, the original source code better also be under the control of the VCS. Since upstream author and Debian package maintainer do not usually share the same repository, the maintainer is faced with the challenge to import each upstream release into the package's repository while maintaining changes made to previous versions.

The challenge in the deployment of source control mechanisms for Debian package management is thus to find a way to maintain the upstream source code, improvements and bug fixes, and Debian-specific changes in one place, while being able to easily separate different sets of changes to make it possible to contribute contained versions to the upstream project. An elegant approach to meet this challenge is through the use of branches, which are best handled by distributed VCSs.

2.3 Status of source control in Debian

In the Debian project, VCS tools are having a slow start. For a long time, only a small number of projects used the official Debian `CVS` server, which can only be used by Debian developers. In early 2003, the Debian project introduced `Alioth` [29], a management service for Debian-related services similar to SourceForge [3]. Among communication tools, such as web space, mailing lists, and bug and request trackers, `Alioth` hosts `CVS`, `Subversion`, and `Arch` repositories. At the time of writing, 600 projects were registered with the `Alioth` server, and around 200 of them used one of the source control mechanisms available, with the majority on the side of `Subversion`. With around 10 000 source packages in the Debian archive, the number is strikingly low.

The use of these tools ranges from simplistic management of the Debian-specific control files, which are then manually merged with the upstream source before a release is made, to generic methods of handling packages of virtually any complexity, exploiting the advanced branching and merging capabilities of distributed VCSs [54, 36]. The tendency in Debian package management seems to be towards distributed VCS, partly because of the need for a network connection for effective use of centralised VCSs, which makes it harder for developers to contribute when not online. Distributed systems also seem to fit better with the distributed, global nature of Debian development.

The slow acceptance of source control tools in Debian is probably due to several reasons. First and foremost, developers are currently reluctant to invest time in centralised systems, given the aforementioned trend. The distributed methods, however, are still too novel and young for a developer be able to judge one system over another. Several "lead users" [59] are pursuing orthogonal strategies and other developers prefer to let them sort out the intricacies before jumping on the bandwagon.

A large part of the Debian archive consists of trivial packages with little or no upstream activity, and few Debian-specific modifications, if any. For those packages, as well as ones maintained by single individuals, where a VCS is not essential, no motivation exists to migrate the source package to a VCS repository. Nevertheless, even packages with two or more maintainers often do not use VCS. Common practices in such a situation include turn-taking and sending patches back and forth. While this procedure tends to work fine most of the time, (temporary) disappearance of a developer, which is quite common in volunteer projects [40], can cause maintenance of packages to stall. The lack of revision history or a proper patch management system make it difficult for new maintainers to pick up where others left off, and one-time/infrequent volunteers frequently do not even know where to start.

Another important reason for the slow adoption of source control for Debian package management is the inertia on the side of the maintainers, who have maintained packages the traditional way for a longer time and would rather employ quirks than to invest time in learning a new system. Several developers cling to standard Unix tools they can expect to find on any compatible system, rather than to embrace helper utilities designed to automate repetitive tasks in Debian package management (*c.f.* [54]). One of the most important motivations driving Debian developers is that they are allowed to do what they want, as long as they observe basic rules of conduct [7], try to follow best practices [11], and their packages adhere to the Debian policy [8].

2.4 Usability problems: bottom-up vs. top-down

By far the largest reason that Debian package management and VCSs are still unwed, however, seems to be the low usability and steep learning curves of VCS tools[2]. Perhaps these reasons explain why `CVS` continues to be the most widely used system, despite the technical limitations: it is moderately easy to learn, and it is not hard to find assistance. When the Debian installer team decided on a VCS to manage the new installer, one of the reasons they chose `Subversion` was because a large part of the hundred contributers was already challenged enough by the usage paradigm of `CVS`, so an even more complex system would have been counter-productive [30].

[2]A formal usability/acceptance study in the spirit of [25] does not seem to exist.

This is in line with Raymond's claim: "the number of contributors [...] is strongly and inversely correlated with the number of hoops each project makes a contributing user go through" [50]. Specific to the domain of VCS, Hudson adds: "In many environments, a shallow learning curve is the most important feature of a VCS. [...] [A] steep learning curve [is one of] fundamental and insurmountable obstacles." [31]. Translators, for instance, are motivated to contribute to a project such as the Debian installer to help people from their cultural background use Debian in their native language; they are not (necessarily) keen on learning how to interact with a VCS, and then to apply for commit access to the project's repository, only to be able to do their work (*c.f.* [55]).

The low usability of source control tools is a result of several factors. First and foremost, VCS itself is subtle and non-intuitive [38], and requires perhaps more of an understanding of the underlying model from the users than other tools [48]. Therefore, it is necessary to distinguish between usability as experienced by the uninitiated user, and usability experienced by the acquainted developer.

The two centralised tools, `CVS` and `Subversion`, seem to form the basis for both groups. Their underlying model for basic usage is accessible to most, the user interface is consistent (if awkward at times), the tools are vastly documented, and support is readily available as most every acquainted developer will have had to use either of these tools at one point in time. `Subversion` is still a step up from `CVS` and thus the first choice for projects looking to deploy source control.

When it comes to distributed systems, however, the experiences of new users and experienced developers readily diverge. First, the underlying concepts are harder to learn [31]. And second, the tools are generally younger and thus not too well documented, known, or supported[3].

Furthermore, the distinguishing features of distributed VCS systems exceed the basic tasks like updating and committing. In distributed source control, other topics are at the focal point, such as conflict resolution, three-way merging, repository aggregation, and complex branching strategies. Advanced concepts such as the aforementioned are primarily of benefit to experienced developers taking part in projects of greater complexity [9]. Usability tends to play a less important role for advanced users, as they are usually familiar with their tools and have learnt their paradigms, or are generally more capable of abstraction in this domain [46, 57]. Therefore, the demand for technical correctness and flexible features is much larger than the desire for usability. Second, advanced users are also often contributors to the tools they employ, or develop tools that meet their specific needs (*c.f.* Torvald's `git`, and [20]), which results in "top-down tool design": the development is driven by the theoretical model underlying the VC model, and the designer moulds the code until it fits the model [14, 15].

Tools designed to meet a certain purpose require users to learn to bridge the gap between their intuition and the user interface or model of operation. Quite often, this gap decides between success and failure of a tool. A good example for the lack of usability leading to the failure of a tool would be the `tla` implementation of `GNU Arch` (and all its ancestors), which quite possibly has the worst user interface across all VCS tools. This is further backed by the existence of `bazaar`, a fork from `tla`; `bazaar` does not add features but merely aims to remove the awkwardness from `tla`, and to make it more usable. It is not surprising that `tla` never got off the ground, and due to design deficiencies in `GNU Arch` itself, the days of `bazaar` are also numbered (as `bazaar-ng`,

[3]Documentation does exist for the major tools, but it is not always to be found easily. Also, it is usually targeted at the experienced user or just serves as a scratch pad for the developers to dump their thoughts.

a compatible but otherwise clean-sweep re-implementation, emerges as successor), while the development of `GNU Arch 2.0` seems to have been discontinued.

If usability is the goal, simple top-down approaches will not succeed, and a design strategy must include a bottom-up, user-centric component [56]. The balance between approaches decides whether the result is "usable but not useful" [16], or *vice versa*. As previously noted, however, an understanding of the conceptual model of VCS is indispensable for users of VCSs at any level. Therefore, traditional bottom-up approaches, such as user process analysis [56], are not going to succeed without the user acquiring knowledge about the theory of VCS. To be able to understand this theory requires users to be acquainted with software development processes, at which point they are not beginners anymore. This Catch-22 situation makes it unsurprising that VCS has been identified as a "wicked problem" [51], that is a problem which is not understood until after the formation of the solution, a problem of which stakeholders have radically different views, a problem with constraints and resources changing over time, and a problem that is never solved [23].

This does not intend to suggest that usability in source control tools is hopeless. On the contrary, the development teams behind modern tools, such as `Bazaar-NG`, are making an effort not to forget about usability. A motto such as "Bazaar-NG should be a joy to use." [47] may be sound like a vacuous slogan from the marketing department, the actual studies around usability and the several rounds of mock-ups of the tool seem to suggest otherwise; at the very least, they allow for a spark of hope. `Bazaar-NG` aims to address usability problems by reducing the number of concepts for a simplified mental model, and by providing a consistent, modular user interface with only a handful of commands. Furthermore, the developers tried to stay true to the conventiones of existing VCS to facilitate migration, and to speed up essential processes for instant gratification, which increases acceptance. The development of `Bazaar-NG` is sponsored by Canonical Ltd., the company behind the Ubuntu operating system, a Debian-derived distribution.

3 Proposed research

3.1 Overview

The `Bazaar-NG` team is producing a tool what is technically sound, as well as engineered with usability in mind. However, it still requires a general understanding of the model of the VCS to be useful [48], and just like any other VCS, it depends on the users to make decisions about the organisation of files in the tree under version control, their discipline to meticulously commit changes, and their understanding of the VCS to allow them to extract information. The reason for all these points stems from the genericity of a VCS, which is designed to bookkeep essentially any type of file data.

Conceptually, a layer on top of the VCS, which hides the actual VCS is possible; a user would not tell the VCS to commit a file, or to create a branch, but rather issue commands such as "put this project tree under version control," "I want to add the feature *foo*," or "prepare everything for a release." However, VCSs are already at a fairly high conceptual level, and tools that implement even higher concepts are bound to make decisions or assumptions about the data they control, thus losing genericity. In Debian package management, however, the genericity of VCSs is not needed and assumptions can be made for the highly specific domain, as certain operations would always follow more or less the same patterns.

I thus propose to research the process of Debian package management from the point of view of developers and team maintenance, with a focus on VCS. The goal is the design and implementation of a set of tools, which automate Debian package management, while harnessing the coordinative power of VCSs to allow teams of developers to work together on specific components of the Debian operating system. Rather than drafting an abstract usage model and expecting developers to accept and learn the model — an approach that would almost certainly fail — I want to study the habits and processes of developers with the goal to create tools that bridge the gap between Debian package management and version control.

Starting small, the idea is to successively integrate sub-processes into a coherent procedure while reducing redundancy and the amount of manual (and error-prone) work and concentration required from contributors. Many projects seem to suggest to "start small" and yet fail because their implementers cannot rid themselves of the big picture and the bias that comes from their own interpretation of the problem and the solutions they envision. I intend to circumvent this problem with a bottom-up approach, situated in the position of individual developers. Debian package management uses proven concepts and approaches, and to attempt to revolutionise these approaches would thus jeopardise the robustness of its sub-processes and risk losing compatibility with the hundreds of existing tools. Instead, I want to optimise the existing approaches through integration, and by hiding those repetitive processes that come with but do not constitute a conceptual part of the Debian packaging endeavour.

A major point of focus will have to be the compatibility of any new tools and approaches with existing methods, for a number of reasons: First, acceptance of novel techniques in Debian is slow, so a transitional period of several years is to be expected, and some users will never switch away from their own way of packaging.

Second, Unix users, and Debian developers especially, like to stay in control and frown upon monolithic solutions with a lot of obscure, inaccessible magic; the entire Debian operating system is built in accordance with the Unix philosophy of modularity, and much of Debian's robustness and professional reputation comes from its numerous small utilities that do exactly what they should (and nothing more), and which are purely optional in that the steps they take to accomplish a certain task are standardised and/or well-documented, such that no developer is forced to use them.

The third point calling for new tools to maintain compatibility is the tight interdependence between the Debian infrastructure and the processes of Debian package management. A "revolutionary" approach would require parts of the infrastructure to adapt, which, given the size and complexity of the project and its infrastructure, would simply not be possible.

Fourth, because the goal to produce a standardised approach to Debian package management is somewhat idealistic, it is important to produce compatible alternatives and gradually attract users through functionality rather than to set out writing "the standard;" processes cannot be made standard, they become standard, and this is especially true in a volunteer project such as Debian.

Finally, using existing and well-known building blocks will facilitate contributions by others, and hence make development of these tools possible in true F/OSS fashion.

3.2 The role of Canonical and Ubuntu

Canonical, the company behind the Ubuntu operating system [4], a Debian derivative, has also investing months of work into *Launchpad*, an infrastructure designed to unify all aspects of Debian packaging (bugs, translations, enhancements) behind a single interface. As part of the endeavour, Canonical is also sponsoring the so-called *super mirror*, a server with the goal to host *all* active F/OSS projects' produce imported into `bazaar` (soon: `Bazaar-NG`) repositories. Based on this giant backbone, the company is now working on `hct` (the "hypothetical changeset tool") as a single user interface to package building, which attempts to hide the complexity of flexible package management in teams behind a simple user interface.

For a number of reasons, Debian developers are unlikely to adopt `hct`. First, it is a delicate task to get Debian developers (or volunteers in general) to adopt methods that deviate too much from their traditional solutions [40], and frictions between Debian and Ubuntu will make it even less likely. Second, and perhaps more importantly, the Launchpad backbone provided by Canonical is available for free use, but the company restricts access to the raw data, which is frowned upon by many developers (*e.g.* [18]), and seems to bite with the ideology of freedom that drives many of the Debian developers [5]. Third, the `hct` and Launchpad combination is a centralised system and thus suffers from the same problems that *e.g.* centralised VCSs experience (see above, but also [17]). And lastly: `hct` and `Launchpad` are at the core of Canonical's business plan, and are hence not going to primarily focus on and follow the interests of Debian developers [58]. This is exemplified by recent efforts to add distributed abilities to `hct`, which are, however, not endorsed by the management.

Thus, it is unlikely that Debian developers will adopt Launchpad/`hct` for their packaging practices. The Ubuntu project, however, continues to play an important role. As the most successful Debian derivative, with about 40 people paid by Canonical to work full-time on the operating system, Ubuntu is a source of improvements that Debian must not ignore. Any strategies implemented by Debian which fail to integrate Ubuntu (and other derivatives) at this stage are bound to failure, or will introduce new problems. Therefore, my intent is to closely work with the `Launchpad/hct` developers to ensure compatibility of both methods[4]. Furthermore, any surveys and studies of habits will include Ubuntu developers (as well as developers from other derivatives) in the test groups.

3.3 Specifically targeted areas of improvement

Debian package management is a very broad field, and one without any "right answers." It is thus difficult to isolate specific areas that need to be improved. The lack of VCS is certainly a huge gap because it puts the burden to coordinate between multiple developers on those developers. But even with a VCS in place — several sub-projects in Debian have been using VCS for years — improvements can be made in a variety of ways. The following paragraphs give some ideas, and more improvements are expected to become visible throughout the research endeavour.

Every Debian package is required by policy to contain a change log identifying modifications between Debian revisions of a package, but not including changes made by upstream. Good packaging practice includes keeping this log meticulously up to date. Version control systems can

[4]I have successfully applied to become an Ubuntu developer for this reason.

attach log messages to changes in the repository, so it would only be logical if those log messages could be trivially reused for the Debian change log. No tool currently exists to automate this process in a meaningful way.

The Debian change log is also used to interact with the Debian bug tracking system [1]. For instance, the recommended procedure to mark a bug as resolved is to include the bug number in the change log next to the relevant entry, which will automatically cause the bug to be closed when the package has been successfully updated.

At the same time, the bug tracking system is often used as a request tracker and coordination platform for single developers or maintainer teams. Currently, the bug tracking system provides for no integration with VCS mechanisms. For instance, patches attached to reports filed with the bug tracker need to be manually imported into source control, and vice versa, changes committed to the repository addressing a certain bug are not made available as part of the bug report, which is often a helpful resource for users encountering problems.

Furthermore, the Debian archive infrastructure is unaware of VCSs. Thus, the upload of a new version of a package currently requires a developer to check out the source from the repository, build the package locally, and then upload it to the Debian archive. This process can potentially introduce security problems [34], and developers with slow or unreliable Internet connections find it difficult to upload larger packages. In [37] exist some preliminary thoughts on how to improve the situation through the integration with VCSs, while paying particular attention to quality assurance.

Lastly, Debian's infrastructure encompasses several dozens of servers, as well as the most extensive mirror network in the F/OSS world. Most of the infrastructure (and the mirrors especially) are designed in a decentralised fashion to reduce bottlenecks. An integrated package management system designed for team maintenance should harness the power and availability of this infrastructure.

3.4 Plan of research and research methodology

The proposal suggests a bottom-up approach to process integration and workflow improvements in Debian package management. Here, bottom-up refers to the intent to use habits of developers as a basis and work upwards to the conceptually high goal of an integrated workflow. Rather than expecting developers to adopt a new mental model, any solution must be usable by developers without much adjustment. The research plan consists of three stages, with a forth stage corresponding to the release of the results to the community in form of an open source project with the hope that it shall foster from this point on.

3.4.1 Stage 1: surveying and studying habits

During the first stage of the research, I intend to survey the fields of version control, Debian package management practices, as well as approaches to process integration found in other, comparable projects (*e.g.* the BSD operating system family, in which version control is used extensively). The results will be used to devise strategy that maximise reuse of existing tools in true Unix spirit, and to gather ideas for possible integration and consolidation steps, as well as experience with pitfalls and successful migration strategies. Looking at comparable projects, it is my hope to learn from their approaches, but also to identify problems and pitfalls their use has uncovered to try to guard against them in later stages of the proposed research.

Also part of this stage is the studying of developer habits. I intend to conduct numerous interviews with developers to determine the processes with which they are familiar, and thus to define a set of methods that can be expected from Debian developers on all levels.

In a further step, I then want to study how developers would like to approach certain tasks. A part of this study will include "live command-line sessions," in which I pretend to be the "system," while the interviewed developer pretends to steer the system with high-level commands. The following is an example interview held over IRC (Internet Relay Chat) as part of the writing of this proposal:

```
<oracle> i want to work on the mutt package
<martin> downloading mutt (1.5.9-2) to ./mutt-1.5.9 ... done.
<oracle> i want to fix bug #654321
<martin> preparing isolated environment for fixing of bug #654321 ... done.
 * martin records intent ...
 * oracle edits a file, fixes the bug
<oracle> i have fixed the bug
<martin> thank you. please give a brief description of the bug fix.
 * martin spawns editor-of-choice
 * oracle describes the bug fix and closes editor
<martin> fix recorded in change log, what else can i do for you?
<oracle> please prepare a new release
<martin> processing release of mutt, version 1.5.9-3 ...
 * martin runs any unit tests ... success.
 * martin integrates fix with distribution source ... success.
 * martin prepares a new package file in ../mutt_1.5.9-3_i386.deb ... done.
 * martin runs policy checkers and other tests against the package ... success.
<martin> new package ready for release in ../mutt_1.5.9-3_i386.deb .
<oracle> please upload the package to unstable
 * martin verifies authenticity and authorisation of requester ... success.
<martin> sealing revision 1.5.9-3 ...
<martin> upload request sent to archive daemon.
```

This transcript describes a typical procedure in Debian package management, but it does not mention VCS at all; all interaction with the VCS mechanisms is hidden behind the high-level wrapper, which will have the interface of a tool designed according to the human language requests formulated by the developers. This method of real-time research over IRC allows for very specific, adaptive studies into the habits of individual developers.

With the data from such live sessions, patterns in management procedures can be identified. Furthermore, screening the results will give a good indication of what approaches developers consider intuitive.

3.4.2 Stage 2: formalisation of processes and mock-up implementation

In the next stage, the existing processes will be described formally, as well as the processes condensed from the studies on developer habits. A formal representation of the processes should allow for the identification of potential areas for integration and consolidation.

In this stage, I will also begin with the implementation of mock-ups to be able to determine and possibly improve acceptance from the side of the developers. These mock-ups are going to be "empty hulls" in that they provide a user interface, but no internal functionality (other than fake feedback and status messages to the user).

Feedback from the users will help improve the intuitive feel of the user interface in the context of Debian package management.

3.4.3 Stage 3: deployment in projects and iterative improvements (XP)

The third stage starts with the implementation of functionality into the tools, after having settled on a user interface in the second stage. Here, the intention is not to implement tools and release finished versions, but rather to take the "release early, release often" approach [49] and draw from extreme programming [12] to implement and iteratively improve the tools in cooperation with select developers and in the context of some packaging projects within Debian[5].

As soon as developers start adopting the proposed tools, further integration and improvements can happen independently in distributed fashion.

3.5 Expected impact

Since this research uses a user-centric approach, it increases the probability of acceptance among developers. By standardising the Debian package maintenance process, it will become easier to coordinate teams of developers to work around the bottleneck problems the Debian project is experiencing due to overloaded or inactive developers holding a monopoly on packages that use peculiar packaging approaches and are thus not accessible to others.

With an intuitive interface on the side of the developer, it will be easier to adjust processes to new policies or guidelines, and it will be possible to consolidate and integrate processes further for improved workflows. This should improve Debian's scalability.

Furthermore, a standardised method will make it easier for one-time/infrequent contributors or non-technical contributors (such as translators) to donate their expertise without having to invest time into learning package-specific techniques or guidelines.

This research is relevant because it will help Debian scale and retain high technical standards. This will benefit not only the countless users, who rely on the technical strengths of the Debian operating system, but also to the more than a hundred derived distributions, which help to diversify the Linux distribution market and thus drive innovation.

Furthermore, results from this research will be useful to other projects who face the task of coordinating small teams on sub-projects characterised by contributor fluctuations typical to volunteer efforts. Quite possibly, the results could also influence business processes in companies with flat hierarchies.

[5]As the instigator of the `pkg-zope` project, I am in a good position for field testing. Moreover, Zope packaging in Debian has been catastrophic prior to the inception of `pkg-zope`, which has caused many users to prefer upstream releases over the Debian packages. Therefore, the possibilities of experimentation in `pkg-zope` are greater than in projects with a large user base.

4 Acknowledgements

I would like to thank Biella Coleman for her assistance in writing this proposal. Further thanks go to Martin Pool, Aaron Bentley, Nathaniel Smith, James Blackwell, and several others on the `#revctrl` and `#bzr` IRC channels (`irc.freenode.org`) for their input and interesting discussions. And, of course, Debian…

References

[1] Debian Bug Tracking System. http://bugs.debian.org. Last accessed: 3 September 2005.

[2] The Debian project. http://debian.org. Last accessed: 4 September 2005.

[3] Sourceforge. http://www.sourceforge.net. Last accessed: 1 September 2005.

[4] The Ubuntu project. http://www.ubuntu.com. Last accessed: 4 September 2005.

[5] The Debian Free Software Guidelines. http://www.debian.org/social_contract#guidelines, April 2004.

[6] Debian Social Contract. http://www.debian.org/social_contract, April 2004. Last accessed: 1 September 2005.

[7] Debian machine usage policies. http://www.debian.org/devel/dmup, August 2005. Last accessed: 1 September 2005.

[8] The Debian Policy Manual. http://www.debian.org/doc/debian-policy, June 2005. Last accessed: 1 September 2005.

[9] ASKLUND, U. *Configuration Management for Distributed Development in an Integrated Environment.* PhD thesis, Department of Computer Science, Lund Institute of Technology, Lund University, Lund, Sweden, 2002.

[10] ASKLUND, U., AND BENDIX, L. A study of configuration management in open source software. *IEE Proceedings - Software 149*, 1 (February 2002), 40–6.

[11] BARTH, A., CARLO, A. D., HERTZOG, R., AND SCHWARZ, C. Debian developer's reference. http://www.debian.org/doc/developers-reference, August 2005. Last accessed: 3 September 2005.

[12] BECK, K. *Extreme Programming Explained: Embrace Change*, 1st ed. Addison-Wesley Professional, Cambridge, MA, USA, October 1999.

[13] BENTLEY, A. How is Bazaar-NG related to Bazaar and Arch in general? http://article.gmane.org/gmane.comp.version-control.bazaar-ng.general/1104/, June 2005. Mailing list post.

[14] BERCZUK, S. P., AND APPLETON, B. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Addison-Wesley Professional, Boston, MA, USA, November 2002.

[15] BERLACK, H. R. *Software Configuration Management*. Wiley Series in Software Engineering Practice. John Wiley & Sons, Hoboken, NJ, USA, September 1991.

[16] BEVAN, N. Measuring usability as quality of use. Tech. rep., National Physics Laboratory Usability Services, Teddington, England, 1995.

[17] BICKING, I. Distributed vs. centralized version control. http://blog.ianbicking.org/distributed-vs-centralized-scm.html, August 2005.

[18] BREITNER, J. Launchpad, Google and why Microsoft is not the problem. http://www.joachim-breitner.de/blog/archives/60-Launchpad,-Google-and-why-Microsoft-is-not-the-problem.html, July 2005.

[19] BROOKS, F. P. *The Mythical Man Month: Essays on Software Engineering*, anniversary ed. Addison-Wesley Professional, Boston, MA, USA, February 1995.

[20] BUCCIARELLI, L. L. *Designing Engineers*. MIT Press, Cambridge, MA, USA, 1996.

[21] COLEMAN, E. G. *The Social Construction of Freedom in Free and Open Source Software: Hackers, Ethics, and the Liberal Tradition*. PhD thesis, University of Chicago, Chicago, IL, USA, August 2005.

[22] COLLINS-SUSSMAN, B., FITZPATRICK, B. W., AND PILATO, C. M. *Version Control with Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, June 2004.

[23] CONKLIN, J. *Dialog Mapping: An Approach for Wicked Problems*. PhD thesis, CogNexus Institute, Napa, CA, USA, 2003.

[24] DAFERMOS, G. N. Management and virtual decentralised networks: The linux project. *First Monday 6*, 11 (November 2001). Last accessed: 30 August 2005.

[25] DAVIS, F. D. User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. *International Journal of Man-Machine Studies 38* (1993), 475–87.

[26] FOGEL, K. F., AND BAR, M. *Open Source Development with CVS*. Paraglyph Press, Phoenix, AZ, USA, July 2003.

[27] GONZÁLEZ-BARAHONA, J. M., ROBLES, G., ORTUÑO-PÉREZ, M., RODERO-MERINO, L., CENTENO-GONZÁLEZ, J., MATELLÁN-OLIVERA, V., CASTRO-BARBERO, E., AND DE-LAS HERAS-QUIRÓS, P. Analyzing the anatomy of GNU/Linux distributions: Methodology and case studies (Red Hat and Debian). In *Free/Open Source Software Development*, S. Koch, Ed. Idea Group Publishing, Hershey, PA, USA, 2004, pp. 27–58.

[28] GREIF, I. *Computer support for cooperative work:*. Morgan Kaufmann Publishers Inc., San Mateo, CA, USA, 1988.

[29] HERTZOG, R. Introducing Alioth: SourceForge for Debian. http://lists.debian.org/debian-devel-announce/2003/03/msg00024.html, March 2003. Mailing list post.

[30] HESS, J., September 2005. Personal communication.

[31] HUDSON, G. Undiagnosing subversion. http://web.mit.edu/ghudson/thoughts/undiagnosing, January 2004. Last accessed: 2 September 2005.

[32] IANNACCI, F. Coordination prrocesses in open source software development: The linux case study. Tech. rep., Department of Information Systems, London School of Economics, London, England, April 2005.

[33] JOERIS, G. Change management needs integrated process and configuration management. In *Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97)* (Heidelberg, Germany, 1997), M. Jazayeri and H. Schauer, Eds., Springer Verlag, pp. 125–41.

[34] KRAFFT, M. F. Rebuilding packages on all architectures. http://lists.debian.org/debian-security/2004/09/msg00014.html, September 2004. Mailing list post.

[35] KRAFFT, M. F. *The Debian System — Concepts and Techniques*. Open Source Press, Munich, Germany, June 2005.

[36] KRAFFT, M. F. Managing pkg-zope packages with GNU Arch. http://pkg-zope.alioth.debian.org/wiki/Arch/Package, August 2005. collaborative wiki, others have contributed.

[37] KRAFFT, M. F. Thoughts on a new upload process. http://blog.madduck.net/debian/2005-08-11-rcs-uploads.html, August 2005. Last accessed: 4 September 2005.

[38] LORD, T. Diagnosing Subversion. http://web.mit.edu/ghudson/thoughts/diagnosing, February 2003. Last accessed: 31 August 2005.

[39] LORD, T. GNU Arch. http://www.gnuarch.org/arch, 2005. Last accessed: 1 September 2005.

[40] MICHLMAYR, M. Managing volunteer activity in free software projects. In *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track* (Boston, MA, USA, 2004), pp. 93–102.

[41] MICHLMAYR, M. Quality improvement in volunteer free software projects: Exploring the impact of release management. In *Proceedings of the First International Conference on Open Source Systems* (Genova, Italy, July 2005), M. Scotto and G. Succi, Eds., pp. 309–10.

[42] MICHLMAYR, M. Software process maturity and the success of free software projects. In *Proceedings of the 7th Conference on Software Engineering, KKIO 2005* (Krakow, Poland, 2005). accepted.

[43] MICHLMAYR, M., AND HILL, B. M. Quality and the reliance on individuals in free software projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering* (Portland, OR, USA, 2003), pp. 105–9.

[44] MOCKUS, A., FIELDING, R. T., AND HERBSLEB, J. D. Two case studies of open source software development: Apache and Mozilla. Tech. rep., Avaya Labs Research, eBuilt, Bell Laboratories, Lucent Technologies, Basking Ridge, NJ, USA, 2002.

[45] MOEN, R. Source-code management / software-configuration management (SCM) / version-control systems (VCS) / revision-control systems (RCS) on Linux. http://linuxmafia.com/faq/Apps/scm.html. Last accessed: 31 August 2005.

[46] NICHOLS, D. M., AND TWIDALE, M. B. The usability of open source software. *First Monday 8*, 1 (2003). Last accessed: 31 August 2005.

[47] POOL, M. Bazaar-NG design. http://bazaar-ng.org/obsolete-docs/design.html, December 2004. Last accessed: 30 August 2005.

[48] POOL, M. Personal communication, September 2005.

[49] RAYMOND, E. S. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, October 1999.

[50] RAYMOND, E. S. The magic cauldron. In *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, October 1999, ch. 3.

[51] RITTEL, H., AND WEBBER, M. Dilemmas in a general theory of planning. *Policy Sciences 4* (1973), 155–69.

[52] ROBLES, G., GONZÁLEZ-BARAHONA, J. M., AND MICHLMAYR, M. Evolution of volunteer participation in libre software projects: Evidence from Debian. In *Proceedings of the First International Conference on Open Source Systems* (Genova, Italy, July 2005), M. Scotto and G. Succi, Eds., pp. 100–7.

[53] SCHULDEI, A. Small teams in Debian everywhere. Talk at Debconf5, Helsinki, Finnland, July 2005.

[54] SRIVASTAVA, M. Debian package management with GNU Arch. http://arch.debian.org/arch/private/srivasta/, August 2005. Last accessed: 1 September 2005.

[55] STÜRMER, M. Open source community building. Master's thesis, University of Bern, Bern, Switzerland, March 2005.

[56] THACKARA, J. *In the Bubble: Designing in a Complex World.* MIT Press, Cambridge, MA, USA, April 2005.

[57] THOMAS, M. Why free software usability tends to suck. http://mpt.phrasewise.com/discuss/msgReader$173, November 2002.

[58] TOWNS, A. Launchpad. http://azure.humbug.org.au/ aj/blog/2005/09/04#2005-09-04-launchpad-freeness, September 2005. Last accessed: 4 September 2005.

[59] VON HIPPEL, E. Lead users: A source of novel product concepts. *Management Science 32*, 7 (July 1986), 791–805.

[60] WHEELER, D. A. Comments on open source software / free software (OSS/FS) software configuration management (SCM) systems. http://www.dwheeler.com/essays/scm.html, May 2005. last visited: 30 August 2005.

[61] WILSON, G. V. Is the open-source community setting a bad example? *IEEE Software 16*, 1 (1999), 23–5.