Department of Computer Science and Information Systems

College of Informatics and Electronics

UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

A Delphi study of the influences on innovation adoption and process evolution in a large open-source project

The case of Debian

Martin F. Krafft, B.A.hons

Lero — the Irish Software Engineering Research Institute

<phd@martin-krafft.net> — http://phd.martin-krafft.net

A dissertation submitted for the degree of Doctor of Philosophy
at the University of Limerick, Ireland

Under the supervision of
Prof. Dr. Brian Fitzgerald and Dr. Kieran Conboy

6 Apr 2010

To infinity, and beyond!

Contents

List of tables

List of figures

Abstract

The Debian Project is possibly the largest FLOSS project, but its processes have not scaled in relation to its growth and complexity. Many aspects of daily tasks require the volunteer contributors to expend time on repetitive, minimally-integrated tasks, leaving less energy for progress and innovation. Techniques that support collaboration and foster efficiency exist, but they are not being adopted as readily as could be. As a cultural insider and active participant in the Debian Project, I sought to bridge this gap through the identification of the influences that shape the adoption behaviour of contributors to the project.

I conducted a Delphi study with a panel of 21 carefully selected Debian Contributors, and we found 24 influences, which are supported by numerous arguments from the participants' study. We established and defined the role and scope of each influence, explored its facets, and argued as to its effects. For instance, we found that (a) adopters require multiple perspectives of an innovation, as well as active marketing involving their peers, (b) perceptions of elegance and familiarity are closely related, and considerably affect acceptance of a new approach, (c) consensus depends on drivers and executive decisions in due time, and (d) standardisation needs to define interfaces, not processes. From the findings, I crystallised 19 implications for practice, which can help designers and diffusers optimise the adoption of tools and techniques in the Debian Project. I postulate these will help increase competition between ideas, boost efficiency and collaboration, and improve scalability of the project's processes.

This was the first application of the Delphi technique in a FLOSS context, to my knowledge, and the method fit spectacularly the mode of communication of the Debian Project, and the specific research objectives. My research demonstrates the approach's suitability to FLOSS research, and my second contribution is the meticulous documentation of all design considerations, and details of the entire process, from sampling the panel to processing the data. This work should facilitate the use of the Delphi method for future research, as well as day-to-day consensus-finding in FLOSS projects.

In the spirit of the Debian Project, and to foster further research into the adoption behaviour of Debian Contributors, I obtained permission from all panelists to publish all data generated during the study under a free licence. The availability of these data also reinforces the validity and reliability of my research.

Originality note

I, Martin F. Krafft, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I have duly indicated the origin.

Martin F. Krafft

Version history

I expect this work to evolve after its first publication. Therefore, I encourage readers to check for new versions at `http://phd.martin-krafft.net`.

If you find a mistake, or would like to make a suggestion, please write in to <phd@martin-krafft.net>.

The following documents the evolution of the document:

2010.04.06 — Submitted to University Examination Board.

2010.03.04 — Revision with minor changes submitted to examiners.

2009.10.01 — Submitted to examiners.

Copyright & Licence

[1] `http://creativecommons.org/licenses/by-sa/3.0/de/deed.en` [12 Sep 2009]

Acknowledgements

Throughout the writing of this thesis, I was looking forward to putting together this section, for two reasons: first, it would be the last, finishing touch on a work that kept me busy four years of my life. And second, because I would finally get to strive to find the appropriate words to thank all those who have their role in the completion of this dissertation. I like finding appropriate words.

I am deeply indebted to Brian Fitzgerald, my research supervisor, for his guidance and respect, his patience and impetus, and for all the advice he had to offer from the moment I presented my first, unstructured idea, to the final read of this work. Brian, you have recently been elected Vice President Research of the University of Limerick — a position made for you — and yet, you managed to disprove my fears that this would mark the end of your dedication and time available for me. *Míle buíochas.*

My second research supervisor, Kieran Conboy, has provided invaluable insights of the Delphi method, and pushed me onto the right track on multiple occasions. Kieran, apart from phone conferences, you even typed elaborate replies on your Blackberry (!) so that I did not have to wait too long for your pragmatic suggestions. I greatly appreciate your help, *go raibh maith agat.*

I would like to thank Lero — the Irish Software Engineering Research Institute, who have sponsored all my travels to Limerick, as well as my tuition fees, and their staff and students, who let me be one of them every time I came over.

My research would not have gone anywhere without the magnificent contributions from the participants of my study. Russ Allbery, Luca Capello, Pierre Habouzit, Raphaël Hertzog, Joey Hess, Damyan Ivanov, Guillem Jover, Scott Kitterman, Steve Langasek, Loïc Minier, Christian Perrier, Charles Plessy, Jonas Smedegaard, Andreas Tille, Niko Tyni, Colin Watson, James Westby, Neil Williams, Gunnar Wolf, Stefano Zacchiroli, Enrico Zini: it was a great experience working with you. Thank you for your insights, and for your patience and commitment.

I appreciate the generous sponsorship I received from Nokia,[2] Hewlett-Packard,[3] Tuxbrain,[4] and Thecus.[5] It was through your help that I could compensate the participants.

[2] `http://nokia.com/` — thank you, Ari Jaaksi and Quim Gil.
[3] `http://hp.com/` — thank you, Bob Gobeille and Martin Michlmayr.
[4] `http://tuxbrain.com/` — thank you, David Reyes Samblas Martinez and Victor Remolina.
[5] `http://thecus.com/` — thank you, Charlene Chang and Ivy Chiu

Numerous people have provided inestimable advice and feedback at various times. Enrico Zini and Stefano Zacchiroli always had answers when I was stuck on method or Debian concerns, and they proofread all of the correspondence with the Delphi panel, in addition to participating. *Grazie mille.*

E. Gabriella Coleman helped me with valuable suggestions, particularly in the early stages of my work, and she also proofread whenever she could. Biella, thank you, and I am sorry that I could not reciprocate when you turned your impressive dissertation into a book.

Gian-Marco Baschera never hesitated to challenge my assumptions, test my theories, read and comment on what I put in front of him, or come up with convincing arguments for why I should not work a given evening. *Merci vielmal.*

My brother Thomas was always available to receive and respond to my thoughts, and he helped me understand the true meaning of epistemology and the other "metatheoretical assumptions". I would not be able to call myself a scientist without our conversations. *Danke.*

Mel Gorman and Daren Nestor let me share their house(s) during my stays in Limerick, and readily engaged in discussions over the sense and non-sense of what we were doing, or provided irresistible distraction; I was very lucky to meet you chaps, and the rest of the gang. *Go raibh maith agaibh.*

I am much obliged to my examiners, Drs. Barbara Russo and Fergal McGrath, whose challenging questions and excellent suggestions helped me improve my work.

Thank you to everyone who makes Debian what it is — a powerful operating system, and a close-knit community of great people. I especially appreciate the patience of the folks in the `#debian-devel` channel, who put up with my off-topic blather for so long. I am also indebted to the makers of the countless FLOSS products I employed to author this work.[6]

I am extremely grateful to my parents who supported me in any way they could throughout all these years (thanks!), and who remembered with astounding precision in regularity to ask me when I would finally be done; I never lost sight of my goal.

And last, but not least, I owe it to my partner Penny Leach: you disliked when I had to work, but you were always there to help, contribute, discuss, challenge, and support me through hard times. You knew me well not to point out my procrastination skills, and you did not lose your patience with me. I could not have asked for more.

Zürich, February 2010

[6] I wrote this thesis on Debian GNU/Linux, using the Vim editor, and had TeXlive (with KOMA-Script) typeset it. I organised my bibliography with JabRef, and created all graphs with Inkscape. Mutt faithfully managed the vast amount of e-mails generated throughout this study, and Git tracked all the data for me. My scripts used Python, Perl, or zsh.

Chapter 1

Introduction

1.1. Motivation

The Debian Project is a highly successful project, run entirely by over 2,500 volunteers,[1] and possibly the largest FLOSS[2] project [Amor-Iglesias et al., 2005b]. It produces several computer operating systems, the most popular of which (Debian GNU/Linux) provides over 27,500 software packages for twelve different hardware architectures. Furthermore, the Debian System is the basis for around 100 derivative distributions, such as the popular Ubuntu distribution, and large-scale, localised installations, including Guadalinex and Limux.

I have been involved with the Debian Project for over a decade. At first, it was only a hobby; then teaching Debian and planning Debian deployments became my main source of income. Along the way, my fascination with the project kept building up and I documented what made Debian so special in a book [Krafft, 2005]. Today, I am an active contributor, and more than ever interested in helping Debian continue its line of success.

Growth and scalability After 16 years in existence, it is evident that some of the project's processes did not scale to meet the tremendous growth we have seen over the years. Endeavours, such as library transitions, currently require dozens of contributors to work hand in hand,

[1] Please refer to appendix A.4 for sources, more numbers and facts.

[2] The acronym FLOSS is an inclusive way to refer to software that is (a) Free, as in costless, (b) Libre, as in unrestricted, (c) Open-Source, meaning the sources are available. The acronym grew out of FOSS, because of the amiguous meaning of "free", which in turn was used to distinguish truly free, open-source software from commercial open-source software, whose use might be restricted despite the availability of the source code.

and often stall because of bottlenecks or lack of coordination. Day-to-day tasks consist of tedious and thus error-prone tasks, which are anything but integrated: keeping track of patches, triaging bugs, follow policy changes, and work with upstream and derivatives — to name just a few.

Looking at the way these processes are currently handled, it seems strange that contributors of a system as technically sound and universally applicable as Debian are still doing by hand what the computer should be doing for them. Tasks like the aforementioned could be streamlined and optimised to avoid redundancy and points of failure due to brittle integration. For instance, a bug report should be marked as done only when the patch fixing the issue has been signed off and appears in the repository. Furthermore, the patch should be linked to the bug report in a non-redundant and standardised way.

The Debian Project and version control In recent years, the use of version control systems (VCSs) in the Debian Project has increased. VCSs come in two flavours: centralised and distributed. While centralised VCSs used to dominate the field, distributed version control systems (DVCSs) were mostly unheard of. I had been exposed to several of them,[3] but the only one I had used extensively was GNU arch. GNU arch was more of a pain to use than anything, but it was the best tool I could find, and I was adamantly convinced of the distributed approach. I stuck with it for too long.

In 2005, several new DVCSs were born, including Mercurial,[4] Bazaar,[5] and Git.[6] While Bazaar was specifically designed to be user-friendly and the developers made every effort to facilitate the switch for users coming from classic VCSs (such as Subversion), Git gained an early reputation of being unusable and complex. Yet, in the Debian Project, it seemed that Git usage grew fastest of the three.[7]

This aroused my curiousity: what is it about these tools that makes their adoption rates different? As soon as I had formulated the question, similar cases popped up all over the project: debhelper *vs.* CBDS, lintian *vs.* linda, and quilt *vs.* dpatch, to name but a few. Moreover, looking beyond the Debian Project, the question was seemingly ubiquitous across the

[3] darcs, Monotone, and BitKeeper
[4] `http://mercurial.selenic.com/`
[5] `http://bazaar-vcs.org`
[6] `http://git-scm.com/`
[7] According to `http://popcon.debian.org` [3 Sep 2009], Git usage grew about four times as fast as Bazaar, and about twice as fast as Mercurial use, although these figures are not really representative of actual use in the Debian Project.

entire FLOSS domain, with the exception of projects that had very clear top-down leadership.

Voluntary adoption The Debian Project is, however, a volunteer project, and adoption rates depend entirely on individual decisions, as well as dependencies that result from cooperation. Improved tools and techniques are necessary to increase the efficiency of the project contributors, who are not paid to work on Debian, and can hence only allocate parts of the limited time their day jobs leaves.

Some such tools and techniques already exist, but they are not being adopted as readily as they should be. No guidelines exist on how to help their project-wide adoption, and everyone is on their own trying to communicate better approaches to the majority, without understanding the behaviour of potential adopters. As a result, those ideas only slowly rise to become competitors with existing methods, people will have little reason to change, and progress grinds to a halt.

I wanted to find out what considerations Debian Contributors (DCs) make, which extrinsic and intrinsic properties of ideas, concepts, techniques, and tools affected these considerations, and how, and what other influences shape their decisions for or against change. I postulate that knowledge of the influences to adoption decisions among Debian contributors would help everyone figure out what is necessary to enter competing ideas into the field. With healthy competition, only the best tools and techniques will emerge as winners. And because competition drives progress, the project should be able to scale better as a result.

A Delphi approach The more I investigated, the more my focus shifted from action to analysis, and I eventually abandonded the participatory action research approach I had devised as a means to incorporate DVCS with Debian packaging. In searching for a new approach, I came across Surowiecki [2004] and discovered the Delphi technique (see chapter 5), a tool set for group communication which lets a diverse panel of knowledgeable representatives iteratively approach questions, while anonymity between panellists helps them maintain focus on content of the discussion rather than personal preconceptions about the peers. This approach, which has often been used in forecasting studies, seemed an ideal means to explore the previously unfamiliar terrain of adoption behaviour in the Debian Project.

The Delphi method is not a cookbook approach, but a collection of tools and processes which can be combined to fit a wide variety of use-cases. As I developed the design of the Delphi study I set out to conduct, I grew more convinced that the Delphi technique is a very suitable approach

to research in the Debian Project, and possibly FLOSS in general: it fits the communication modes of DCs spectacularly, because it imposes a structure on communication that is painfully absent in day-to-day activity, and underlines the meritocratic spirit of Debian by focusing on contents of messages, rather than on who emitted them.

Despite the suitability of the method for such research, I could not find existing applications of the Delphi approach in the FLOSS context, or adoption research that was in any way related. Motivated by the prospect to facilitate and further the use of the Delphi approach in FLOSS research, I meticulously documented my design as well as the entire approach. Future researchers wishing to employ the Delphi technique for their own endeavours will find in this work a wealth of information to help them with design and conduct of their approach.

Outcome The study I conducted involved 21 carefully selected contributors to the Debian Project. In three iterations of Delphi discussions, we identified 24 different influences to adoption decisions, and collected numerous, seminal quotations to explain the facets of each of those influences. I presented these influences with the help of a stage-model derived from existing organisational and individual adoption stage models, and provided meaningful labels for each of them, so that they can be easily referred to with a common terminology.

From the data, I also crystallised 19 implications for practice. These implications include obvious and surprising results, but constitute an instructive set of guidelines which will help contributors communicate new ideas and tools, and allow the Debian Project to scale.

1.2. Research question and objectives

The problem on which my research focuses is a problem commonly found in volunteer communities: lack of authoritarian structures make it impossible to *push* change. While the Debian Project, its members, the collaboration between them, and the approaches used are in continuous flux, there is no obvious means to drive change in a given direction, because ultimately, the decision to change lies with each individual, and project-wide change thus depends on the entire community.

The research problem is not specific to Debian, but exists in the context of other volunteer communities, especially FLOSS projects, unless these projects have grown organisational structures that allow for executive decision-making. It is not my goal to assess the degree to which

executive decision-making can benefit a volunteer project — I think the answer to that is "yes, within bounds". Instead, I seek to find out how to cause change in non-authoritarian ways.

While various aspects of the Debian Project constantly evolve, and the ability to influence this evolution could be beneficial in the context of each of these aspects, the Debian Project is foremost a technical project concerned with the production of an operating system. Therefore, I chose to focus only on change in the chief technical way this goal is met: software packaging.

I claim that the packaging workflows used by DCs are suboptimal: the common methods are minimally integrated at best, and package maintainers lose time and energy on repetitive, error-prone tasks. Alongside individual frustration and unnecessarily high impetus to spend more than the absolute minimum time on package maintenance, it slows project progress and hinders the investigation of novel approaches.

On the other side of the spectrum one finds improved tools, featuring better integration, collaboration facilities, and greater degrees of automation. In the 16 years of age of the Debian Project's, only a few such tools have been adopted at a wide scale, while many have never reached significant levels of use. In recent times, the project has seen strong trends towards techniques supporting distributed development, which promise solutions to many of the (centralised) deadlocks and bottlenecks in the project, and also present a welcome opportunity to rethink processes. Yet, project-wide acceptance has been slow, for reasons which are not always obvious.

As previously stated, the decision to change rests with the individual, and hence project-wide change is a community effort. Change cannot be dictated, but change can be driven through persuasion and facilitation. This is in contrast to an authoritarian drive for change, because change through persuasion and facilitation remains a cooperative endeavour between the change driver and the individual decision-makers.

And yet, times and times again, a given tool or technique grows quickly to reform processes without any drivers. At first, it might seem that said tool or technique is outstanding in that it spreads without help. Upon closer inspection, however, the adopters and their social system play non-trivial roles as well.

Research question The logical next step is then to ask what attributes and traits of innovation, adopter, and community determine the adoption rate. However, I found this perspective

too abstract, at least in the context of voluntary adoption, where the adoption rate depends entirely on the individuals making decisions. Therefore, I focused on those decisions and sought the influences that shape them. This feels more tangible in a FLOSS-context, where there is never a clear distinction between producers and consumers of tools and techniques, but where every producer is essentially also a consumer.

The question I seek to answer through my research is hence:

> What are the influences that shape DCs' adoptions of innovations?

I hypothesise that knowledge of these influences would allow tool authors and technique designers to cater better to the needs of their future users, thereby increasing acceptance of their output. Furthermore, those concerned with project-wide optimisation and integration of processes will be able to put their finger on shortcomings of tools or techniques they are trying to diffuse, and to address these in an effort to drive change.

While I strive to identify these influences in the Debian Project, I expect few to be specific to the project. The general concept of influences to adoption decisions will be applicable to other voluntary settings, and FLOSS in particular, and I anticipate most of the influences to be relevant.

Objectives A researchers seeks to answer questions through the definition and accomplishment of objectives. In response to my aforementioned research question, I aspire the following objectives:

1. to determine the salient influences to DCs' tool/technique adoption or rejection decisions;
2. to propose a terminology of labels for these influences, and present them in a meaningful, accessible way;
3. to crystallise a number of implications for practice from these influences, to help increase the rate of diffusion of improved tools and techniques in the Debian Project, to foster competition and progress, and to enable the project to scale better with its growth.

I also plan to achieve the following sub-objectives:

1. to provide stepping stones for future work, by making all data available under a Free licence, speculating research ideas and identifying niches for further work;

2. to further the use of the Delphi method in the FLOSS environment, by highlighting its applicability in the FLOSS context, meticulously documenting the design of my approach, and analysing the performance.

I postpone further discussion of these objectives to section 4.2.

1.3. Outline of the thesis

This thesis extends across eight chapters, of which the introduction you are currently reading is one. The remainder of this document is partitioned into three parts, followed by the back matter.

Where appropriate, I start each chapter with a review of the relevant prior content, rather than summarising chapters at the end of each. This is in pursuit of my goal to keep chapters self-contained, such that chapters 5 and onwards, which are themselves part of my overall contribution, can be used by themselves.

Part I provides the necessary background information for the research, reviews the literature, and defines the terminology used throughout the document. I have split it into two chapters, because this work unites two disjunct fields: chapter 2 covers all aspects related to Debian, while the field of diffusion research is subject of chapter 3.

Chapter 2 introduces Debian, gives a glimpse of the characteristics of the community involved in the project, and describes on of the primary task areas of the project to help the reader understand the work and cooperation that happens within the project. Before I present previous research on Debian, I identify the need and availability for improved processes in the project in section 2.4, and contrast it with the non-authoritarian nature of the project.

In **chapter 3**, I start with a short account of the history of the field of diffusion research, and cut through the somewhat convoluted terminology to ensure that the most important terms used throughout this paper are clearly defined. An overview of frameworks, which have been commonly used to structure the characteristic attributes of diffusions, follows a short description of necessity and role of a framework, but since none of the frameworks I found were suitable for my research, I argue in section 3.2.3 for the best course of action to be to let the framework emerge from the data.

Entering **part II**, the reader knows of the dissonance between the need for improved tools and processes in the Debian Project, and the slow adoption of existing solutions. This dissonance is core to the field of diffusion research, and with the intention to apply diffusion theories in a novel context — a voluntary social system with peer-to-peer communication and high levels of interdependencies — I expand my objectives and provide the philosophical foundations of my research in **chapter 4**.

Chapter 5 is devoted to introduce the reader to the Delphi method, and to argue its suitability in the context of my research. The chapter is very detailed as part of my objective to facilitate future use of the method in the FLOSS context (see section 4.3.2).

The Delphi method is a group communication approach involving moderated discussions, rather than one-on-one interviews, and **chapter 6** details my implementation of the approach in the context of the my research. Section 6.1 explains how I chose the participants of the discussion panel, and in section 6.2 the entire process is described in-depth. Again, the level of detail is motivated by my objective to facilitate future use of the Delphi method, because the existing literature seldom explained the actual process.

The analysis of my research endeavour follows in **part III**, starting with the presentation of the results in **chapter 7**. With the intention to let this chapter stand on its own — the results have implications for DCs, who may not want the surrounding details — the structure used to outline the results is described in section 7.1, as opposed to the background chapter (see section 3.2.3). I then use this structure to present the results in section 7.2.

My contributions, and the implications of my research are to be found in **chapter 8**. There, I also identify problems and shortcomings of this first application of the Delphi method in a FLOSS context, reflect upon the research and design decisions I had made, and speculate some future research questions that emerged out of the research.

The collective bibliography can be found on page 287. The document ends with four appendices, which provide more background information on Debian (**appendix A**), diffusion research (**appendix B**), and the research approach (**appendix C**). In the last appendix, I collected all acronyms and abbreviations for reference.

Part 1.

Background and literature review

Chapter 2

Debian

Debian is one of the largest FLOSS projects [Amor-Iglesias et al., 2005b]. As such, it has been the subject of a considerable number of studies. In addition to scholarly works, vast volumes of text have been written on the subject in on-line publications, weblogs, and last but not least, publicly-archived mailing lists. The purpose of this chapter is to introduce Debian and one of its core workflows, and present an overview of existing literature. You can find treatment of the most important Debian terminology in appendix A.1.

2.1. Introducing Debian

Debian is primarily the name of a project, christened after its founder Ian Murdock, who was married to Debra when he created the project, thus Deb-Ian (/'debiən/). The Debian Project is a globally-spread, non-commercial "association of individuals who have made common cause to create a free operating system" [Debian Project, 2006]. The project came to life 16 years ago to publish Debian GNU/Linux, the second oldest GNU/Linux distribution after Slackware Linux.[1]

The Debian Project is possibly the largest and most complex FLOSS project [Amor-Iglesias et al., 2005b, Lameter, 2002]: 1002 people from 55 countries are registered as official developers.[2] In addition, an uncountable number of sponsored maintainers, translators, testers, and, last but

[1] The GNU/Linux distribution time-line: `http://futurist.se/gldt/` [26 Sep 2007]
[2] Please refer to appendix A.4 for sources, more numbers and facts.

not least, users provide their input and contributions around the clock. Steinlin [2009] counted about 2,500 regular contributors to the Debian Project.

"Debian" is also used synonymously to refer to the concepts shared by a family of operating systems produced by the Debian Project. Debian GNU/Linux is the most prominent of these operating systems, and the Debian Project's main product: a distribution, which integrates the Linux kernel, GNU user-space,[3] and Debian system administration concepts and techniques.

Debian GNU/Linux is a *free* computer operating system. "Free" can refer to both, costlessness ("free as in beer") and unlimited use ("free as in freedom").[4] Debian takes an extraordinary and somewhat radical approach to freeness and freedom in that the project promises "that the Debian system and all its components will be free [(in both senses of the word), and that the project] will never make the system require the use of a non-free component." This promise is formalised in one of the project's fundamental documents, the Debian Free Software Guidelines (DFSG) (see appendix A.7), a part of Debian's Social Contract (see appendix A.6). As I shall mention in section 2.2.4, this adherence to freeness and freedom is a fundamental motivation for the project's members.

To date, Debian GNU/Linux is made up of over 27,500 software packages, most of which are available for all of the twelve supported architectures. Amor-Iglesias et al. [2005a] find "Debian [to be] one of the largest software systems in the world, probably the largest."

I have published a book on Debian, which documents these concepts and techniques. The book also contains chapters on the project history and culture, development techniques in use, and traits of its community [Krafft, 2005, chs. 1, 2, 4, 5, 9, 10]. In this work, I researched adoption behaviour among the group of contributors, which my book does not cover (yet). Nonetheless, it serves as a comprehensive source of background information on the topic and portions of the following chapter are based on its contents.

2.2. The Debian community

To understand the influences to adoption behaviour in the Debian Project, it helps to know the actors, particularly because of the ideological orientation of the project. This section gives a

[3] GNU (GNU is not Unix) is a FLOSS project that pre-dates the Debian Project by a decade and strives to provide free, open-source implementations of the basic user-space applications, which hold the system together.

[4] To emphasise adherence to both principles, the word is often capitalised: "Free". The Debian Project uses the lower-case version, even though it attributes both meanings to the word.

brief overview of the community, and attempts to explain what motivates contributors to work together on Debian, why Debian Contributors (DCs) might at times chose quality over efficiency, that getting work done is not always the top priority, and the role played by the different communication media in the project.

2.2.1. Project members and contributors

Traditionally, the Debian Project distinguished only between developers and non-developers, all of whom are considered users. Over the years, as the project grew and more people contributed in an ever increasing variety of ways, additional roles came into existence [Coleman, 2005b, Michlmayr, 2004, Wallach et al., 2005]. Please refer to appendix A.3 for details on the evolution of membership in the Debian Project and the history of the various roles.

Today, the community is still partitioned mainly into developers and non-developers, and only developers are official project members. The official developers have a number of privileges in the project, which enable them to perform a wide variety of tasks. However, more and more critical contributions come from non-developers, who dedicate their time as package maintainers, translators, designers, or who assist others and offer support. Those contributors are generally not project members, because they often do not require the full set of privileges that come with membership, and because the membership process is chiefly technical, elaborate, and time-consuming (see appendix A.3.5).

Nevertheless, precluding such contributors from voting (one of the core privileges of project members) deprives them of the ability to influence the direction of the project, and does not pay proper tribute to their contribution. The situation is unacceptable to many people involved with the Debian Project,[5] but efforts to put a more flexible system in place are progressing slowly (see appendices A.3.6 and A.3.7).

In the Debian Project, not only Debian Developers (DDs) adopt new tools and techniques, but every contributor. Moreover, the clear trend towards more intensive collaboration within the project, across teams, and even across distributions yields higher degrees of interdependencies between individuals. One contributor's decision for or against a tool may have significant effect on another contributor, and on this level, it matters little who is an official project member and who is not (see appendix A.3.7).

[5] *e.g.* `http://www.debian.org/vote/2008/vote_002` [22 Sep 2009]

Therefore I did not distinguish between official and inofficial project contributors throughout my research, but use the term "DC" throughout to refer to every contributor who makes adoption decisions, and/or may be affected by adoption decisions made by others. In the well-known onion model [*e.g.* Crowston and Howison, 2003], this would be the set of "co-developers".

By extension, participants of the study detailed in part II were not selected according to their project membership status, because any attempt to analyse adoption behaviour in the Debian Project must take into account all players.

2.2.2. Debian governance

The organisational structure of the Debian Project is briefly described in appendix A.3.1. Even though the project members elect a leader, who appoints officers and delegates, and those bodies are empowered to make decisions within the powers invested in their roles by the Debian Constitution, the Constitution also places the project member collective at the highest position. This means that any decision made by an official delegate can be overturned by the developer collective. This ensures that no delegate can abuse his/her powers against the rest of the project (see appendix A.3.4). At the same time, the absence of executive power causes a certain loss of focus in the project, which is exemplified by long discussions in the project, as well as slow overall progress.

Furthermore, the Constitution explicitly exempts project members from all obligation. By extension, this means that no one can be told what to do, or how to go about a task [Robles et al., 2005b]. Each DC has his/her own set of motivations to work on Debian, which have been studied separately [*e.g.* Feller and Fitzgerald, 2002, Hertel et al., 2003, Raymond, 1999, Späth et al., 2008, Subramanyam and Xia, 2008, Weber, 2004b]. Of particular interest is the work by Lakhani and Wolf [2005], who identified intrinsic factors relating to creativity as core motivational components of FLOSS involvement. Yet, Coleman [2005b, p. 319] notes that too much attention has been devoted to the study of motivation, probably because of "the ways in which the norms and convictions of the researchers, [and] deeply embedded cultural ideologies surrounding property and creativity" have been challenged by FLOSS, but that the literature does not "account for the plasticity of human motivations and ethical perceptions." I will return to this point in section 2.2.4.

2.2.3. The role of volunteers

The Debian Project is an organisation driven entirely by volunteers. The project does not pay any of its developers,[6] nor does it let its sponsors or its legal entity have any influence in the project's technical interests.

Robles et al. [2005b] offer a comprehensive definition of what it means to be a volunteer, which I use in this study: volunteers are those who work "in their free time not profiting economically in a direct way from their effort." A number of project participants are also specifically instructed by their employer to contribute to Debian, and some employers permit their staff to devote some of their time to Debian during work hours. Nevertheless, the majority of contribution is performed by volunteers in their spare time [*ibid.*].

The degree to which one's paid work has an effect on the involvement with the project has not been quantified, to my knowledge. For my study, I did not consider this background and offered incentives to participants to allow them to participate in their free time (see section 5.4.9.2).

The voluntary nature of the project is significant in the context of this research, because the decision how to perform one's work for the project is almost entirely within the hands of each individual. Network effects can restrict these choices, but teams tend to make their decisions so as not to exclude individuals from cooperation.

Even though authoritarian instances mandating certain processes exist in the form of foundation documents (*e.g.* the Debian Policy, see appendix A.1.5), those only document and mandate what is already being done, rather than define how new tasks are to be done; no authority exists in the project that could tell the contributors what to do, or how to do it, which, assuming a given tool or technique is suitable to a task, makes the decision to adopt entirely voluntary.

[6] The Dunc-Tank experiment (`http://www.dunc-tank.org/` [7 Nov 2007]) is one notable exception, but the experiment is said to have failed.

2.2.4. Stereotypical traits of Debian developers

Coleman [2005b], by way of three examples, illustrates how Levy's seven hacker ethics[7] [Levy, 1984] have been embodied and furthered by the Debian community, and highlights the importance of acknowledging how these ethics evolve in new contexts, such as the Debian Project. For instance, Levy's "true hackers" did not have a concept of legality, which is a strong point of focus in the Debian Project (*cf.* section 2.2.5). Coleman [2005b, p. 319f] observes that researchers who defer to Levy ignore how "hacker valuations, motivations, and commitments are transformed by the lived experiences that unfold in FLOSS projects and institutions that are mediated through project charters and organizational procedures."

Coleman offers an alternative perspective: "developers commit themselves to an ethical vision *through*, rather than prior, to their participation in a F/OSS project" [*ibid.*, her emphasis]. This is a powerful perspective, because it goes beyond character traits that are allegedly held by all FLOSS participants, and enables to shift focus from individuals to their project involvement, and interaction with each other.

Commitment is "the state of being bound emotionally or intellectually to some course of action" [Klein, 1994, p. 179]. Apart from the number of traits shared by project members, which are subject of this section, and which unite contributors in their commitment to the project, a significant degree of organisational loyalty may also be found among DCs. This loyalty may be rooted in the firm belief in the value of everyone's individual contribution, or be in turn an instance of a network effect: contributors are also users, and not identifying with Debian as an organisation would have to suggest switching to a different operating system. By consuming their own produce, DCs fulfill nearly entirely the three key conditions necessary for jobs to generate high levels of internal work motivation identified by Hackman and Oldham [1980]: "meaningfulness of work, responsibility for outcomes, and knowledge of results."

Intra-project peer relationships are another source of commitment and motivation [*cf.* Krackhardt, 1994, p. 218ff]. As is commonly the case in communities, tasks crop up over time, some requiring short-term involvement, and some of a continuous nature. Because Debian community members are at the same time consumers of their collaborative produce, a sense of duty or obligation arises in everyone to varying degrees to approach a task. Perhaps as a means to

[7] (1) Access to computers — and anything which might teach you something about the way the world works — should be unlimited and total. (2) Always yield to the Hands-On Imperative. (3) All information should be free. (4) Mistrust authority — promote decentralization. (5) Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position. (6) You can create art and beauty on a computer. (7) Computers can change your life for the better.

justify one's commitment of resources to a given task, DCs commonly take pride in their obligations, and exercise diligence in their jobs, possibly to increase their merit in terms of quality output and peer respect. It is interesting to note that this is unlike closed environments, where obligation arises less out of personal choices than out of contractual duties and commands. In fact, volunteers — at least in Debian — have been known to overcommit themselves, which is a curious phenomenon in and of itself.

Despite long-term obligation, the degree of voluntary involvement creates a fluidity in tasks, while the diversity among project members ensures that (nearly) all activities are covered. Cases of trading jobs (or favours) are not unheard of, and in many instances, people strive to increase their own efficiency if the effects include improved working conditions for their peers [Klein, 1994, p. 181]. Instead of altruism, this is another motivational factor in that it increases one's standing with one's peers. I will investigate Debian as a meritocracy in section 2.2.8.1.

Coleman perspective above is not bound to an ethical vision, and the previous two paragraphs highlighted two other aspects of commitment. In my research, I concentrate on productivity and efficiency, and there is no reason to assume that a vision of productivity and efficiency could not similarly be affirmed through participation in the Debian Project. I discuss some other instances in which motivation through interaction is important in my book [Krafft, 2005, p. 50ff].

The semi-obsessive hacker traits, as discussed *e.g.* by Turkle [1984], are still relevant in the Debian Project, because many of the DCs are what Levy would call hackers. The following is a very brief discussion of additional traits relevant to this research. Most are a function of involvement rather than a feature of personality.

2.2.4.1. Quality orientation

In my book, I described Debian as "academically-inspired applied functionalism" [Krafft, 2005, p. 31] suggesting that DCs approach problems patiently and academically in search of solid, long-term solutions. Yet, the solutions have emerged out of the practical needs of Debian users. As a consequence, Debian's solutions are far more powerful and robust than needed in most situations, but these solutions can be put to use in standard and complex scenarios alike. This is one of the reasons why we call Debian the "universal operating system."

Quality is a core value of the Debian Project and its members. Among the most prominent symptoms of the rigorous quality orientation of the project is the resistance to market pressure, and

in particular the adherence to the principle to release "when it's done", rather than to subscribe to a regular release cycle [*cf.* Halloran and Scherlis, 2002].

The quality orientation of the project is one of the motivations for contributors to invest time into Debian. High levels of peer review ensure code of outstanding quality [Michlmayr, 2005b], which is a source of pride, but also of direct benefit to the same contributors using the system themselves. Given the wide basis that is an operating system — all programmes, and thus all computer use builds and depends on it — the contribution cycle quickly becomes self-sustaining, when administrators discover the benefits of integrating robust and high-quality solutions with Debian, rather than maintaining local solutions [Krafft, 2005, p. 31].

The preference for solid solutions in combination with absence of market pressure allows the project to advance conservatively, and maintain the quality orientation. This also means that adoption behaviour among project members is likely to be more conservative, and I expect robustness and quality to be more salient than in other contexts.

2.2.4.2. Conservative adoption

There seems to be no reason to assume that the distribution across adopter categories (see the later discussion in section 3.1.2) in the Debian Project would be any different than the spread identified by Rogers [2003, p. 281], which would suggest that only a very few DCs innovate, while the majority tends to wait for the early bumps and holes to be fixed before adopting. Even though adoption tends to be slow in the absence of an authoritarian drive to change [*cf.* Rogers, 2003, p. 29f], levels of awareness and interest in a new technology tends to be high much earlier among members of the Debian Project, due to the extensive use of communication media (see section 2.2.7), and possibly also as a function of age and voluntary involvement, as expressed by a Russ Allbery during the study I conducted for this research[8] (see section 6.2):

> I think the young average age in the project is one reason why Debian has such a high percentage of early adopters compared to a lot of other projects I follow and workplace environments I'm used to — not so much directly the matter of age, but the fact that a lot of Debian contributors are focused on experimenting rather than working to accomplish something in particular, may be able to experiment for

[8] <87wsep9a04.fsf@windlord.stanford.edu>

> Debian as part of their coursework, and also don't have years of familiarity with specific tools to overcome.

Combining rapid communication with the quality orientation identified in section 2.2.4.1, I expect adopters in the Debian Project to be aware of new ideas earlier than may be the case in other social systems. DCs seem to absorb information and consider improving the efficiency of their workflows, or follow the lead of others,[9] but without losing sight of the work that should get done in the limited amount of time available. Instead, many prefer to postpone adoption until the early adopters have polished the solutions:[10]

> I find your approach very interesting but probably not worth the trouble at the moment. Once the early adopters such as yourself have come up with a streamlined workflow for keeping Debian modifications in branches, I'll reconsider.

2.2.4.3. Getting things done, or shaving yaks

Given a chunk of time, a DC is free to use it in any way s/he pleases: invest it towards completion of a contribution to the project, or spend it improving one's arsenal of tools, figuring out improved ways of achieving common tasks, or investigating new tools — to name but a few pastime activities. The aforementioned activities lie on opposite ends of the spectrum between "getting things done" and what has become known as "yak shaving":[11]

```
< madduck> is there a term for wasting time playing with and improving
           the tools one uses in accomplishing a task, instead of
           accomplishing the task?
< a_developer> I know exactly what you mean. fiddle with the tool for
               4 hours instead of just doing the 4-minute task
< another_dd> most of us do
< yet_another_dd> it's yak shaving
```

— #debian-devel, 7 Nov 2007

Arguably, yak shaving is an extreme, and *working on* one's tools rather than *putting them to use* may well be a worthwhile time investment, especially in the light of the quality orientation commonly found in the project (see section 2.2.4.1). In fact, as the above quote suggests, everyone succumbs to yak shaving, whether getting things done, improving tools, or envisioning new ways

[9] *cf.* http://lists.alioth.debian.org/pipermail/vcs-pkg-discuss/2007-October/000039.html [25 Oct 2007]

[10] http://lists.alioth.debian.org/pipermail/vcs-pkg-discuss/2007-October/000032.html [10 Oct 2007]

[11] A true gem of the Internet: http://projects.csail.mit.edu/gsb/old-archive/gsb-archive/gsb2000-02-11.html [7 Nov 2007]

to work.[12] Nevertheless, it seems that DCs are increasingly more aware and cautious of sinking time into anything other than getting their Debian work done.

A better spectrum than one between getting things done and yak shaving is provided by Moore [1991] in the context of the aforementioned adopter categories (see section 2.2.4.2): "visionaries" are those who devise new tools, or are quick to adopt them, whereas the "pragmatists" usually wait until the majority picks up a new idea.

The Debian Project has around 2,500 active contributors (see section 2.1), as of August 2009, 110 people were signed up to the `vcs-pkg-discuss` mailing list, which explores how to improve package maintenance with version control systems (VCSs), and not more than a handful actively participate in discussions or write articles on the issue. This comparison indicates that the number of workers and pragmatists outnumber the visionaries by an order of magnitude. The sheer size and quality of Debian GNU/Linux furthermore suggests that the project has been been busy creating solutions based on a modest level of conceptualisation, rather than engage only in visionary activity.

The distinction between visionaries and pragmatists is of relevance in the context of this research, because visionaries are the drivers of new tools and techniques, while the pragmatists belong to the later majority when it comes to adoption behaviour. In section 6.1, I asked participants of the study to self-qualify not according to their actual position on the spectrum between those two, but according to their interest in workflow design and improvement.

Finally, yak shaving awareness is also of relevance: given limited amount of times, being aware of how one's time is spent can result in initially unfavourable views about new tools or techniques, which would consume time to adopt.

2.2.4.4. Absorptive capacity

An important concept in this context is absorptive capacity — the ability to recognise, assimilate, and apply external information to internal processes [Cohen and Levinthal, 1990, Zahra and George, 2002]. The Debian Project's absorptive capacity depends on the absorptive capacity of its members, which is limited mainly by the fact that contributors want to contribute to the project (and FLOSS at large), instead of learning new methods [*cf.* Stürmer, 2005]. While

[12] I suspect that much of Debian's core is affected by the issue in one way or another, and if it's only the distractions on IRC channels, which are central to Debian development, but also very "chatty". In the study I conducted, a participant considered yak shaving to be a common trait among enthusiasts (see section 7.2.3.5).

in the business domain, where the concept of absorptive capacity originated, a firm's innovativeness is vital and absorptive capacity can decide between failure and success, but this is not the case in a volunteer project, such as the Debian Project, which exists without competition.[13]

The absorptive capacity of DCs is challenged by the high volume of communication and information exchange that is at the core of the project.

2.2.4.5. Minimise divergence of tools

As noted before, DCs are volunteers who spend their free time on Debian, and time is generally a scarce resource. Package maintainers especially often face repetitive tasks as they keep track of necessary changes to software across new versions (see section 2.3.2), but a more general version of this trait applies to all contributors, probably spanning most FLOSS projects: divergence in their use of a tool or technique from how that tool or the technique was designed to be used. Such tools and techniques tend to evolve, and if this evolution includes changes that are incompatible with someone's non-standard use, time has to be invested to make things work again. Continuing use of an outdated version of a technology can bring similar problems in the wake of security problems, or when the circumstances under which a tool is used change; the user is then faced with the task to amend the tool, possibly through further divergence from the maintained version.

As a result of this, DC are frequently wary of modifying programmes or abusing them in non-standard ways.

2.2.5. Freedom and independence

Another motivation for participation is Debian's rigorous observance of freedom. The DFSG (see appendix A.7) express the project's dedication to building a complete operating system based entirely on free software. While one of the characteristics of a FLOSS project is the freeness of its produce, Debian GNU/Linux is the only GNU/Linux distribution which is built on a strong belief in freedom and does not suffer from licensing restrictions, which could restrict its users.

[13] This is not to say that Debian stands above its potential competitors, just that there are no hard facts, such as turnover or market share. Instead of money, the Debian Project is run on volunteer time and monetary donations, neither of which are essential for its survival [*cf.* Robles and Gonzalez-Barahona, 2006].

Here, I can make the link back to the third of the hacker ethics identified by Levy: all information should be free. This belief equally unites contributors to Debian [Coleman, 2005b, p. 321f].

The Debian archive contains a small number of "non-free" packages, which are maintained and distributed for convenience, but which are not considered to be an official component of the Debian System. In particular, the Social Contract (see appendix A.6) promises that the project "will never make the system require the use of a non-free component." The document is referring to the operating system as installed by users, but the underlying belief exists across all parts of the project. For instance, even though it has done so in the past, the Debian Project infrastructure does not build on non-free components currently, and is unlikely to do so in the future.

Incorporating proprietary tools into the workflow of a project creates a level of dependence, which might come to haunt the adopters at a later point: not only does the project place itself at the mercy of the vendor to keep the software usable and renew this licence, it is also not necessarily possible to extract the data out of such a system and migrate to something else. In this context, a reference to the Linux kernel project is in order. Linus Torvalds, head developer of the kernel, opted for the proprietary BitKeeper version control system, which had been specifically developed by its vendor to suit the kernel project's workflow, and for which Linux received a use-licence for free. Over a disagreement, the vendor revoked this licence, leaving Linus no other choice than to seek an alternative and eventually develop Git himself.

As said before, the Debian Project is unlikely to make a decision for a non-free product, especially now that it has freed itself of all such dependencies that existed in the past. A primary and well-guarded trait of the Debian Project is its independence. Even though several companies work closely with Debian, and it is not uncommon for companies to employ DCs to work on the operating system directly, the project has always steered clear from letting anyone external exercise influence over it.

Ex-DD Lars Wirzenius explains this adherence to freedom and independence in contrast to the choice by Linus Torvalds to enter a state of dependency through his choice of version control system[14]:

> "Linux" didn't accept, Linus did, and most kernel developers just followed his example; Linus is purely pragmatic, and has no ethical ideas whatsoever… so he's

[14] #debian-devel, 25 Jul 2009

> fine with non-free stuff if it makes his life a little bit easier. We in Debian, however, have Principles with a capital, flowery 'P' (unreproducible in plain text).

The independence is also a source of pride to DCs, as stated by Colin Watson during the Delphi study[15] (see section 6.2):

> I'm fiercely proud of the fact that Debian is "merely" an association of individuals and yet has achieved so much, and I'd hate to see it become corporatised somehow — assuming that it's balanced by moderate opinions, which it does seem to be.

2.2.6. Debian derivatives and Ubuntu

Debian GNU/Linux is an operating system on which around 100 projects build customised and specialised derivative distributions. While DCs mainly work with original authors ("upstream"[16]) to get their software integrated with Debian, the project is itself upstream to these derivatives. Collaboration across distributions poses additional challenges when it comes to the adoption of tools and techniques. Therefore, it seems reasonable to give a brief overview of the situation.

In theory, derivatives should make every attempt to re-integrate their changes back into Debian, or work with DCs to make Debian more customisable, to lessen the amount of divergence that the derivatives have to track (*cf.* section 2.2.4.5). Debian should proceed analogously with the original software authors. As a result, enhancements flow upstream, the set of people profiting grows, and only those changes that constitute a derivative's specialisation remain at the level of the derivative.

That is the theory, which was practically unchallenged for the longest time. While DCs were (only) encouraged by unwritten social rules to push as much as possible upstream, very little came back from the derivatives, but no one complained, maybe because no one noticed. Collaboration between projects took place, but more on an *ad hoc* basis than by default.

In 2004, Ubuntu, another Debian derivative, was released to the public, and quickly rose to become the most popular Linux distribution among desktop users. The media attention it received eclipsed exposure of the Debian Project, and several DCs started to wonder what Ubuntu would

[15] <20090726154705.GA16966@riva.ucam.org>

[16] Upstream refers to the (external) source of software packaged for Debian; *E.g.* the GNOME project is Debian's upstream for GNOME-related packages. Debian users, as well as derived distributions, are sometimes referred to as downstream.

do to Debian in the long run, and why they were still voluntarily doing what others got paid for.

Six years later, the Debian Project is alive as ever, and Ubuntu is still popular among desktop users. There has been friction between the projects,[17] but Debian has also benefitted a lot from the collaboration between some of the teams of the two projects.

Ubuntu has helped Debian realise that the project may have to reconsider its view on collaboration with downstream projects: derivative projects need to be able to submit their changes without much having to "jump through hoops", and those changes need to be considered on the side of the DCs.

Nonetheless, a strictly hierarchical flow of enhancements to software may be somewhat antiquated, but in the absence of proper coordination, it scales better than a chaotic peer-to-peer process, as advocated by Ubuntu founder Shuttleworth.[18] His company set out to provide tools designed to coordinate the peer-to-peer collaboration process (Launchpad[19] and Bazaar[20]), but those tools have not been adopted by the Debian community (and probably never will, due to pedigree and dependence considerations).

The relationship between Debian and its derivatives, and specifically Ubuntu, is relevant to my research in two ways: (1) cross-distribution collaboration requires stepping beyond social systems with shared norms, which provides an additional challenge to understanding behaviour; (2) the provision of single tools to support the collaboration has not yielded their wide-spread adoption in the Debian Project. For the study described in part II, I made sure that Ubuntu, as well as other derivatives and specialisations of Debian were represented in the Delphi discussion panel (see section 6.1), to add further diversity and broaden the results.

2.2.7. Communication basics

In a global project like the Debian Project, communication is essential. In fact, communication is what ties any FLOSS project together and enables it to move towards a shared goal. It is interesting to note that despite rare physical proximity, the level of communication is enough

[17] An interesting, recent starting point for further analysis of the relationship between the two projects may be the discussion on release cycle cadence: `http://lists.debian.org/debian-project/2009/08/msg00092.html` [4 Sep 2009]

[18] `https://wiki.ubuntu.com/MarkShuttleworth#Why%20are%20you%20funding%20Ubuntu,%20instead%20of%20giving%20the%20money%20to%20Debian?` [4 Sep 2009]

[19] `http://launchpad.net`

[20] `http://bazaar-vcs.org`

to yield the propinquity effect, which is the tendency for people to form close relationships found to normally increase with *decreasing* distance [Allen and Henn, 2006, Festinger et al., 1950].

DCs are very tightly interlinked by way of the various communication media. Since the spread of information about innovations is a core component of adoption in general, I will give a brief introduction of the use of electronic mail in the Debian Project, which is by far the most central medium. An overview of other communication channels can be found in appendix A.2.

Most avenues of contribution to Debian require a valid e-mail address: package maintainers are identified with their address, the Debian build and archive daemons send all notifications via e-mail, project members use electronic mail to vote, and to interact with the bug tracking system, and almost all discussion happens on mailing lists. Electronic mail is asynchronous in that messages are sent with the implicit understanding that they may not be immediately read and acted upon. This is an important feature of the medium in the global, time-zone-spanning context in which the project operates.

A handful of teams within the Debian Project operate several hundred topical mailing lists, which serve as primary discussion platforms for everyone related to the project. Those lists are governed by a set of guidelines geared towards high volume,[21] and it is expected of those interacting with the lists, who are often referred to as posters, to abide by the rules and be in control of their e-mail programmes to enable effective use of the medium. All traffic crossing the Debian lists is publicly archived,[22] searchable,[23] and indexed by all Web search engines. DCs furthermore have direct (shell) access to an archive of raw messages on a project-operated server.

A related, asynchronous communication medium is the Bug Tracking System (BTS),[24] which is also based on electronic e-mail, and sports a public web interface that is similarly indexed. The BTS organises electronic messages based on individual bug reports; please refer to appendix A.1.3 for more information.

Since e-mail is such an integral part of the Debian Project and used by contributors in multifarious ways, it is reasonable to assume that DCs are experienced users of electronic mail, and skilled in dealing with large volumes thereof.

[21] `http://www.debian.org/MailingLists/#codeofconduct` [26 Jun 2009]
[22] `http://lists.debian.org`
[23] `http://lists.debian.org/search.html`
[24] `http://bugs.debian.org`

2.2.8. Relation to organisational theory

The organisational literature covers several types of organisational types, some of which relate to the Debian Project. Heckscher and Donnellon [1994] discuss these types of organisation with a focus on bureaucracy, or rather post-bureaucracy, as an organisational form better suited to modern requirements and needs. In this section, I assume a similar point of view. While the Debian Project appears bureaucratic at first, it strongly resembles the interactive, post-bureaucratic organisation [Heckscher, 1994].[25] As some of the project's scalability problems may well relate to shortcomings of this ideal form of post-bureaucracy, it will help to put the two into perspective.

2.2.8.1. Bureaucracy and meritocracy

From an outsider's perspective, the Debian Project may easily be considered a bureaucratic behemoth: the extents of the project are large, and there appear to be no limits to the number of terms, roles, processes, guidelines, and rules. However, a bureaucracy is primarily defined through the exact specification of offices, duties, accountability, and the flow of information, which is usually exclusively vertical in an organisational hierarchy [Weber, 1947]. The Debian Project is *not* a bureaucracy in that sense, but more of a "virtual organisation", subject of section 2.2.8.3.

It is true that roles and guidelines are abundant, but those roles are not clearly defined, and the guidelines seldom binding. The voluntary nature of the project (see section 2.2.3) leaves no room for duties and accountability, and information flows directly, thanks in large to the available communication media (see section 2.2.7). Where a bureaucratic organisation clearly specifies *how* tasks are to be done, guidelines in the Debian Project only describe the desired result. The notable exceptions are the few clearly specified officers discussed at the start of section 2.2.2, which have been instituted to cope with the fast growth of the project. None of these are hierarchically defined or given full executive power, as stated before.

Despite the flat organisational structure, the Debian Project is considered and often described as a meritocracy, neither in the satirical sense in which Young [1959] invented the word, nor in the way in which scholars have used the term since: as an antonym of class-based or aristocratic systems, when achievement displaced heritage Bell [*e.g.* 1976, cited by Donnellon and Scully

[25] The birth of the project seems to coincide with the beginnings of scholarly interest in moving beyond static, hierarchical organisations. I deem it unlikely, however, that the there was any influence in between: Debian was created by hacker enthusiasts, while scholars at the time focused primarily on corporations.

[1994]]. It may well be that the term meritocracy was initially misused in the Debian and FLOSS contexts, and has since grown new or gotten rid of some of the connotations. Hence, it is in order to devote a short treatment to the development of the concept and differences in meaning.

According to Donnellon and Scully [1994, p. 64], a meritocracy is "so intrinsically associated with an individualistic, non-cooperative, bureaucratic approach" that meritocracy must be abandoned to move beyond bureaucracy. This does not seem to hold in the context of Debian, and possibly other FLOSS projects, which are defined by cooperation. In fact, it seems that meritocracy is in its traditional sense rooted in explicit comparisons and competition, while in the FLOSS context, it relates to respect of the individual: people with merit to their name are more likely to persuade and influence others in dialogue.

Daniels [1978] defines the principles of meritocracy three-fold: (1) selection of individuals for positions based on well-defined merits, (2) the means to develop and display such merits, including equality of opportunity, and (3) a relation between merit and reward . None of these definitions describes the way in which meritocracy is used to describe Debian's social structure, because (1) "selection" is too active a term in the context of a self-organising system such as Debian (in which "appointment" is the exception), (2) the means to develop and display merit are too multifarious, and not controlled for variables such as equality of opportunity, and (3) there is no (extrinsic) reward structure apart from individual motivation .

In the Debian Project, appointment as official project member, delegations, and other kinds of status change in the Debian Project depend entirely on need, and a demonstrated talent and ability (transitions to statuses with lesser privileges are rare as a consequence [*cf.* Jensen and Scacchi, 2005]). Due to each individual's voluntary involvement, as well as the individualistic aspects of meritocracy, membership of the various groups (or layers) has been found to be in continuous flux [González-Barahona and Robles-Martínez, 2003, Ye et al., 2004]. Steinlin [2009] combined statistical methods, game theory, and a quantitative analysis of contributions to Debian and found that cooperation — at least in Debian — builds upon notoriety, reputation, and merit.

The meaning of meritocracy in the context of Debian (and possibly other FLOSS projects) is thus much simpler than what is found in the literature on organisational theory. While in the literature, merit seems to be mostly linked to extrinsic rewards and assessment, meritocracy has two separate, specific effects, or symptoms, in a FLOSS context: (1) those who approach a task are the ones that get to make the decisions (but they are free to consult with others or build consensus

a priori), and (2) those with achievements to their name might be more respected and their voice attributed more persuasiveness/influence in discussions.

2.2.8.2. The interactive model vs. the closed community

Both of the effects of meritocracy in the Debian Project may be intrinsically regarded as rewards or used as the basis for assessments, but they are not extrinsic or explicit, as *e.g.* monetary rewards would be. However, they pave the way towards a deceptive variant of the ideal post-bureaucratic type of organisation, which Heckscher [1994] calls the "simple federation" or "negotiated order" [p. 34f]. Such an organisation defines itself more through anti-bureaucracy than post-bureaucracy, and is categorised by the lack of higher-order reflection and binding decisions, and thus the difficulty in developing an overall strategy.

Debian as an organisation approaches the "interactive" organisational type, defined by Heckscher [1994, p. 24ff] as the ideal post-bureaucratic form of organisation, in that it builds on "institutionalized dialogue", as well as in the other eleven points of the conceptual description offered by Heckscher.[26]

The existence of a meritocratic culture inside the Debian Project does not tenaciously link the project to bureaucracy, however. In fact, Gordon [1994a, p. 195] summarised Heckscher's idea of leadership in the interactive model as follows: "leadership is itself informal, based not on organizational position but on influence, which moves in a fluid manner from one individual to another depending on the problem at hand and the experience and judgement that different people can bring to bear." This sounds remarkably similar to how I defined meritocracy in section 2.2.8.1, and is also reminiscent of the "organic organisation", characterised by network structures [*cf.* Burns and Stalker, 1994]. Krackhardt [1994, p. 219f] doubts that these are truly fluid and organic, but rather emergent and stable.

It is interesting to note in this context the criticism by Krackhardt [1994, p. 212f] of the interactive model: his "law of N-squared" states that in groups of increasing size, the set of correspondents of each individual does not increase linearly. A ubiquitous communication medium, in combination with a directory, which enables everyone to get ahold of everyone else independent of the size of the organisation, causes the more knowledgeable to be overwhelmed with requests, diminishing their ability to contribute to the organisation. In the Debian Project, this problem is overcome by promoting a strong culture of communication via public forums, rather

[26] The sixth point on principles *vs.* rules is a special case to which I shall return shortly during the discussion of communities-of-practice in section 2.2.8.5.

than peer-to-peer, thus giving everyone the choice whether to follow or participate in specific, topical discussions (see section 2.2.7).

Even though the Debian Project exposes none of the classic bureaucratic features, such as hierarchical governance, assessment, and extrinsic rewards [*cf.* Weber, 1947], the effects of meritocracy — being able to make decisions by forging ahead, and reap respect rooted in one's contribution — create an individualistic decision culture in which a shared objective cannot be specified beyond a shared, idealistic vision. As such, the Debian Project suffers from becoming a interactive organisation very early on in that it lacks the empowering effects of leadership: orientation towards common objectives, motivation of members, and a clear direction providing for decisions to be made [Hackman, 1980].

Gordon [1994a, p. 200ff] posits the interactive organisation to be worse than the "closed community" (which Heckscher [1994, p. 30ff] had dismissed as deceptively non-ideal), and summarises the closed community in four points, which I abbreviate in the following:

1. Consensus about organisational goals;
2. A sense of community and peer sanctioning to keep people aligned;
3. Local knowledge and intelligence at every level, freedom of choice of what to work on, and technical experts available to help with decisions;
4. Leadership that generates and symbolises the values and aims of the organisation, including shared identity, objectives, and altriusm.

Debian exhibits the first three points: a vision of freedom and quality unites the community, a culture of discussion fosters exchange and continuously re-aligns outlying factions, and everyone is free to do almost anything in almost any way they want, with discussion forums and the technical committee (see section 2.2.2) available to them in case of uncertainty of disputes.

It is in the fourth point that Debian differs from the closed community and rather tends towards the interactive model: there is no clear leadership in the project, but a "cabal" is commonly alleged to exist. Related to a further shortcoming of the interactive model, which Krackhardt [1994, p. 215f] labels the "iron law of oligarchy, [...] rule by the few", a cabal is a group of core developers that make decisions in non-public fashion and then use their influence to pull the rest of the project along.

Heckscher [1994] considers the interactive organisation to be an ideal, and as such a theory. Even though the Debian Project may come close to the conceptual ideal, it does not seem sensible

to determine the degree to which this is the case, if only because the ideal type is not without faults anyway. Instead, it might be worthwhile to investigate the degree to which leadership, and thus the move towards the closed community model, could benefit the project. This would be beyond the scope of this work, however.

2.2.8.3. The virtual organisation

The virtual organisation [*cf.* Davidow and Malone, 1992] was conceived as the logical next step in organisational theory, at a time when computers became powerful and decentralised enough to part with the Weberian bureaucracy [*cf.* Weber, 1947]. The digital age made information independent of time and space and therewith announced the end of the classically structured organisation [Nohria and Berkeley, 1994, p. 119]. Rather than structureless, however, the virtual organisation continues as an organisation — as a "complex system of mutually dependent individuals" [p. 123] — but less holistic and less highly differentiated. Structures form and dissolve *ad hoc* as a result of actions of the members, and may even span different organisations [p. 120]. The individual — the "knowledge worker" — assumes a role that Nohria and Berkeley describe as "a kind of holographic model of the organization at large" [p. 123].

The Debian Project may well be seen as one such virtual organization, but it goes beyond the classical definition. In particular, the Debian Project never existed as a non-virtual entity, and much of what Nohria and Berkeley call "utopian visions" and "eloquent promises" [*ibid.*, p. 125] are taken for granted: electronic communication, and the virtual space in which project members act and collaborate, are defining properties of the community surrounding the Debian Project. This may well be due to the Debian Project starting up after the concept of the virtual organisation had been formalised.

Nevertheless, the "new kind of static" introduced by electronic media as a "powerful new form of coordination people across time and space" [*ibid.*, p. 122] is present in the Debian Project. Discussions among Debian Project members often last days or weeks, while face-to-face groups might have found consensus in shorter time [Sproull and Kiesler, 1991]. Increasingly, face-to-face interaction takes the salient role posited by Eccles and Nohria [1990], and physical meetings bringing project members together are being organised with increasing frequency.

Electronically-assisted or -mediated communication is less dependent on presence cues, such as age and looks [*cf.* Poster, 1990], although cues such as status and gender continue to play a significant role due to the strong attention to identity of project members (*cf.* appendix A.3.5).

If virtual media cause the loss of hierarchical structure in the post-bureaucratic sense, then the strong commitment of Debian's contributors (see section 2.2.4) may be seen as a counter-reaction.

2.2.8.4. Debian as a team organisation

Donnellon and Scully [1994, p. 78f] argue that "the common approach to becoming post-bureaucratic appears to have two incompatible tendencies: [...] *major* revisions in the division of labour, such as the performance of tasks by teams [with] only *minor* changes in how evaluation and rewards are handled." While team work readily crosses hierarchical boundaries, evaluation and rewards still take place chiefly vertically. As an alternative, they propose [*ibid.*, p. 79ff]

1. to rely on intrinsic motivation;
2. to celebrate excellence as a group, rather than identify or focus on individual stars;
3. and to avoid the concept of merit altogether, in language and in culture.

It requires a bit of a stretch to apply this to the Debian Project, which is, after all, a voluntary organisation with neither rewards nor evaluation.[27] I will briefly address each item in turn.

Motivation — Contributors to the Debian Project are generally intrinsically motivated to work on the project, which is very much a core trait of the voluntary involvement. An undetermined number of project members are motivated extrinsically, because *e.g.* their employers pay them to be involved with the project. However, there is never a direct link between involvement and rewards (see section 2.2.3). Debian hence already relies on intrinsic motivation.

Excellence — In general, project members identify as members of the project at large, and stars are not celebrated. Occasionally, individuals rise to stand out through their actions, findings, or media attention, but such focus is not permanent. The sole exception may be the Debian Project Leader (DPL), elected yearly, who is a primary contact for the press, and who gets to stand in the spotlight and answer high-level questions about the project. This is more often regarded a burden than a privilege.

[27] The exception here is the evaluation required as part of the vetting of new project members (see appendix A.3.5).

Merit — I refer to the discussion in section 2.2.8.1, in which I established that merit in the context of the Debian Project does not have the meaning traditionally associated with the word in bureaucratic organisations, where it seems to be closely related to rational fairness [Donnellon and Scully, 1994, p. 86]. The suggestion to remove the concept of merit entirely from language and culture does not seem applicable, because merit in Debian is only a means to explain how influence and executive power fluctuate by those who study or speculate about the project; it is not a term in common use. The Debian Project does not build on merit, but meritocracy provides a useful lens to assess the project's organisation.

As the team organisation is regarded an instance of post-bureaucracy, the above again suggests that Debian is not a bureaucracy in the traditional sense of the word, but rather approaches the ideal, post-bureaucratic organisational type, which I have argued before.

2.2.8.5. Communities of practice

The term of "communities of practice" was coined by Lave and Wenger [1991], who used it in the context of knowledge acquisition (situated learning). Today's meaning is reflected more accurately in the description by Wenger [1998]: people in a joint enterprise, linked informally to each other through their involvement in certain common activities, form a "community of practice". Wasko and Faraj [2000] use the term in the context of broader knowledgement management, to contrast knowledge as an object and knowledge embedded in individuals (knowledge is private property) to knowledge embedded in a community. Hildreth and Kimble [2002] similarly approach communities of practice from the knowledge management perspective, calling a community of practice as "a group where [softer types of] knowledge are nurtured, shared and sustained".

On the basis of the characteristics of communities of practice identified by Hildreth and Kimble [2004], the Debian Project is host to multiple communities of practice, each of which "posesses" a great deal of knowledge managed through public discussion, and publicly available, group-maintained knowledge resources (*e.g.* wiki and web pages, documentation, guidelines, *etc.* [*cf.* Clark and Brennan, 1991, Wenger, 1998]). Debian *per se* is "a joint enterprise in as much as it is understood and continually renegotiated by its members" [Hildreth and Kimble, 2004], following a common goal (see sections 2.2.4 and 2.2.8.2). In fact, FLOSS at large embodies the core ideas of communities of practice: knowledge sharing and cooperation among participants of FLOSS projects. A FLOSS project, however, primarily strives to produce a software.

This differentiates it from a classic community of practice, which more commonly unites people individually pursuing goals that are related to each other, such as teachers, crafters, engineers.

It seems that the literature has been influenced by the development of the communication and sharing culture of the Internet (and FLOSS, as a group of the most visible and innovative ambassadors of public knowledge management). For instance, it is unsurprising that Kimble et al. [2001] found that "the literature has shown no reason why in theory a community of practice might not be able to exist in a distributed international environment". In their case study, they found that the sharing of soft knowledge was the central problem of a virtual community of practice. The communication culture and density of exchange in the Debian Project (see section 2.2.7), however, shows that it is possible for soft knowledge to be nurtured and managed in a distributed organisation — although the knowledge is not managed optimally, largely due to the fast-paced nature of the project of this size, and the volume of information generated on a daily bases.

Since FLOSS predates the coining of the term "communities of practice" by decades, it is unrealistic to expect the literature to provide insights, other than a structured frame of reference grounded in sociology and anthropology. It is without doubt that findings from research in those fields could be used to advance the Debian Project and FLOSS. Nevertheless, I could not find studies applying the communities of practice model to FLOSS projects.

2.3. Debian packaging

2.3.1. Tools, techniques, processes, approaches, solutions, and methods

Throughout the following discussion, as well as the entire thesis, I use the words tools, techniques, processes, approaches, solutions, and methods interchangeably. Each of those is subtly different, but in the context of this discussion, they all refer to means of doing a task. Moreover, they are all subject to innovation, as shall be discussed in chapter 3: a known task may be approached through the innovative application of old or newly invented tools and techniques, via new processes, by way of previously unknown approaches, with novel solutions, or using methods that stem from other domains or have been freshly conceived.

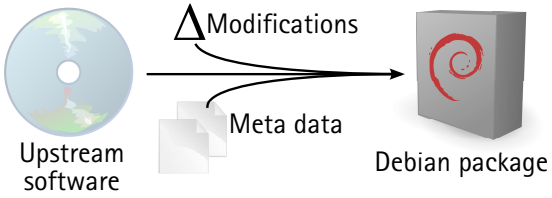

Figure 2.1.: *The process of "Debianisation" by which upstream software is modified and packaged for installation on a Debian System.*

2.3.2. The Debian packaging workflow

It is outside the scope of this text to cover development practises of the Debian Project. However, since development practices and processes differ greatly across FLOSS projects [Michlmayr et al., 2005], a short discussion is in order. In the following, I present a run-down of the standard Debian package maintenance process as an example of a Debian development process in need of improvement. I also highlight some of the problems, and hypothesise solutions.

Figure A.1 on page 311 illustrates the various stages a Debian package traverses during its lifetime. A package is born when a user (or developer) finds (or writes) a piece of software, and wants to make that software available via the Debian archive. Through a process of "debianisation", depicted in figure 2.1 on this page, the software is transformed to meet the requirements of the Debian System, and meta data are attached to ensure that the package fits in with all the other packages in the Debian archive (see appendix A.1.5). The package is then uploaded to the Debian archive, subjected to quality and licence checks, before it begins its lifetime as official Debian package in the unstable archive (see appendix A.1.4). The packager becomes the maintainer.

Every package in the Debian archive has an associated bug tracker[28] (see appendix A.1.3). As the package trickles through the archives and eventually becomes part of an official, stable Debian release, the number of users grows, and bugs will be found and reported. It is the maintainer's task to process these bugs, each of which can be either Debian-specific, or applicable to the upstream version of the package.

In the first case, the maintainer might fix the problem, then prepare and upload a new revision of the package. In the package's changelog, s/he notes the fix and bug number, which causes the Debian archive management scripts to mark the bug report as done. To make life easier for other distributions, which may be tracking the Debian package, the maintainer could manually

[28] `http://bugs.debian.org/`*`packagename`*

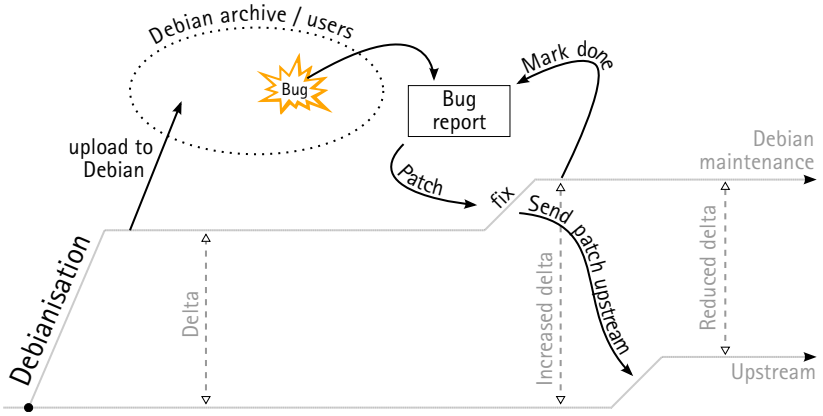

Figure 2.2.: *Debian maintenance involves tracking differences between the original (upstream) source and the Debian package. If bugs are fixed, it is in the interest of the maintainer to push the fix upstream so as to reduce the delta.*

attach the patch fixing the problem to the bug report, but this is rarely done due to the extra effort involved; unfortunately, it cannot be automated reliably, a point to which I shall return shortly.

If the reported problem applies to the upstream package, the maintainer may similary fix the problem in the Debian package. It is then in the maintainer's interest to forward the patch to the software's author for upstream inclusion, which would reduce the volume of differences s/he has to track between the original software and the version published in the Debian archive (patch management). Nevertheless, the maintainer will have to track the patch until a new upstream release incorporates it. Some patches against the upstream code may be Debian-specific and thus have to be tracked indefinitely, as they cannot be submitted to upstream. Figure 2.2 on the current page is a simplified illustration of what it means to track differences.

Should the maintainer run out of time before solving the problem, and forget to make his/her work available, someone else approaching the problem has to start from scratch. S/he might append a patch to the bug report, but could also just upload the fixed package to the Debian archive. Once the maintainer again finds time for the package, s/he needs to figure out where the patch can be found, apply it, resolve conflicts with his/her earlier attempts, and manually mark the bug as fixed so that the report can be closed. This situation is sketched in figure 2.3 on the facing page.

What this outline strives to illustrate is the complexity of the process and the number of related but separate tasks awaiting the maintainer in response to every bug report. Current Debian

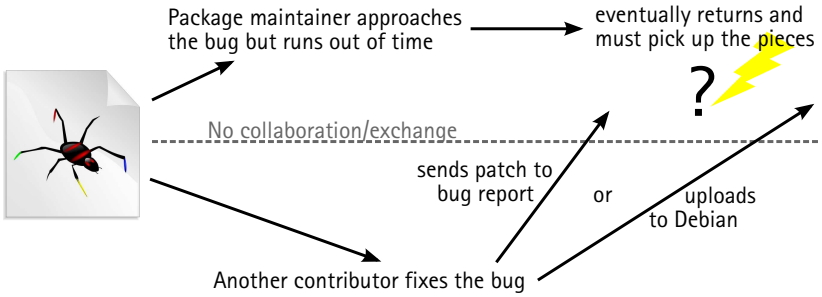

Figure 2.3.: *In the absence of proper maintenance techniques (*e.g. *a VCS, work may be duplicated and require later conflict resolution.*

package maintenance involves many repetitive tasks, which hold a high potential for human error. In addition, as the complexity of a package increases (due to large numbers of bug reports and a slow upstream, or related to fundamental differences between the upstream software layout and Debian's standards), package maintenance becomes tedious and error-prone, negatively affecting the quality of the entire distribution.

In an ideal world, package maintenance, patch management, and bug reports would be integrated in such a way that patches are treated as separate, logical entities stored within the package. Each patch could be manipulated (applied, forwarded, removed) separately, and status changes would be automatically recorded in the associated bug report.

Using VCSs for package maintenance is a strong trend in the Debian Project, but without any standard practices; or rather, in the spirit of Grace Hopper: "the wonderful thing about standards is that there are so many to chose from."[29] This is not necessarily a bad thing, as competition fosters quality, but it dampens cross-package collaboration and hinders one-time contributions: users trying to fix bugs in a package as part of a Bug Squashing Party (BSP) may first have to learn how to interact with the VCS. It also makes it difficult to automate the aforementioned integration of the BTS with the VCS.

2.4. Process improvement and volunteers

The previous sections provided a glimpse of the Debian Project and its fundamental processes. Just as the Debian System is continuously evolving, so is the community that produces it, as well as their work modes. For instance, in stark contrast to the early days of the Debian Project,

[29] This quote is widely attributed to Grace Hopper, but appears in Andrew Tanenbaum's *Computer Networks* [1996] without attribution.

people are less stationary today, and even though Internet connectivity is becoming more and more ubiquitous, the need to work off-line (in a train, or on the beach) is more present than ever. However, many of Debian's core processes are not ready to accommodate those workflows.

Off-line operation is but one (symptom of an) example of how the community has evolved. Other areas, in which processes have not scaled over the years include collaboration between distributions and derivatives, and targeting Debian at specific usage domains. In addition, the sheer size and complexity of the system as a whole has increased (*cf.* appendix A.4), calling for new development approaches.

Efforts to improve the processes are in place,[30] but their results (where results already exist) are not being adopted as readily as they should be. Because the Debian Project is made up entirely of volunteers, no-one can be told what to do or how to approach their tasks (see section 2.2.3). Instead, Debian contributors prefer to choose their own tools, based on criteria which range from technical benefits to ideology, from pragmatism to sheer stubbornness. In addition, once settled, a developer is often unmotivated to change his/her techniques at a later point in time, or may only be able to do so through considerable additional effort without the ability to assess whether the effort will pay off.

Without adopters, a tool or technique cannot evolve. For coordinators, integrators, or tool designers, no guidelines exist that could help them properly cater to everyone in the target group, and entice their involvement to help further the innovation. Instead, everyone is on their own when trying to improve collaboration in the project.

This impedes progress: some ideas may be good, but never manage to convince more than a handful, who fail to communicate them to the majority, because they do not understand others' decision behaviours. The volunteer nature of the Debian Project can have a considerable impact on the diffusion rate, and lack of the ability to drive change slows down the rate of diffusion. As a result, improved ideas do not effectively rise to become competitors with existing approaches, people will have no reason to change, and progress is slowed.

This situation is a core motivation for this research (see section 1.2). I have postulated improved workflows,[31] and participated in various discussions on how to improve collaboration in the project. Even though the availability of a solution is a prerequisite for its adoption, considerations are in order to drive the diffusion of a solution in the Debian Project; an assumption that

[30] For instance, I spearheaded the `vcs-pkg` effort (`http://vcs-pkg.org`) to investigate how to use VCS for cross-distribution collaboration.

[31] `http://madduck.net/blog/2005.08.11:rcs-uploads/` [17 Oct 2007] and `http://madduck.net/blog/2007.10.03:packaging-with-git/` [17 Oct 2007]

any technique, however advanced or appropriate, will be widely adopted in the project would be naïve, as it does not take into account the characteristics of the Debian Project, its community, its members, and the flow of information.

Marketing has a negative connotation in some academic circles for being used synonymously to "manipulating human purchasing behaviour for commercial advantage" [Rogers, 2003, p. 84], and this is common too in the context of the Debian Project, where commercial links and terms are not in accordance with the project's adherence to freedom and quality (see section 2.2). However, outside of the corporate world, marketing still exists when needs are created or met. Marketing in a not-for-profit context is referred to as "social marketing" [*ibid.*], and this form of marketing can be found in FLOSS, where people talk about innovations and information spreads across networks.

Driving change does not need to imply authoritarian diffusion. In particular, it needs not be about subversion or coercion. Among volunteers, the decision to adopt fundamentally lies with the individual. Marketing in such a context is just targeted communication, without any negative strings attached. Marketing messages can create needs, but more frequently, the messages need to enable their recipients to cross the chasm between the *status quo* and the potential improvements brought about by the innovation that is being marketed.

To be able to target marketing messages, emitters need to be aware of the targets and their characteristics. Through my research, I seek to enumerate and describe the salient influences to adoption decisions among DCs. When innovations are to be adopted, knowledge of these influences would enable everyone (peers and marketeers alike) to identify and close gaps of information, and enter competing ideas into the field more effectively. With healthy competition, only the best tools and techniques will emerge as winners. And because competition drives progress, the project should be able to scale better as a result.

2.5. Previous research on Debian

As stated in the introduction, Debian has been the topic of many investigations and articles. By far, most of the coverage comes via electronic publications, mailing lists, and Internet discussion forums. As one of the largest FLOSS projects, however, it has also been subject of books,[32] and a fair number of scholarly articles.[33]

[32] `http://debian.org/doc/books` [8 Nov 2007]

[33] I have made efforts to bring together researchers of Debian with a mailing list: `http://lists.debian.org/debian-research/`

Most commonly, the project is cited for its strict adherence to moral, ethical, and ideological standards [Coleman, 2005a, Feller and Fitzgerald, 2002, Raymond, 1999], and on licencing topics [Garzarelli and Galoppini, 2003]. It has also been studied in the context of project success measurements [Crowston et al., 2006, Senyard and Michlmayr, 2004], project management [Garzarelli and Galoppini, 2003, González-Barahona and Robles-Martínez, 2003, Michlmayr, 2004, O'Mahony and Ferraro, 2004, 2007, Robles et al., 2007, Robles and Gonzalez-Barahona, 2006], release management [Michlmayr, 2005a, 2007, Michlmayr et al., 2007a], and quality assurance [Michlmayr et al., 2005, Michlmayr and Senyard, 2006]. The volunteer nature of the project has also received attention [Michlmayr, 2004, Michlmayr and Hill, 2003, Michlmayr et al., 2007b, Robles and Gonzalez-Barahona, 2006, Robles et al., 2005b,c]. Robles et al. [2006b] have explored social network analysis techniques for community-driven software projects and used Debian as case study, and Sowe et al. [2006, 2007] study Debian's social network with a focus on knowledge brokerage.

Steinlin [2009] has studied the influence of notoriety and merit on the cooperation between DCs, in particular looking at bug tracking and identifying correlations between reputation and the willingness of others to spend time working on faults reported by reputable individuals.

The Debian package archive has served as basis for network analysis studies [Fortuna et al., 2007, Robles et al., 2006b], statistics [Amor-Iglesias et al., 2005a, Robles et al., 2006a, 2005a], and effort analyses [González-Barahona et al., 2001]. Thanks in large due to the high quality of meta data found in the archive, Debian is a prime source for sampling [*cf.* Späth et al., 2007], and it is beginning to gain momentum in software product line studies [Bermejo and Dai, 2006, Kojo et al., 2003].

To my knowledge, no research has been conducted into the Debian workflow, or method diffusion/adoption within the project, nor in another FLOSS context. The closest I have encountered is a working paper by Späth et al. [2009] in which the authors analyse code reuse in the Debian Project and identify a number of factors that cause programmers to prefer one library over another. Also, Häfliger et al. [2008] find that "developers reuse code because they want to integrate functionality quickly, because they want to write preferred code, because they operate under limited resources in terms of time and skills, and because they can mitigate development costs through code reuse." However, both studies investigate code/component reuse during programming, and while an overlap is undeniable, my work focuses on tools and techniques used to approach tasks.

Chapter 3

Assessing diffusions

The previous chapter introduced Debian. In particular, I identified the most pertinent characteristics of the Debian community and project contributors (section 2.2), and described one of the core task areas within the project — packaging (section 2.3.2). I focused on some of the shortcomings of the current approach, which prevent the project from scaling more appropriately alongside its rapid growth. Improved processes could help lessen the burden on individuals, reduce the error rate, and pave the way for accelerated growth of the project, without impairing quality. However, even with such improved processes available, the volunteer nature of the project can have considerable effect on the diffusion of such processes, and presents a challenge of how to drive change in the project. This challenge was subject of section 2.4, where I postulated that knowledge of the characteristics of adopters is a prerequisite to engineer and effectively diffuse improved tools or techniques in the project.

In this chapter, I build the basis for the later presentation and discussion of influences to adoption behaviour of Debian Contributors (DCs). I start by defining the important terms used henceforth. Next, I briefly present a number of existing frameworks that have been developed to assess diffusions. Finally, I identify related studies of diffusion research in the context of FLOSS projects.

3.1. History and terminology

If an idea is widely accepted, it propagates faster from one person to the next; the process underlying this propagation is called diffusion. Not only ideas diffuse, but also products, strate-

gies, theories, and the many other types of innovations; and diffusions are studied in numerous fields: politics, marketing, process improvement, and engineering, to name but a few. It is thus unsurprising that the study of diffusions, despite its young age, has received considerable attention and continues to be a busy research domain with a large volume of output.

The earliest treatment of the study of diffusions I could find is in Gabriel Tarde's book *The Laws of Imitation* [Tarde, 1903]. Tarde, a French sociologist and social psychologist, who views society as made up of individual interactions between people, claims that "society is imitation" [*ibid.*, p. 74] of peers by peers. In this context, imitation is the prerequisite of diffusion, because ideas spread by being imitated. Core to Tarde's theory is the search for traits that determine the successful spread of an object of imitation, also referred to as innovation [*ibid.*, p. 140]:

> Our problem is to learn why, given one hundred different innovations conceived of at the same time — innovations in the form of words, in mythical ideas, in industrial processes *etc.* — ten will spread abroad, while ninety will be forgotten.

To date, more than 5,000 research papers have appeared on diffusions [Rogers, 2003, p. xviii]; while almost all of these studies treated the respective diffusion as separate and did not consider potential insights gained from other diffusions, Rogers [1962] was the first to present a concerted approach (see section 3.2.2.1).

In the following sections, I introduce the core terms of diffusion research. The terminology of the domain is very convoluted and definitions from business and marketing usually contradict their counterparts from the social sciences. Given the ubiquity of marketing terms, thanks to today's media, I deemed it appropriate to highlight the differences before taking a stand. Rogers [2003] is widely cited and accepted as an authoritative reference on the topic of innovations in the social sciences. A scholar of communication and technology adoption studies, he gained interest in the diffusion of innovations through exposure to rural sociology, and has since studied several thousand diffusions in copious different fields. While it is tempting to settle on his terminology, alternate definitions are occasionally more appropriate to this research.

3.1.1. Innovation and invention

In later sections, I often use the word innovation, so it is important to establish its meaning. Both in academic and industry circles, innovation is a buzzword with an unclear meaning. The

word is overused for its seemingly positive connotation, when in fact it does not imply success; politicians and marketing people are especially fond of its (ab)use. Scholars, dictionaries, and the media disagree over the actual meaning. For instance, the Oxford English Dictionary[1] double-defines innovation as (1) the action or process of introducing new methods, ideas, or products; (2) a new method, idea, product, *etc.*; in the social science literature, an innovation is "the successful implementation of creative ideas within an organization" [Amabile et al., 1996].

Pierce and Delbecq [1977] note that "change, innovation, invention, creative be havior, and adaptation have on some occasions gone undefined, and on others have been defined and interchangeably used," and provide a brief literature review of the "range of definitions and divergence in conceptualizations of innovation." The following discussion synthesises a consistent meaning for this text, and establishes the difference between innovation and invention.

By the definition of Rogers [2003, p. 181], an invention is "the process by which a new idea is discovered or created", while an innovation is only a perceived novelty [*ibid.*, p. 12]. Re-invention is the process of changing or modifying an innovation in the process of implementing it [*ibid.*, p. 180].

Kline and Rosenberg [1986, p. 279] acknowledge that an innovation is often perceived as a new product (an "invention"), but that there are alternative interpretations for the word: new processes, substitution of component materials in production, reorganisation of production leading to increased efficiency, and even improvements in "instruments or methods of doing innovation" [*ibid.*, the cyclical definition is theirs].

According to Schumpeter [1942, p. 89], an economist, an invention is economically irrelevant as long as it is not carried into practise. Earlier, Schumpeter [1934, 1939] had defined innovation as the commercial or industrial application of something new, thus suggesting that an innovation is what makes an invention economically relevant. Furthermore, von Hippel [1988, p. 17ff] positions the consumers as the most important source of innovations.

By the definition of Rogers [2003, p. 12], an innovation is only a perceived novelty, without any connotation of implementation or success. Later in his tome, he explicitly defines the innovator (who innovates) as the person putting an innovation to use: "the innovator must be able to cope with a high degree of uncertainty about an innovation at the time he or she adopts" [*ibid.*,

[1] `http://www.askoxford.com/dictionaries/compact_oed` [31 Oct 2007]

p. 282]. For completeness: the adjective "innovative" then applies to the degree to which a person innovates

Especially in the FLOSS context, and thus in the Debian Project, innovation may quickly follow invention, giving rise to "open innovation" [Chesbrough, 2003]. Contrariwise, an invention may exist for a long time before it is noticed, or before adopters can put it into context, but that does not diminish the novelty in the eyes of the potential adopter (see section 3.1.2) at time of exposure. Moreover, independent of the age of an invention, an adoption process does not start before it is picked up by potential adopters. Therefore, it makes sense to use innovation to refer to a novel idea that gives rise to an adoption process.

3.1.2. Diffusion, adoption, and critical mass

Diffusion is about communication [Mahajan et al., 1990]. Rogers [2003] defines the diffusion of an innovation to be "the process in which an innovation is communicated through certain channels over time among the members of a social system. [...] Diffusion is a kind of social change, defined as the process by which alteration occurs in the structure and function of a social system" [p. 5f]. Alteration in this context may refer to knowledge, beliefs, or practice: ideas can be diffused, just as well as tools or techniques can.

The rate of diffusion is the speed of communication, or the rate with which ideas are passed from one member to the next. A diffuser is an individual or agency driving a diffusion. Thus, a company's marketing department, individuals, and whole groups can take on the role of a diffuser. Another type of diffusers are change agents, individuals who influence clients' innovation-decisions in a direction deemed desirable by a change agency, whether the goal is adoption or rejection [*ibid.*, p. 366].

The term adoption refers to the psychological process leading up to a consumer's individual decision to accept an innovation [Ozanne and Churchill, 1971]. Potential adopters are those confronted with an innovation and who are not unlikely to adopt it; the subset of people accepting the innovation are the adopters, and those who do not adopt are said to reject an innovation.

Rogers [2003, ch. 7] separates adopters into five categories, depending on how early they adopt with respect to the surrounding social system: innovators are the first (2.5%), followed by early adopters (13.5%), the early majority (34%), the late majority (34%), and finally, the laggards (16%) [*ibid.*, p. 281ff]. Moore [1991] concentrates on the divide between

early adopters ("visionaries") and the early majority ("pragmatists") and how to overcome this chasm.

A common way to measure the success of diffusions is the rate of adoption, which Rogers defines as the "relative speed with which an innovation is adopted by members of a social system, [...] usually measured by the length of time required for a certain percentage of the members of a system to adopt an innovation." [Rogers, 2003, p. 23].

A related concept is awareness-knowledge: every innovation adoption is preceded by some form of knowledge about the adoption, and the rate at which awareness-knowledge diffuses can have a significant impact on the rate of adoption, although earlier knowledge does not imply earlier adoption [*ibid.*, p. 174]. Knowledge spreads more rapidly than an innovation [*ibid.*, p. 214], and increasing the rate of awareness-knowledge is one way to speed up the innovation-decision process [*ibid.*, p. 212].

With the Internet and its broad-band communication channels, information spreads at a swift rate, and the rate of awareness-knowledge rises significantly. With reference to the concept of adoptive capacity (AC) (see section 2.2.4.4), it is questionable however, whether this rate of awareness-knowledge continues to exert an effect on the rate of adoption. In high-volume communication channels, information drowns. This aspect — the effectiveness of high-volume communication media in terms of information transport — has not been adequately studied.

A diffusion may also be considered successful if a critical mass of adopters is reached [*cf.* Markus, 1990], which is the point when the number of adopters is high enough such that the rate of adoption starts to sustain itself. Once this point is reached, a diffusion is said to be irreversible [Rogers, 2003, p. 343ff].

3.1.3. Adoption process and innovation stages

The marketing literature seems to be mostly concerned with successful diffusions, meaning diffusions that elicit adoptions, which are split into three phases: initiation, adoption, and implementation. Kwon and Zmud [1987] note that such a view "excludes any post-adoption or post-innovation evaluation process." Furthermore, an explicit distinction between individuals and a whole organisation adopting is often missing [Pierce and Delbecq, 1977].

Several models of the adoption process exist, which build on the concept of stages and transitions between them. I have not found a different type of model for the adoption process, which

is not to be confused with models used in variance research to represent properties and attributes. Even though the existence of stages in the adoption process has not been validated [Rogers, 2003, p. 195], the evidence strongly suggests that stages exist [*ibid.*, p. 198], but that they may not be sharply distinct from each other, not obvious to the adopter [*ibid.*, p. 195]. In any case, stages are useful for simplification.

As opposed to the view that adoption is an act between initiation and implementation, Rogers purports that adoption is rather one possible outcome of a potentially lengthy process undergone by an adopter: "adoption is a decision to make full use of an innovation as the best course of action available" [*ibid.*, p. 177]. He calls the process the "innovation-decision process," and through his research he identified and validated the existence of five stages in the adoption process (see appendix B.1.3: knowledge, persuasion, decision, implementation, and confirmation. Different to the marketing literature, his model also accommodates failure at several stages: disinterest, rejection, and discontinuation.).

In the context of adoption as a process, change agents (see section 3.1.2) play differing roles at the various stages. Initially, change agents may identify or create needs, then aid people in the assessment of an innovation, and finally support them with their implementation [*ibid.*, p. 369f].

Organisational adoption In an organisational context, *i.e.* when an innovation is being adopted not by a single individual but by a group at once, Rogers builds upon the aforementioned three phases in organisational innovation, and presents a similar, but disjunct five-stage model (see table 3.1 on the next page): agenda-setting and matching are the two stages corresponding to the initiation phase, and are followed by a decision (adoption phase). The implementation phase consists of three stages: redefining/restructuring, clarifying, and routinising.

Fichman [1992], on the other hand, refers to Kwon and Zmud [1987] for organisational innovation. Kwon and Zmud have performed a more granular break-down of the same process across six stages, calling it the information systems (IS) implementation process. They align the stages with Lewin's change model: unfreezing, change, refreezing [1952]. The initiation stage is the lonely occupant of the unfreezing phase, the change phase includes adoption and adaptation, and refreezing spans acceptance, the use-performance-satisfaction stage, and ends with the incorporation.

Rogers [2003, ch. 10]	Kwon and Zmud [1987]	Parallels
Agenda-setting	Initiation	perceived need or pressure to change evolves
Matching	Adoption	resources are allocated and a solution sought
Redefining/restructuring	Adaptation	innovation and organisation converge
Clarifying	Acceptance	understanding of innovation grows
	Use-performance-satisfaction	benefits become noticeable, further spreading
Routinizing	Incorporation	integration into regular activities

Table 3.1.: *Comparison of the organisational innovation stage-models by Rogers [2003, ch. 10] and Kwon and Zmud [1987], and the parallels between the stages. The dual-mapping of the "clarifying" stage (4th stage of the first model) to stages 4 and 5 of the second model is a result of the stages 3 and 5 of Rogers' set mapping perfectly to Kwon and Zmud's stages 3 and 6. The mapping makes sense, since "clarifying" starts with "acceptance" (understanding), and ends with "use" (spreading).*

The two organisational models are very similar to each other. In table 3.1 on this page, the two have been placed side-by-side, and the model by Kwon and Zmud [1987] emerges as the more granular of the two.

Kwon and Zmud explicitly include "feedback loops" between the stages to make the model more applicable to reality, and Rogers similarly states that such stages are only "a means of simplifying complex reality, [...] a social construction, a mental framework," and that there exists no empirical evidence that the stages actually exist [*ibid.*, p. 195].

In particular, the six-stage model differentiates between acceptance, and what they call "use-performance-satisfaction" for lack of a clear precedence relationship among those three "assessments". The difference between the two is that the second factors out benefits that result from the organisational adoption of an innovation, such as improved performance, or user satisfaction, whereas "acceptance" merely specifies the point at which the organisation as a whole as understood the innovation to the point where it can put it to use. According to the authors, acceptance precedes benefits in at least two cases: "when *use is voluntary*, and when performance is dependent on committed, rather than vapid, use" [*ibid.*, p. 232, my emphasis]. In this light, the distinction seems appropriate in the context of research on the adoption behaviour in volunteer projects.

3.1.4. Network externalities and excess inertia

The study of diffusions distinguishes between two types of innovations: those which can be freely adopted by members of a social system, without affecting any other members, and innovations, whose singular adoption has an effect on the rest of the social system.

The literature refers to effects of an innovation on the rest of the social system as network externalities, a term first used by Farrell and Saloner [1985] and Katz and Shapiro [1985]. The term[2] combines the meaning of network effects, commonly used in the communications field [Rohlfs, 1974], and externalities, an economic concept defined as "an effect of a purchase or use decision by one set of parties on others who did not have a choice and whose interests were not taken into account."[3].

Network externalities are generally positive, but may also have negative effects. For example, a larger number of users of one technology may prevent others from switching to another (possibly superior) solution because of the need to cooperate with those who use the current product. Farrell and Saloner [1985, 1986] call this effect "excess inertia", and Strader et al. [2007] call the same phenomenon "cross-impact network externalities". Examples of such cases are numerous, especially in the information technology (IT) domain, and often involve years of established practise before a new product comes along: web browsers, office suites, programming languages, version control systems, and sundry other productivity and development tools. The effect is especially pronounecd in cases where collaboration depends on a shared toolset, and collaborators cannot be forced to migrate, as is the case in the Debian Project.

3.2. Diffusion frameworks

When assessing or comparing diffusions, it helps to focus on sets of comparable, normalised attributes. With such a focus, it is possible to concentrate on salient aspects of a diffusion, and groupings or links between attributes further the clarity and comprehensibility of such a collection.

Such a set of attributes, along with their domains and relation to each other is called a diffusion framework. The Oxford English Dictionary[4] defines a framework as "a supporting or underlying

[2] Leibenstein [1950] coined the term bandwagon effect as "the extent to which the demand for a commodity is increased due to the fact that others are also consuming the same commodity."

[3] `http://economics.about.com/cs/economicsglossary/g/externality.htm` [1 Nov 2007]

[4] `http://www.askoxford.com/dictionaries/compact_oed` [29 Oct 2007]

structure." For the purpose of this text, an analytic framework is a skeleton which allows the characteristics of diffusions to be framed and aligned.[5]

The ideal diffusion framework references the attributes of diffusions whose domains are orthogonal: the value of any one attribute can change without affecting the values of the other fields. Furthermore, orthogonality in attributes reduces redundancy. Orthogonal attributes are sometimes referred to as principal components.[6] Given that diffusion is about communication, and adoption a (human) behaviour, it is unlikely to find orthogonal attributes in all but a few very constrained cases, but orthogonality remains a worthwhile target.

The concept of a model is related to frameworks. The literature uses these words interchangeably. For instance, Venkatesh et al. [2003] provide a comprehensive overview of "models and theories of individual acceptance," all of which are labelled frameworks by their respective authors. I follow the literature and use the two words synonymously.

As early as 1977, Pierce and Delbecq, scholars of organisational theory and management, reported that "during the past two decades a number of theoretical models of organizational innovation have appeared in the literature." In terms of vastness of choice, the situation has not improved today, as plenty of additional frameworks have since appeared. Tornatzky and Klein [1982] seem to have been the first to recognise that most of these frameworks overlap and tried to integrate them into a consistent, unified framework. Few years later, Kwon and Zmud [1987] (again) realised that the models of IS implementation were (still) fragmented and made an attempt to merge them. In recent years, several other meta-frameworks have appeared, but failed to build up on each other and thus fragmenting the space even more [*e.g.* Fichman, 2004, Frambach and Schillewaert, 2002, Wejnert, 2002].

To my knowledge, no framework exists to explicitly structure influences to adoption behaviour in FLOSS projects. Chau and Tam [1997] provide a framework for the diffusion of open systems within businesses, which is the opposite of what I need: it assesses the adoption of open systems within authoritarian structures, not within volunteer-oriented FLOSS projects.

[5] It is easier to compare the performance of two stocks with two data sheets issued by the same bank (and thus using the same presentation framework), than with separate data sheets using different frameworks, which present disjunct sets of characteristics.

[6] The statistical technique "principal component analysis" is also known as "proper orthogonal decomposition."

3.2.1. In search of a framework

Out of the discussion in section 2.4 emerged the need to identify the salient influences to adoption decisions among contributors to the Debian Project. To improve their presentation, I sought a framework that would properly capture those influences, and be compatible with the characteristics of the social system of DCs (see section 2.2), the traits of contributors (see section 2.2.4), the communication media (see section 2.2.7), and the complexity of the project.

I started by investigating over a dozen different diffusion frameworks and found that each framework was specific to a class of social system that could be described with the following four dichotomies:

Communication:	top-down	peer-to-peer
Motivation:	authoritarian	voluntary
Focus/adopting body:	individual	organisation
Adopter interdependencies:	low degree	high degree

A framework suitable for the analysis of diffusions in the Debian Project would have to strongly tend to the right-hand side on all four spectra, because the Debian Project is a voluntary organisation with a strong peer-to-peer communication culture, and high degrees of interdependencies between many (but not all) contributors. None of the frameworks I studied was suitable, as they either focused on individual decisions that had low impact on other adopters, or considered organisation-wide adoption only in authoritarian settings.

Most frameworks encapsulate characteristics of innovations, adopters, the surrounding social system, and other entities in groups, arguing that each group has distinct impact on the adoption behaviour. I find such distinctions problematic, because they suggest independence and may even seduce people into focusing on one group in isolation, when in fact, adoption behaviour is a human behaviour in the end, and as such subject to plentiful, intertwined influences.

The ability of frameworks to structure data and hence increase their usefulness, make instances comparable, and facilitate understanding of the data and relations is at the same time a commonly criticised downside of frameworks: they obscure everything in the data that was not deemed relevant at the time of design or use of the framework. In cognitive science research, the dilemma of deciding what is relevant and what is not is called the "frame problem" [McCarthy and Hayes, 1969], and the term has since been used in other fields as well. Popper

[1994, p. 137f] maintains that a framework is like language and hence incomplete in expressiveness.

In the early stages of my research, I conducted a series of unstructured, casual interviews over Internet relay chat (IRC), as well as in person, with 11 peers. The goal of these interviews was mainly to help me gain an understanding of what I am seeking, not to generate data I would later present as part of my research output. Nevertheless, in these interviews, I identified numerous what I called "factors affecting tool adoption by Debian Developers (DDs)". Seeing a value in the breadth of the collected data, I considered to invite the public to further develop the set of factors, and hence I used a Wiki to structure and present the data.[7].

Before opening the collection up to the public, however, I decided to investigate ways to structure these data, and I considered several existing frameworks in turn. Trying to associate factors with categories, attributes, or components of the respective frameworks, I found myself very frequently making exceptions, squashing factors, or unable to find a suitable place for them. Arguably, the problem could be in the data rather than with the frameworks, but, according to Kearns [1992], an "effort to 'force fit' observed behaviour within the confines of preconceived theoretical frameworks [might suggest] that researchers have apparently fallen short in the theory development phase of their methodology."

None of the frameworks I inspected seemed suitable to frame the list of factors I had collected. This made me question whether these data were useful in the context of my research. Given that I had not kept track of how these data were generated, nor used rigorous data collection techniques, I decided to suspend all efforts in this direction, and to concentrate instead on a rigorous strategy to data acquisition. I kept the Wiki private to not confuse or influence potential later subjects, and postponed the selection of a structural framework until I had better data available.

Applying a framework — or a lens — to a situation enables and restricts at the same time. Kuhn [1970] postulates that one does not abandon a model just by thinking about it. Instead, one abandons a model when a better one has taken its place. Analogously, I chose to move on without a framework and let it emerge from the data, rather than defining a the structure before, not knowing whether the data would fit. After all, I acknowledged that the test data I had gathered through the unstructured interviews were not representative, or gave any indication about the actual influences to adoption behaviour in the Debian Project. My definition of a framework based on speculation over those data would have likely ended up being more restrictive than enabling.

[7] `http://phd.martin-krafft.net/wiki/factors/`, user:phd, password:thesis

I return to this point in the introduction of the results (chapter 7), and I briefly discuss the common obstacles I encountered in the overviews of the three frameworks that were the strongest candidates in section 3.2.2.

3.2.2. An overview of existing frameworks

In the following sections, I briefly present a selection of frameworks for individual and voluntary adoption behaviour. Even though I inspected organisational models, all of them were specific to systems with authoritarian decision-making and thus unsuitable for research in the Debian Project.

The first two are the most widely known and a short discussion in the context of this research is in order. Section 3.2.2.3 introduces a recent framework that looked promising but could not adequately capture the aforementioned test data in a representative and meaningful way. In section 3.2.2.4, I mention other unsuitable but nonetheless notable frameworks. The design and discussion of the framework used to structure the results of the study at hand is left to section 7.1.

3.2.2.1. Rogers' theory of diffusion of innovations

Rogers [1962][8] is often cited as default literature for studies of diffusion of innovations. Following a cooperation with Beal et al. [1957], Rogers conceptualised a generalised model of diffusion. Around 1960, the number of publications on all kinds of diffusions increased sharply after several studies led to the assumption that the innovation diffusion process was independent of communication infrastructure and seemed cross-culturally similar [Deutschmann and Borda, 1962, Rahim, 1961, cited in Rogers [2003]]. While almost all of these studies treated the respective diffusion as separate and did not consider potential insights gained from other diffusions, Rogers [1962] was the first to present a concerted approach towards the study of diffusions. His tome "summarized research findings to date, organized around a general diffusion model, and argued for more standardized ways of adopter categorization and for conceptualizing the diffusion process."

Drawing on over 3,000 studies in sundry domains, Rogers enriched and validated his findings and divided each of these elements into component attributes, characteristics of diffusions that determine (or help to explain) their success. Rather than pretending to be objec-

[8] Rogers' book is currently available as fifth edition: Rogers 2003.

tive truths, the focus is on perceptions and experiences by the potential adopters; according to Rogers [2003, p. 223], it is the "subjective evaluation [which] drives the diffusion process [...]."

Rogers' theory of diffusions is based on numerous generalisations that he formulated during the course of his extensive work with agricultural innovations[9]. Fichman [1992] summarises the most relevant generalisations as follows:

- Innovation characteristics, as perceived by the adopters, determine the rate and pattern of adoption.
- Some adopters are more innovative than others, and these usually have different personal characteristics than the laggards (see section 3.1.2).
- The adoption process can be broken up into a series of 4-5 stages, and adopters are subjected to different influences in each stage (see section 3.1.3).
- Certain individuals can influence others and hence influence adoptions (change agents and opinion leaders, see section 3.1.3).
- The rate of adoption (see section 3.1.2) is an S-shaped curve when plotted over time, as is the cumulative number of adopters.

Through his generalisations, Rogers identified four "elements" of diffusion: (1) Attributes of the innovation; (2) Communication channels; (3) Time; (4) Social system. I briefly summarise these in appendix B.1; an elaborate summary is provided by Raghavan and Chand [1989].

Criticisms of Rogers' framework Rogers' framework has been validated by thousands of diffusion studies. It may thus seem a little far-fetched to speak of criticisms, but in the context of this thesis, the framework is not without limitations. Many of my criticisms may be rooted in its generality: Rogers' framework has been successfully applied in studies of fertility-control methods, political reforms, policy changes, agricultural practises, rural development, technology and IT, health, education, and many more.[10] With such a broad range of applications, the framework has to be a loose fit in any single case.

In the following, I illustrate some of the limitations, specifically with respect to the Debian Project (a constant and complex social system) and the adoption of tools and techniques by volunteers.

[9] These generalisations may be found throughout chapters 5–11 of Rogers [2003].
[10] I refrain from citing representative publications, which would bloat the bibliography.

Static attributes — The fourth element of Rogers' framework captures the social characteristics of the environment, which are likely not to change significantly across diffusions in the same environment. Furthermore, while the attributes of the communication channel may appear differently to any two adopters, they describe the communication patterns within the social system, which are slow to change. With respect to the Debian Project and the study at hand, these attributes will be mostly static, and while this enables diffusions research across social systems, too much of Rogers' framework captures aspects of diffusions that are irrelevant within the same social system

Orthogonality — A significant problem in Rogers' elements of diffusion are overlaps between the attributes of the framework. Even though it is likely impossible to construct a completely orthogonal framework for qualitative data.[11] Rogers' framework lacks orthogonality, which may be a function of its generality.

For instance, the compatibility attribute of an innovation, which references social norms, is almost entirely a subset of "norms of the social system," another attribute in the framework. Similarly, the fourth element includes the communication structure of the social system, while the entire second element is devoted to communication. One may argue that in those examples, one of the overlapping attributes merely expands on the other, providing more details, but then the framework still suffers from the problems related to redundancy of information.

Furthermore, if you compare just pairs of innovation attributes, the overlap is more striking: low levels of observability might cloud the assessment of an innovation's relative advantage, and high trialability might be annulled by high complexity. Similarly, commpatibility includes complexity to a certain degree, for an innovation that is difficult to understand has lower compatibility.

Attribute domains — The attributes of each of the four elements differ in their nature and domains, some of which do not render themselves for comparison. For instance, while each of the characteristics of an innovation can be represented linearly (*e.g.* high *vs.* low degree of relative advantage), the characteristics of the communication process are more complex: both types of communication channel have their merits, and the best adoption rates are exhibited by groups that are neither purely homophilous nor heterophilous, nor does either of them imply higher adoption rates than the other. Moreover, a social system such as the Debian Project has various communication channels with different characteristics,

[11] In statistics, principal component analysis is a rigorous technique to reduce and reorder the dimensionality of a (quantitative) data set, but it only finds optimal solutions, *i.e.* solutions with the least amount of overlap between those dimensions. It is doubtful that one can come close to such results with qualitative data.

and innovation diffusion usually spans multiple channels, which the framework cannot represent.

Variance vs. process theory — While the first and fourth elements of Rogers' framework seem like candidates for analysis by variance theory, and the third element renders itself more for process research [*cf.* Mohr, 1982], it is unclear which approach to take for the second element; communication is a process [Dance and Larson, 1976], but Rogers identifies strongly interdependent variables of the communication channel in his second element of diffusion, rather than focusing on the process of communication over time.

A solution might be to analyse changes in the characteristics between subsequent phases of the process model. When considering this way forward, it quickly becomes evident that innovation and adoption are processes, but Rogers' framework has no provisions to capture change across time beyond his identification of stages. These stages, however, are individualistic and do not take into account changes to the social system in which a diffusion is taking place.

Simplistic — Rogers generalised only across diffusions in which little or no specialised knowledge about the innovations was required prior to the adoption [Robertson and Gatignon, 1986]. In addition, those innovations' consequences did not affect other members of the social system (at least not directly), and their potential adopters were able to decide independently for or against the innovations, according to Fichman [1992], who designed a model for choosing a framework according to the complexity of the diffusion (see section 3.2.2.4). In his words, "the opportunities to apply classical diffusion 'as is' may be rare indeed."

Outdated — Furthermore, Rogers' framework does not include the degree of interconnectedness of communication networks, or the concept of geographical proximity, both of which influence the speed at which information can spread [Allen, 1970, Allen and Henn, 2006, Haegerstrand, 1967]. In Rogers [2003, p. 216]' own words, "the world in which we live today is a different one than that of sixty years ago, when study of the diffusion process began." By extension, his model, which is but a decade younger, may not be adequate in the context of globally-spread diffusion of innovations any more.

Rogers [2003] is a seminal work, and much of my research and write-up builds upon it. In fact, the time element, which defines the five-stage innovation-decision process is a building block of the stage model for adoption behaviour I present in section 7.1. But his framework is not suitable as a lens for adoption behaviour in a voluntary context, where individual decisions often have considerable effects on others, innovations are continuous in nature (tools/techniques are

not finished when they get adopted), and the separation between individual and organisational adoption is not clear-cut.

3.2.2.2. The Technology Acceptance Model

The Technology Acceptance Model (TAM) [Davis, 1986, Davis et al., 1989] builds on the Theory of Reasoned Action (TRA) [Ajzen and Fishbein, 1980, Fishbein and Ajzen, 1975], which in turn is a refined version of the Fishbein model [Fishbein, 1967]. The TRA is a model about the relation between attitude and behavioural intention, specifically as applied to "objects" (which could be immaterial, such as a process): an individual may hold beliefs about a given object and as a result, form an attitude towards the object. An attitude may lead to intentions with respect to the object, and those intentions result in behaviour. Behaviour can be positive and negative, as long as it is explicit: said individual may start using the given object, or may refuse it. This decision may be influenced by social norms, another component of the TRA [Fishbein and Ajzen, 1975, p. 301ff], albeit one that is only minimally understood [*ibid.*, p. 304]. Curiously, the component of social norms of the TRA has not been included in the TAM.

The TAM has been referred to as "the most influential and commonly employed theory for describing an individual's acceptance of information systems" [Lee et al., 2003]. It has emerged as a powerful model [Pijpers, 2001], robust and parsimonious [Venkatesh and Davis, 2000], and has been validated through a replication study, covering non-computer technologies, as well as computer software [Adams et al., 1992].

Many extensions to the TAM have been proposed: Chau [1996] distinguishes between short-term and long-term usefulness and found that while both have a net-positive effect on an individual's likelihood to accept, the short-term perception was most significant. Furthermore, Chau found that perceived ease-of-use had "no significant, direct relationship" with "behavioral intention to use a technology." Wixom and Todd [2005] integrated user satisfaction measures with the TAM. Bagozzi et al. [1992] focus on the importance of learning in technology acceptance.

Venkatesh and Davis [2000] defined the TAM2 as an extension to the TAM, adding theoretical constructs spanning social influence and cognitive instrumental processes. Venkatesh et al. [2003] reviewed eight acceptance models, including the TRA, the TAM, and the TAM2, and consolidated their salient features into the Unified Theory of Acceptance and Use of Technology (UTAUT). The UTAUT proposes four key constructs, which influence an individual's behavioural intention (performance expectancy, effort expectancy, social influence, and facilitating conditions), and four

mediators (gender, age, experience, and voluntariness of use), each of which has an influence on the weight of each of the four constructs. The model outperformed each of the eight models on which it was based [Venkatesh et al., 2003]. Garfield [2005] successfully used UTAUT to study the acceptance of ubiquitous computing. Venkatesh and Bala [TBA] have begun work on the TAM3 to extend the TAM2 (and presumably UTAUT) with a "focus on interventions", and greater granularity in the attributes.[12]

Criticisms The TAM was revolutionary at the time of its conception, and it continues to influence IS research. Many extensions to the TAM have been proposed — I only mentioned a few of them above — but the effect has been even further fragmentation of the field. Bagozzi [2007] exemplifies the problem with the case of UTAUT, which he describes as "a model with 41 independent variables for predicting intentions and at least eight independent variables for predicting behavior," when what we need is " a unified theory about how the many splinters of knowledge cohere and explain decision making."

Bagozzi [2007] presents further, substantiated criticism of the TAM, which includes most of my own criticisms, and adds other convincing ones. Instead of duplicating his work, I limit myself to a short treatment of my own criticisms.

The TAM is based only on two features: perceived ease-of-use and perceived usefulness, and the latter is influenced by the former "because, other things being equal, the easier the system is to use the more useful it can be" [Venkatesh and Davis, 2000]. This parsimony is both, a strength and a weakness: it makes the model universal, easy to understand, and conducive to use. At the same time, however, it is too universal to be used in analysis or prediction across a variety of settings. Ajzen and Fishbein [1980, p. 4] describe TAM as "very general, designed to explain virtually any human behavior", although the alleged explanatory ability may be of limited use. But as soon as the simplicity is traded for more granular attributes in attempts to make the model fit better with any particular use-case, we return to the plethora of fragmenting derivatives, none of which has received enough scrutiny to assume a bolder position than the others.

Furthermore, there are grave problems in the TAM's legacy. It is a model that builds only on attitude and behaviour and inherits from the TRA the explicit requirement for behaviour to take place in the form of a decision to accept or to reject [Fishbein and Ajzen, 1975]. This causal relation is logical in the context of the model, and thus not falsifiable [Popper, 1972, Silva, 2007].

[12] `http://www.vvenkatesh.com/IT/organizations/Theoretical_Models.asp` [2 Sep 2009]

The TAM fails to take into account the actual implementation of a decision, but considers a decision to act as final. It portrays technology acceptance as a state model, where subjects remain in a dissonant state until perceived usefulness and easy-of-use align and the subject is catapulted into behaviour, when in fact, technology acceptance is a process constituted by goal striving [Bagozzi and Dholakai, 1999].

The model furthermore cannot explain situations in which the perceived attributes of an innovation speak in favour, but the subject chooses to delay a decision, due to lack of resources, emotional reasons, or any other number of influences which may create uncertainty, despite favourable attributes according to the TAM.

Lastly, of particular relevance in the context of this study, the TAM focuses only on individuals and does not consider any form of group, cultural, or social aspects of technology acceptance. This alone makes it unsuitable for use to study behaviour of individuals within the boundaries of a community such as is the case in the Debian Project.

3.2.2.3. Wejnert's integrated model of innovation diffusion

With a similar claim to Rogers [1962], that diffusions are mostly analysed in isolation from the insights of the others, Wejnert [2002] designed a conceptual framework by merging diffusion variables identified in other studies into a coherent structure. Her framework has three major components with various constituent variables, shown in table 3.2 on the current page.

A slightly more thorough description of the individual variables may be found in appendix B.2.

Characteristics of		Environmental
innovations	**innovators**	**context**
public *vs.* private consequences	societal entity	geographical setting
benefits *vs.* costs	familiarity with the innovation	societal culture
	status characteristics	political conditions
	socioeconomic characteristics	global uniformity
	position in social networks	
	personal characteristics	

Table 3.2.: *The three major components of the integrated model of innovation by Wejnert [2002], and the corresponding variables.*

Evaluation Wejnert's framework is relatively young, which may be the reason that I have only found instances of the author herself using it, *e.g.* to assess the diffusion of democracy over the past two centuries [Wejnert, 2005].

Other than Rogers' framework (see section 3.2.2.1), Wejnert does not consider the process of diffusion, but only characteristics of the innovation, the potential adopters, and the environmental context of a diffusion. It is helpful that she distinguishes between characteristics of the individual and the collective, but most of her variables overlap and lack orthogonality, similar to what I have shown for Rogers' framework earlier (see section 3.2.2.1). For instance, members with high socio-economic characteristics are likely to find themselves in central positions within the social network, thus boosting their social status, and yet those are separate variables.

Wejnert does not distinguish between interpersonal and one-way mass communication. It seems, however, that Rogers' second element, communication, might be a worthwhile addition to Wejnert's framework. Furthermore, Wejnert does not really distinguish between individual, collective, voluntary, and authoritarian decisions, another important candidate from Rogers' framework.

She regards consequences of an adoption to be a feature of an innovation, rather than the social system in which the diffusion is taking place, as does Rogers. Wejnert makes interconnectedness, network structure, proximity and level of communication explicit, and her model caters for globally-spread groups.

Nevertheless, the framework does not seem to be able to accommodate "soft", subjective characteristics of an innovation, including those pertaining to preference and elegance, and falls short on all aspects related to communication, as discussed above.

3.2.2.4. Other notable frameworks

Lin and Zaltman [1973] defined 20 "dimensions of innovations", which looked worthwhile to consider in the short glimpse offered by Kearns [1992]. However, I could not find another work that made use of these dimensions, suggesting that they may not have been adequate.

Downs and Mohr [1976] differentiate between primary and secondary attributes. The first are direct characteristics of an innovation, whereas the secondary attributes only exist in the subjective perception of an innovation. Their claim that diffusion research had thus far been inconsistent, because existing models do not make this distinction and focus on subjective per-

ceptions of attributes only, is weak, and it remains questionable whether truly objective attributes of an innovation exist [Tornatzky and Klein, 1982]. I did not pursue this line further.

Tornatzky and Klein [1982] themselves list a large set of attributes they gleaned from analysing 75 studies of the effect of innovation attributes on adoption and implementation decisions, which extended the attributes of innovation by Rogers [2003], but which only focused on the innovation *per se*, not other elements of diffusion.

Fichman [1992] finds that "IT diffusion research can diverge from classical diffusion assumptions due to characteristics of the technology (user interdependencies , knowledge barriers) or the locus of adoption (individual versus organizational). He proposes a framework which maps classes of technology against locus of adoption to obtain four IT adoption contexts: type 1 technologies come with low knowledge burden and low user interdependencies, while type 2 technologies require high levels of knowledges and affect other users. The locus of adoption is either the individual or an organisation. For each of the four permutations, Fichman points at attributes from other frameworks. For instance, individual type 1 diffusions he leaves to Rogers, while he refers to *e.g.* Zmud [1982, 1984] for organisational type 2 diffusions.

Although he does not design a framework by itself, Fichman splits the field of IT innovation diffusions along two prominent axes and encourages the use of *different* frameworks for each permutation. Unfortunately, I could not find a study making use of his work. However, the idea to combine different frameworks stuck and influenced the design of the presentation framework I outline in section 7.1.

3.2.3. A framework for influences to adoption behaviour in the Debian Project

As stated in section 3.2.1, I found no diffusion model that was suitable for the task of framing the influences to adoption decisions by contributors to the Debian Project. Instead of picking or designing one, I proceeded to collect the data (see part II) and let a structure emerge from them, thereby avoiding to "force-fit" data to preconceived criteria [Kearns, 1992].

The structure which emerged is based on the two stage models presented in section 3.1.3: Rogers' individual innovation-decision process (see appendix B.1.3), and the IS implementation process proposed by Kwon and Zmud [1987]. The combined model is described in section 7.1 and depicted on page 141.

Instead of focusing on attributes of innovations, adopters, or the surrounding social system (variance research), I present the influences to adopter decisions according to the point in time when they appear most relevant (process research). At the same time, I explicitly label the framework as a means of presentation, rather than a structure that can be used to explain or predict. The explanatory (and even predictive) power lie in the data, not the framework, and I used the stage model to outline the data in a way that make them more accessible and thus more useful.

To make it possible for readers to consider the results presented in chapter 7 in isolation, and still make sense of the structure in which the resulting influences to adoption decisions are presented, I chose to postpone explaining the development of the framework to section 7.1.

3.3. Related studies

Fichman [1992], Kwon and Zmud [1987], Tornatzky and Klein [1982], and Swanson [1994] provide an extensive overview of diffusion research in the IS domain. The following concentrates on studies performed in the FLOSS context.

Innovation diffusion in the FLOSS domain has not received much attention by scholars. Oezbek and Prechelt [2007] proposed to study the introduction of innovation in FLOSS projects. Oezbek [TBA] studies innovation diffusion across a number of FLOSS projects using techniques from Grounded Theory [*e.g.* Strauss and Corbin, 1998] as part of his Ph.D. research. He has made a number of technical reports available to me [Oezbek, 2008, 2009], but we agreed not to intensify exchange of information at this stage to maintain independence of our works.

Chesbrough [2003] approaches "open-sourcing" of the process of innovating in a business context — the strategy of a profit-oriented institution to encourage its consumers/users to engage in the process of innovation, and enabling the business to incorporate this input. West and Gallagher [2005] evaluate FLOSS projects from a management perspective and seek to investigate three challenges: (1) maximising the use of internal innovation; (2) incorporating external innovation into the firm; (3) motivating a supply of such external innovation to support the firm. Späth et al. [ming] also consider the idea of open innovation from the perspective of commercial entities, but do not look at adoption behaviour.

I am not aware of any study which studies the adoption behaviour of developers of a FLOSS project.

Part II.

Research approach

Chapter 4

Introduction

The discussion on Debian in chapter 2 highlighted the need for improved tools and techniques in the Debian Project, and the difficulty of diffusing such tools among the volunteers. I defined my goal to be the identification of influences to Debian Contributor (DC) adoption decisions, as knowledge of such influences could help increase the adoption rate of improved approaches in the Debian Project, and help the project scale better.

In chapter 3, I surveyed the field of adoption and diffusion research, but found no existing framework which could be used to structure the influences or guide their search. Therefore, I chose to continue without such a structure, but to let it emerge out of the data.

I concentrated my research on Debian and did not look to other projects. Studying a single organisation may not produce statistically representative data, but according to Mintzberg [1979], the adequacy of such an approach depends on the goals of the study, and a small sample has often proved superior to the statistically relevant one. I opted for the "sample of one", acknowledging the cultural, technical, organisational, and quantitative differences between FLOSS projects, because I wanted to study in-depth, rather than in-breadth. Figure 4.1 on the next page illustrates the research approach I took.

In the following, I would like to describe the metatheoretical and methodolical assumptions underlying my research, and expand on my research objectives, which were introduced in section 1.2. Chapter 5 then introduces the Delphi method, presents the reasons for its suitability to this research, and describes the design considerations I made. I document the

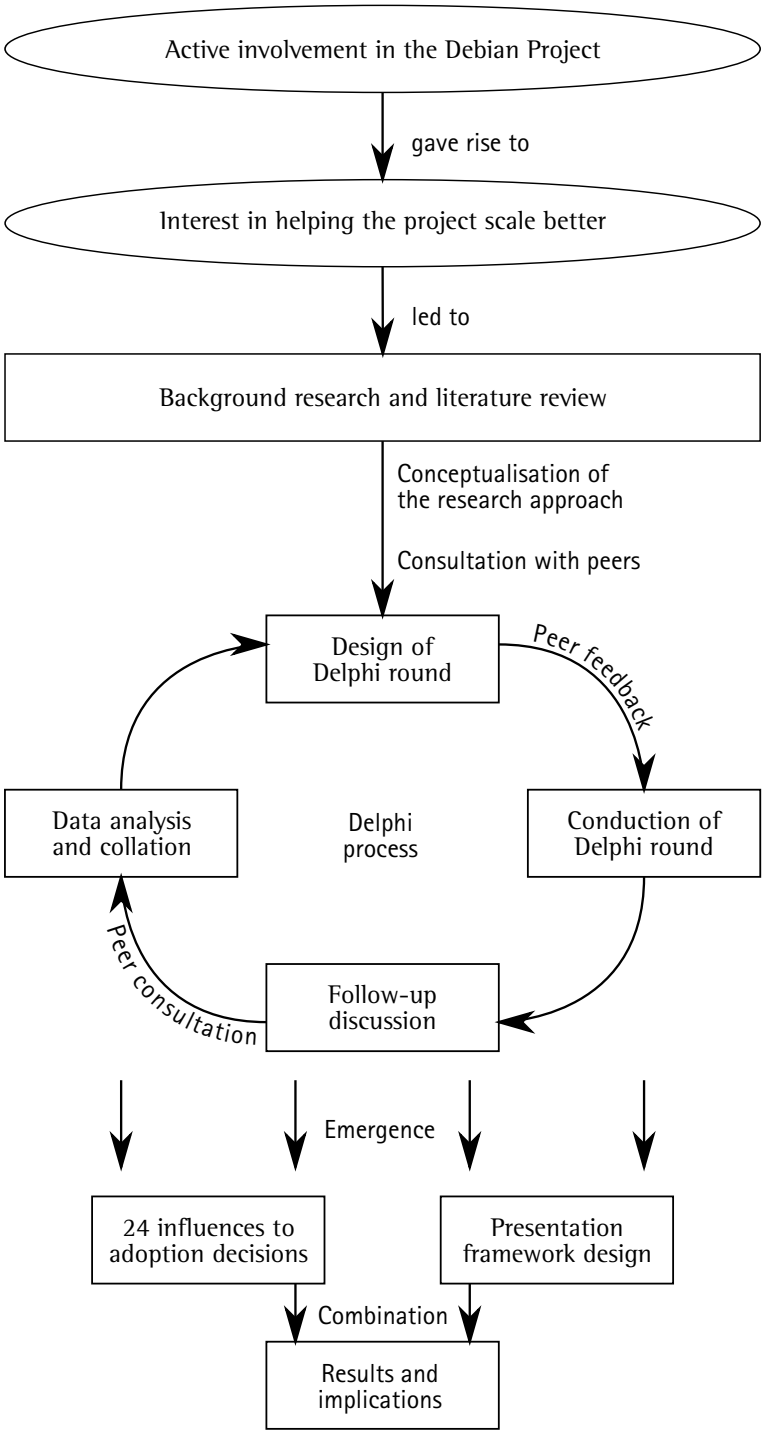

Figure 4.1.: *A schematic illustration of my research approach.*

details of the actual study, as well as the decisions I had to make throughout in chapter 6.

4.1. Philosophical considerations

Any form of research is guided and framed by a number of metatheoretical assumptions about the nature of existence (ontology), possibility of knowledge (epistemology), and truth, as well as methodological considerations including the research object, rigour, and validity. With their Principle of Dialogical Reasoning, Klein and Myers [1999] recommend to be explicitly aware and transparent about the intellectual basis underlying one's work, because preconceptions and prejudices necessarily colour the result and make all research subjective. Results can therefore only be meaningful if the context in which they have been found or generated is also known.

I would like to establish the context of my work in the following paragraphs. In summary, my research involved a *qualitative*, *exploratory* approach, and knowledge was *inductively* created through observations while immersed *in the field*.

These assumptions are common to interpretive research [*e.g.* Boland, 1985], which appears to be in stark contrast to the previously dominant positivist view, and has been denunciated as far as being called "non-scientific" [*cf.* Popper, 1972]. The literature has been burdened by years of heated debates over the acceptability of interpretivism as a rigorous, scientific strategy [*cf.* Hirschheim, 1985], but interpretive research has since emerged as a respected alternative to positivism [Walsham, 1995].

In the last two decades, the observation that positivism and interpretivism are not as separate as previously believed and upheld, and that much of their differences derive from disjunct terminologies, have led *e.g.* Lee [1991] to attempt to unify the two strategies. Weber [2004a] suggests to go beyond the distinction, for he "believes that deep similarities rather than deep differences underlie [the two schools]". Rather than to rehash the debate, which has been summarised numerous times before [*e.g.* Fitzgerald, 1997, Lee, 1991], I would like to accept positivism and interpretivism as inextricable, as two sides of a coin to be inserted into the slot that is constructivism.

I am a constructivist with respect to the research performed and described in this work. I expend no thought on the possibility of a higher influence (like the occasionalist might), nor do I deny chaos (like the determinist would), as any such considerations would distract from

the possibility of progress. I accept constructivism as a simplification of the world, in which causality is fundamental but secondary. Specifically, while building upon the theory that everything is constructed, I do not deny that the constructions are only one way of interpreting the whole.

Interpretivism and positivism are two sides of an ideology born out of motivation: both schools seek to further understanding of the world, and at certain points in time, both will come to accept (and affirm) the *status quo*. Interpretivists may be more prone to accept that everything is relative, and that cause and effect are inseparable, and their affirmations are less definitive, smaller in scope, and more flexible. However, both schools build upon what they accept and affirm.

I affirm the Debian Project *as well as my direct affiliation therewith* as basis of my research. One might label that positive, but I am not a mere observer: I am myself part of the research object. I interpret through interaction, inquiry, and selection of data. While this is expedient for the depth and relevance of my research, it makes it all the more important for me to meticulously document my methods, the problems I encountered, and grant access to the data to anyone wanting to verify my findings.

Instead of regarding myself as a participant of science, as one of many striving towards some common goal of understanding, I actively explore my immediate environment. This realisation has led me to write in an active voice and from the first person perspective.

As exemplified before, sufficient middle ground exists between the extremes of positivism and interpretivism. Sandberg [published by Weber, 2004a] identified seven such alleged extremes, and I would like to position my research on each gamut. I shall end with a brief look at the eight "research dichotomies" by Fitzgerald [1997, p. 140].

4.1.1. Positivism vs. interpretivism

Table 4.1 on the facing page shows the seven alleged characteristic differences between positivism and interpretivism, according to Sandberg, published by Weber [2004a]. In the following, I will argue why my research tends towards the interpretive side. Weber, who appears to be a constructionist himself, has brought the two schools together and reduced the previously fundamental differences to varying perspectives upon the same goal, which is to further individual and shared understanding of a subject matter. I merely want to put my research into context.

Metatheoretical assumptions about	Positivism	Interpretivism
Ontology	Person (researcher) and reality are separate.	Person (researcher) and reality are inseparable (life-world).
Epistemology	Objective reality exists beyond the human mind.	Knowledge of the world is intentionally constituted through a person's lived experience.
Research Object	Research object has inherent qualities that exist independently of the researcher.	Research object is interpreted in light of meaning structure of person's (researcher's) lived experience.
Method	Statistics, content analysis.	Hermeneutics, phenomenology, *etc.*
Theory of Truth	Correspondence theory of truth: one-to-one mapping between research statements and reality.	Truth as intentional fulfillment: interpretations of research object match lived experience of object.
Validity	Certainty: data truly measures reality.	Defensible knowledge claims.
Reliability	Replicability: research results can be reproduced.	Interpretive awareness: researchers recognize and address implications of their subjectivity.

Table 4.1.: *Alleged characteristic differences between positivism and interpretivism [Sandberg, published by Weber [2004a]]*

Ontology — Weber [2004a] finds the alleged ontological differences between the two strategies to be vacuous, because even interpretivists must be able to distinguish between perceptions and reality. Subscribers of either strategy are working to enhance their understanding of whatever world they are studying, and while both are aware of bias and preconceptions, as well as strengths and weaknesses of their approaches, the differences lie in the willingness to accept that what is perceived may be different from what is real, and that perceptions do not necessarily stem from realities. While positivists use rigour to establish and maintain objectivity within their paradigms, and forego the analysis of subjectivity, interpretivists focus on people and the physical and social artefacts they create, requiring more explicit documentation of the underlying beliefs and assumptions.

As the Debian Project is an association of people, and my study focuses on influences to human behaviour, an objective reality cannot be maintained, nor any paradigmatic basis identified. Within the project, one may find hard rules and established processes, but these are the creation of people, and subject to interpretation. To understand people and their

behaviours, the researcher has to immerse in the culture, because the external observer may otherwise *e.g.* only perceive as irrational a behaviour that is perfectly in line with social norms. My involvement with the Debian Project also means that any attempt to separate myself (as observer) from the observants would be absurd.

Epistemology — I think that knowledge can only be objective within an axiomatic system, which renders objective knowledge unfalsifiable, and thus, according to Popper [1972], any theory building upon objective knowledge must be unscientific. Axioms are constructions, and therefore objective knowledge cannot exist outside a constructed world. Traditionally, positivists would work within such a world until a paradigm shift occured and a differently constructed world became standard [*cf.* Kuhn, 1970]; the existence of such paradigm shifts has since become commonly understood to the point where they are anticipated. As Weber observes, positivists "recognize the inherent limitations of the knowledge they seek to build, [… and they] are acutely aware of the ephemeral nature of the knowledge they construct." The traditional positivist seeks to research within well-defined borders, while the interpretivist is more prone to step outside. This alone casts doubt over the possibility of pure positivist knowledge, simply because it would be naïve.

Weber describes research as a "continuous journey to find improved ways to understand reality" and that any artefacts built in the process are socially constructed. There is no single, objective behaviour in the Debian Project. Concepts and ideas within the Debian Project are abstract, and behaviour mostly a function of one's involvement in the project. Obviously, psychological personality traits (which exist outside of the individual involvement with Debian) play their roles in partitioning adopters into groups, and adoption behaviour by extension, but psychology and sociology themselves (need to) employ constructed artefacts to explain human behaviour [Berger and Luckmann, 1966].

Research object — With reference to Heisenberg's Uncertainty Principle [1927], Weber refutes any possibility of a clear separation between researcher and researched object. He concludes that "both positivists and interpretivists understand that they, the research processes they use, and the objects they research are inextricably related."

This applies very much to my own research. First, I designed my approach specifically for use in a FLOSS context, and used my experience with Debian to fit it tightly to the research object. Second, I was acquainted with most participants of my research prior to the study, and their participation as well as my conduct necessarily built on these social ties. In section 8.4.1, I reflect upon the effects of these close relationships.

Research method — While the methods used by the two schools of positivism and interpretivism are classically split into what may be called hard and soft approaches, Weber claims that positivists have learnt from interpretivists, and *vice versa*, and that an approach such as the case study, which is traditionally regarded as an interpretive approach, has also been used by positivists, just as *e.g.* ethnographers have employed rigorous statistical means to find patterns in empirical data.

Studies of both types have been performed in the FLOSS domain, and Debian as well, and the choice of method must depend on the results one seeks. While a positivist stance would have made sense in determining *e.g.* average adoption cycle length, or the spread across adopter categories within the community, a behavioural study such as mine must necessarily lean towards the interpretive side and employ methods geared towards answering questions of "why", rather than questions of "how" and "what".[1]

Truth — Weber argues that building a positive theory and later verifying it is not too different from the hermeneutic circle, in which incongruencies between perception and understanding lead to iteratively refined interpretations. However, by the constructivist theory of truth, truth is perceived as a function of experience and norms, and to understand truth means to understand these experiences and norms.

Rather than using questionnaire responses to build a theory to be verified later, which leaves little room for exploration, I sought the hermeneutic, iterative approach. I had no basis on which to argue for the appropriateness of my preconceptions of the results I was seeking, which necessarily influenced my inquiries (and the design of any questionnaire). Instead, I wanted to place less weight on these preconceptions at the start of the study, than a theory-build-and-verify approach would have allowed me to do.

One may suspect in this view a lack of power of judgement, and that I leave truth to chance and other influences. But on the contrary, this is precisely the strength of the approach: through cyclical and iterative, exploratory construction of a theory, I diminish the influence of my own preconceptions, reducing the need to later verify the theory.

Validity — The extent to which results derive from observations is measured by the notion of validity. Here, Weber argues that positivists and interpretivists alike are simply trying to convince colleagues of the defensibility of their knowledge claims, not of the claims *per se*. Different notions of validity exist because the defensibility of knowledge depends on

[1] The attentive reader will have noticed that my research question starts with the word "what", but it is still a why-type question: seeking the influences that affect behaviour is one way to answer the underlying question why people behave in the way they do.

the research method, and thus positivist validity traditionally builds on "hard" criteria, whereas interpretivists use "soft" concepts, such as credibility or transferability, which are less formally defined.

My study is almost entirely based on "soft" methods, and therefore, "soft" notions of validity are the forefront of my concerns. I ensured validity throughout my research by keeping meticulous records of all steps, and plan to publish all data, which form the basis for the defensibility of all claims.

Reliability — The last alleged difference between positivism and interpretivism is the idea of reliability, and Weber also reduces that to the choice of research methods, saying that positivists' methods are well-defined and routinised, whereas interpretive research is more *ad hoc* and improvised. This is the reason why interpretivists make their assumptions and preconceptions more explicit, as I already stated in the above discussion on ontology, and which serves no other purpose than to establish the reliability of a study. This was the other main reason for my meticulous data-keeping throughout my research.

4.1.2. Research dichotomies

The discussion in section 4.1.1 has established my position as an interpretivist, which puts me on the right-hand side when it comes to the first in the list of eight research dichotomies identified by [Fitzgerald, 1997, p. 140], shown in table 4.2 on the next page. My position on the other spectra similarly tends towards the right column. In fact, not much needs to be said in addition.

For instance, the discussion on research methods in the previous section implies that I chose a qualitative and exploratory research approach, because I wanted to explore an unknown space and generate results that answered a "why" question, rather than a what/how-type question. "Induction" and "field" similarly follow out of the previous discussions on research object and truth, because I was immersed in the social system I was researching. The data are "idiographic", rather than "nomothetic". My role as "Emic/Insider/Subjective" furthermore follows from my involvement in the Debian Project, which I have articulated numerous times.

The work I put forth is hence clearly interpretive, and my approach qualitative. I employed rigour where possible, but mainly concentrated on relevance of the data. My conduct reflects my objectives, articulated in the next sections, which may be summarised as exploring behaviour through research that took place as close as possible to the subject.

Positivist Belief that world conforms to fixed laws of causation. Complexity can be tackled by reductionism. Emphasis on objectivity, measurement and repeatability.	**Interpretivist** No universal truth. Understand & interpret from researcher's own frame of reference. Uncommitted neutrality impossible. Realism of context important
Quantitative Use of mathematical & statistical techniques to identify facts and causal relationships. Samples can be larger & more representative. Results can be generalised to larger populations within known limits of error	**Qualitative** Determining what things exist rather than how many there are. Thick description. Less structured & more responsive to needs & nature of research situation
Confirmatory Concerned with hypothesis testing & theory verification. Tends to follow positivist, quantitative modes of research	**Exploratory** Concerned with discovering patterns in research data, & to explain/understand them. Lays basic descriptive foundation. May lead to generation of hypotheses
Deduction Uses general results to ascribe properties to specific instances. Associated with theory verification/falsification & hypothesis testing	**Induction** Specific instances used to arrive at overall generalisations. Criticised by many philosophers of science (e.g. Popper, Medawar), but plays an important role in theory/hypothesis conception.
Laboratory Precise measurement & control of variables, but at expense of naturalness of situation, since real-world intensity & variation may not be achievable	**Field** Emphasis on realism of context in natural situation, but precision in control of variables & behaviour measurement cannot be achieved
Nomothetic Group-centred perspective using controlled environments & quantitative methods to establish general laws	**Idiographic** Individual-centred perspective which uses naturalistic contexts & qualitative methods to recognise unique experience of the subject
Etic/Outsider/Objective Origins in anthropology. Research orientation of outside researcher who is seen as objective and the appropriate analyst of research	**Emic/Insider/Subjective** Origins in anthropology. Research orientation centred on native/insider's view, with the latter viewed as the best judge of adequacy of research
Rigour Research characterised by hypothetico-deductive testing according to the positivist paradigm, with emphasis on tight experimental control and quantitative techniques	**Relevance** Validity of actual research question & its relevance to practice vital, rather than constraining the focus to that researchable by 'rigorous' methods

Table 4.2.: *Summary of research dichotomies [Fitzgerald, 1997, p. 140, columns swapped for compatibility with table 4.1 on page 65]*

4.2. Research objectives

In section 1.2, I posed my research question: What are the influences that shape DCs' adoptions of innovations? I identified the three main objectives of this study, which follow out of the discussion on chapters 2 and 3:

1. to determine the salient influences to DCs' tool/technique adoption or rejection decisions;
2. to propose a terminology of labels for these influences, and present them in a meaningful, accessible way;
3. to crystallise a number of implications for practice from these influences, to help increase the rate of diffusion of improved tools and techniques in the Debian Project, to foster competition and progress, and to enable the project to scale better with its growth.

4.3. Secondary objectives

I also plan to achieve the following sub-objectives:

1. to provide stepping stones for future work, by making all data available under a Free licence, speculating research ideas and identifying niches for further work;
2. to further the use of the Delphi method in the FLOSS environment, by highlighting its applicability in the FLOSS context, meticulously documenting the design of my approach, and analysing the performance.

I shall briefly detail these in the following two sections.

4.3.1. Sub-objective 1: Data from the Delphi study

The first sub-objective serves two purposes:

First, this being a qualitative study, and one that involved a large amount of data, it is possible that notable statements went ignored, despite my best efforts to prevent that. By making all data available, I want to ensure that any such holes do not remain unnoticed forever.

Furthermore, it was impossible to avoid all bias throughout the study, and at times, I converted abridged or incomplete statemens by panellists into proper quotes (see section 7.2). Where I was

aware of bias, I explicitly stated so, and section 8.3.1 analyses bias retrospectively. I included the message-IDs of the e-mails from which I lifted quotes with the statements to facilitate the verification. The availability of all data ensures that any instance of bias, whether explicitly noted, or one of which I was not aware at the time, can be identified as such, properly analysed, and taken into account when using my results. Moreover, by publishing the underlying data, I work against the chances of any instances of bias to remain unnoticed.

Second, the data are the creative work of 200 or more hours by experienced and knowledgeable contributors of the Debian Project. They were collected to answer the aforementioned research question, but their breadth and depth may offer valuable insights to future researchers in the domain. As a study in the context of a project as dedicated to the ideals of freedom and openness as the Debian Project, seeking permission from all participants to make the data available in a usable form at the end of my study seemed another logical way to contribute to the project, and to enable future work.

4.3.2. Sub-objective 2: The Delphi method in the FLOSS context

The reasons for my second sub-objective are presented in section 5.2 and may be summarised as follows: the Delphi method fills a gap in the communication avenues in use by FLOSS projects nowadays, because it is a structured means to counter the problems inherent in instantaneous global communication: volume, and the inevitable forking of discussions into disjunct threads of conversation, which degrade the usefulness of any discourse after a short time, and which make later information retrieval unnecessarily difficult and ineffective. Furthermore, the Delphi method facilitates all involved to focus on content, rather than deal with preconceptions about other people and their opinions. My study demonstrates the applicability of the Delphi method in the context of a FLOSS project.

When I started investigating the Delphi method, I found hundreds of applications of the approach, all subtly different from each other, but only a handful studies reported on the design considerations (the Delphi method is not a cookbook approach, but a collection of tools), and the details of the approach. Furthermore, not a single application of the Delphi method in the FLOSS context had been documented. I spent a considerable amount of time crawling through existing texts, even exchanging e-mails with authors of reports, to piece together a coherent Delphi design that was suitable in the novel context of the study at hand.

It is interesting to note that while I was investigating the Delphi approach, a group of DCs instituted the Debian Enhancement Proposals (DEP) idea as a way to track discussions in a central location (a web page, or similar), with the goal to build consensus. In a personal consultation with a peer leading up to the Delphi study I conducted, he identified clear parallels between the idea behind DEP and the Delphi approach:

> *Almost all of your reasons for use of the Delphi method were the reasons (or problems if you want) we were trying to solve with DEPs. The one key difference is that we cannot propose (whatever stripped down version of) the Delphi method to structure relevant Debian discussion, for a reason of democracy.*[2]
>
> *With the Delphi method as a first step you choose the board, and its composition should satisfy certain criteria; in Debian, we have, of course, free speech and we must let anybody take part in any discussion. This chain of thought made me wonder whether there are some more interesting aspects of the relationship between Delphi's and really democratic projects which need to be investigated,* i.e. *trying to think about changes to the method that enable communities themselves (rather than* researches studying communities, *as your case) to use the method for their discussions.*
>
> — Stefano Zacchiroli, personal communication
> <20081018124157.GA26653@usha.takhisis.invalid>

If the Delphi method should be applied in the context of a FLOSS project — and by that I mean research as well as day-to-day development and consensus finding — then its users cannot be expected to repeat all this work. With the level of detail presented in this thesis, I hope to entice the use of the Delphi method in FLOSS environments, and provide the necessary details to be able to design an application in considerable time.

Following completion of this thesis, I would like to collect relevant resources into a website[3] to support future use of the method. I have also created the `#delphi` Internet relay chat (IRC) channel on `irc.debian.org`, and plan to set up a discussion mailing list.

[2] "Here I'm deliberately ignoring a lot of problems, like the fact that overall the Delphi method *can* require more energy for the community than our ordinary discussion mechanisms. I'm doing that just because I think the important point is another one" [his footnote].

[3] I have tentatively reserved `http://delphi.debian.net` for this purpose, but will probably seek a neutral (non-Debian) address at some point, and redirect the `debian.net` URL accordingly.

Chapter 5

The Delphi method

The Delphi method was developed at the RAND Corporation in the 1940s as a way of finding "the most reliable consensus of opinion of a group of experts" [Dalkey and Helmer, 1963]. The original Delphi study sought to investigate the impact on technology on warfare [Dalkey, 1972] and was exploratory in nature.

According to Weaver [1971], "Delphi operates on the principle that several heads are better than one in making subjective conjectures about the future, [...] and that experts will make conjectures based upon rational judgement rather than merely guessing [...]." More recently, Surowiecki [2004] collected numerous case studies and anecdotes, based on the argument that crowds have more wisdom than the individual because, given enough diversity in the members of the crowd, their averaged statements converge closer to the "right answer" than an individual's response.

Even though Linstone and Turoff [1975a] claim that the question of whether the crowds are wiser than the few has not received the attention it deserves, and I have not sighted any study going down this path, the Delphi method is gaining popularity. It is an increasingly used forecasting technique for programme planning, resource allocation, policy design, needs assessment, and prediction of the occurrence of events in research domains as diverse as medicine, health services, technology, insurance & finance, sociology, economics, and politics. Yet, I did not find an application of the Delphi method in a FLOSS-related context.

Yet, the Delphi method has been subject to heated debates over methodological aspects (see section 5.4). In recent years, however, evidence of such debates has been thin, suggesting that

the research community is accepting the method for its merits, similar to other qualitative research techniques [*cf.* Silverman, 2005, Wolcott, 2001].

In this section, I introduce the Delphi method and some of its variants, discuss its applications, strengths, criticisms that have been brought against it, and finally outline the considerations that have led me to adopt and design the method for my research.

5.1. The Delphi process

A Delphi study "may be characterized as a method for structuring a group communication process so that the process is effective in allowing a group of individuals, as a whole, to deal with a complex problem" [Linstone and Turoff, 1975a]. It is an instance of moderated communication: a facilitator[1] serves a series of questions to the participants,[2] who return their answers to the facilitator. The answers are anonymised and collated and returned to all participants, who can then modify their response in the light of the feedback from the previous round. Alternatively, the facilitator may pass out a new set of questions, which have been designed to incorporate the returns from the previous round.

These iterations are also known as the feedback process, which

> allows and encourages the selected Delphi participants to reassess their initial judgments about the information provided in previous iterations. Thus, in a Delphi study, the results of previous iterations regarding specific statements and/or items can change or be modified by individual panel members in later iterations based on their ability to review and assess the comments and feedback provided by the other Delphi panellists [Hsu and Sandford, 2007a].

The process can be repeated any number of times, *e.g.* until consensus is reached. It is also not uncommon to predetermine the number of iterations and end the study even if the opinions have not sufficiently con- or diverged [Mullen, 2003]. Cyphert and Gant [1970] note that three iterations is usually enough.

The Delphi method draws its strengths from its three fundamental principles [Dalkey, 1967]:

[1] Also often called a moderator; in this text, I use "facilitator."
[2] Also known as experts, respondents, or panellists; I avoid speaking of "experts" in this text (see section 5.4)

Anonymity — a "device to reduce the effect of the socially dominant individual." In an anonymous setting, participants can voice their opinions without fear of being attacked or ridiculed for them.

Controlled feedback — a "device to reduce noise (among other things)." This feature prevents heated and personal debates among participants and thus helps to keep the study on track.

Statistical "group response" — a number of statistical indices exist which can be used to express the representative group opinion, such as the median, which can be found even without unanimity among the respondents. Thus, this is a "device to reduce group pressure toward conformity."

In section 5.2, I argue why these principles make it a very suitable study technique in a FLOSS context. Rather than a rigid, step-by-step prescription, the Delphi approach is a collection of guidelines and tools, which to mix and match according to the requirements. In section 5.4, I discuss the design considerations for the study at hand.

The Delphi approach has been compared to the "nominal group technique" and differs from it mainly in its distributed nature: respondents do not have to be in the same place at the same time to participate in a Delphi study [Mitchell and Larson, 1987].

5.2. Suitability for this research

The Delphi method is transparent, democratic, and can yield rapid results covering a wide range of expertise [Linstone and Turoff, 1975a, Turoff and Hiltz, 1996]. It is asynchronous and independent of the communication medium, and thus allows for participation from around the planet [Linstone, 1978]. According to Turoff [1970], the Delphi approach is a suitable means to pursue any of the following five objectives:

- to determine or develop a range of possible alternatives
- to explore or expose underlying assumptions or information leading to differing judgements;
- to seek out information which may generate a consensus on the part of the respondent group;
- to correlate informed judgements on a topic spanning a wide range of disciplines; and

- to educate the respondent group as to the diverse and interrelated aspects of the topic.

By giving each participant enough time to formulate his/her opinion independently of the others' opinions, the Delphi method helps to avoid the problems related to "groupthink" [Janis, 1972]. At the same time, the process overcomes many barriers of communication, including language barriers, as well as intimidation, enables the holding of unpopular views, or the general disagreement with another [Barnes, 1987, cited in Yousuf [2007b]].

Furthermore, since responses are presented to the group anonymously, content can be judged on merit, rather than based on the proponent's social status [Dalkey and Helmer, 1963, Linstone and Turoff, 1975a].

Linstone [1978] sees it as the best approach to complex or broad problems, or any problem which "does not lend itself to precise analytical techniques but can benefit from subjective judgments on a collective basis" Where surveys commonly try to identify the *status quo*, a Delphi study seeks to discover what could or should be [Miller, 2006]. My study does not seek to establish which tools and techniques are currently in use in the Debian Project, but how and when new tools and techniques could be spread more effectively. The Delphi approach seems like an ideal means to answer the underlying questions.

5.2.1. Delphi and asynchronous communication in FLOSS

As I have described in section 2.2.7, the Debian Project relies heavily on electronic communication media. Even though synchronous media (*e.g.* Internet relay chat (IRC)) are an integral component of day-to-day activity, most communication takes place on (asynchronous) mailing lists. Debian contributors are thus comfortable with e-mail and use it on a regular basis. Moreover, they have experience with e-mail as a medium for discussion, which Delbecq et al. [1975, p. 84] cite as a necessary precondition for a Delphi study.

Linstone [1975, p. 564] wrote: "if the [Delphi] technique is viewed as a two-way communication system rather than a device to produce consensus it fits this evolving culture admirably." The "evolving culture" he speaks of is characterised by "societal change from a homogenistic to a heterogenistic logic" and a "growing significance of normative planning," and applies equally well to the FLOSS movement, and thus Debian, due to the diversity of the respective community, united by the idealism of freedom and perfection (see section 2.2.4).

Mailing lists allow one to distribute an e-mail sent to a single address to hundreds of people within seconds. The main Debian development list (`debian-devel`, which is only one of many hundred mailing lists the project serves) has over 2,000 subscribers[3] and relays a new message every 25 minutes, or about 60 messages per day.[4] Many developers have stopped keeping up with this volume, a problem that is not uncommon in the FLOSS world: the `linux-kernel` mailing list receives between two and three hundred messages per day[5] and few read the list traffic in its entirety.

To deal with the volume of messages on lists like that, services such as Kernel Traffic[6] have tried to provide regular summaries,[7] but nowadays, only professionally-run sites like Linux Weekly News[8] can keep up with the pace, and only because they skim bits off the top — the vast majority of posts are only read by a few faithful followers, the occasional passerby, and otherwise indexed by search engines for later, non-interactive reference.

On Debian lists (and most other FLOSS lists too), discussion threads often grow out of control, spanning hundreds of messages and running for several weeks. In most cases, it is impossible to stay on top of the discussion, which usually involves several dozen (if not hundreds of) contributors. Part of the difficulty is the asynchronous nature of the medium: in response to a simple question, five readers may send their responses to the mailing list around the same time. It is not unlikely for two or three of these responses to be challenged,[9] which creates sub-threads and repeated arguments. This vicious cycle can start at any level of a thread, and it should not be hard to see how quickly threads on busy mailing lists can deteriorate and become more a burden than a communication medium.

Occassionally, someone will step in and post a summary in an attempt to consolidate the thread;[10] at other times, correspondents voice their desire for summaries.[11] The dilemma is that mailing lists fork into breadth, while for certain types of discussions, cooperation on a single, coherent document would be more effective.[12]

[3] `http://lists.debian.org/lists/debian-devel/details.html` [18 May 2008] and `http://lists.debian.org/stats/` [26 Jun 2009]

[4] `http://madduck.net/blog/2007.10.10:not-so-chatty/` [18 May 2008]

[5] `http://vger.kernel.org/vger-lists.html#linux-kernel` [18 May 2008]

[6] `http://kerneltraffic.org` [18 May 2008]

[7] A Debian Traffic service once existed too: `http://www.kerneltraffic.org/debian/` [18 May 2008]

[8] `http://lwn.net` [18 May 2008]

[9] `http://xkcd.com/386/` [18 May 2008]

[10] *e.g.* `http://lists.debian.org/msgid-search/20030713101750.GD1776@galadriel` [18 May 2008]

[11] `http://lists.debian.org/msgid-search/200610022136.09710.avbidder@fortytwo.ch` [18 May 2008]

[12] The idea behind Google Wave (`http://wave.google.com/`) is not new, and their implementation possibly inappropriate for many FLOSS projects, but there are other ideas floating around (*e.g.* `http://www.nabble.com/inverted-quoting-td20419693.html` [26 Jun 2009]).

Over the past years, the use of Wikis has grown to fill this void, and discussion participants often collaborate on a Wiki page created to keep the arguments and facts manageable (see also appendix A.2). In the Debian Project, the Wiki[13] is used formally to develop so-called "enhancement proposals" (DEPs).[14]

Especially due to the controlled feedback employed as part of the Delphi method, it seems like an ideal approach to discuss over electronic mail without the danger of the thread growing too large to be useful.

5.2.2. Benefits of controlled feedback in a FLOSS context

Prior to the study, I had speculated that the participants of my Delphi panel had not previously thought much about the issues I raised during the study (this was confirmed on numerous occasions during the study). Had I opted for regular "expert interviews", then I could have obtained a collection of different views on the same issue, but there would have been no cross-impact, which might have allowed a participant to consider the questions in a different light.

It is paramount to view the Delphi method not as a data collection tool, but as a mode of communication [Turoff and Hiltz, 1996]. Its principal appeal for my study was the feedback component, because it allowed for the iterative development of each participant's opinion, thus enabling him/her to arrive at a well-founded conclusion. As Yousuf [2007a] put it, the Delphi approach's "iterative feedback method develops an insight, which in its totality, is more than the sum of the parts." With reference to the aforementioned inappropriateness of asynchronous mailing lists in the development of solutions and proposals, the Delphi method comes to the rescue by engaging participants in a feedback cycle, mediated by the facilitator to ensure that information can be merged while it accumulates.

Since all communication in a Delphi study flows via the facilitator, and it is part of the Delphi approach that s/he feeds back summaries of the other respondents' replies to everyone, the Delphi approach is an ideal method for structured communication among FLOSS contributors, for at least the following reasons:

- by summarising messages, the total volume and thus time requirement on the part of each participant is minimised.

[13] `http://wiki.debian.org`
[14] `http://dep.debian.net/deps/dep0/` [26 Jun 2009]

- respondents know they can (and are expected to) take their time, without having to fear to miss anything (or not to be the first to make a certain statement).
- respondents are fed the entire context in a single message, thus alleviating the need for them to wade through repetitive messages and search for information in various places.

Okoli and Pawlowski [2004] provide a comprehensive comparison between traditional surveys and Delphi studies, in which they also address validity of constructs. During a Delphi study, the researcher has a direct channel of communication to each participant, who is non-anonymous to him/her. In the case of unclarity, the researcher can follow-up on participant responses to help ensure the correct capturing of the participant's point of view, which is a significant benefit in terms of construct validity when compared to traditional surveys.

Furthermore, should issues over attrition or non-response arise, the researcher can establish contact with the affected participant to make an effort to rescue the situation.

Finally, Okoli and Pawlowski [2004] mention the "richness of data" as an evaluation criterion and difference between surveys and Delphi studies. While surveys help normalise responses and thus enable large-scale evaluations, Delphi studies foster richer data as they do not (have to) impose restrictions on the participants' answers, incorporate the refinement of responses into the process, and allow for the aforementioned *post-hoc* follow-up questions to eliminate remaining unclarity, misunderstandings, or doubt. In addition, participants can be given a point of contact in case of any questions during and after the study (see section 5.4.6).

Before the Internet became ubiquitous and before it gave rise to the FLOSS movement in such global extents as it has today, Linstone and Turoff [1975b, p. 483] postulated on computers and the future of the Delphi method:

> The current generation of computers, associated hardware, software, and particularly terminals, now begins to provide on an economic basis the capabilities for considerable augmentation of human communications. When this technical capability is coupled to the knowledge being gained in the area of Delphi design, all sorts of opportunities seem to present themselves. Underlying this view (or bias) is the assumption that Delphi is fundamentally the art of designing communication structures for human groups involved in attaining some objective.

Even though technology does not itself solve social problems, it seems a given that the computers and the Internet have revolutionised human communications and made such movements as

FLOSS possible. FLOSS projects are very often characterised by visions shared among their supporters [Raymond, 1999], and if the assumption holds, then the Delphi method is an ideal alternative to the existing communication structures in the FLOSS world.

5.2.3. Reasons for a Delphi approach

According to Reid [1988], the decision for the Delphi method in a research endeavour depends on the appropriateness of available alternatives, and Linstone and Turoff [1975a, p. 4] asserts that the use of the Delhi a significant method depends on the "particular circumstances surrounding the necessarily associated group communication process."

I chose the Delphi approach for the study at hand because the technique and the underlying communication process fit seamlessly with the FLOSS culture, and the Debian Project in particular. The following are the reasons that led me to use the Delphi method[15]:

- "the problem I research does not lend itself to precise analytical techniques but can benefit from subjective judgments on a collective basis";
- "more individuals are needed than can effectively interact in a face-to-face exchange";
- "time and cost make frequent group meetings infeasible";
- "the heterogeneity of the participants must be preserved to assure validity of the results, *i.e.*, avoidance of domination by quantity or by strength of personality ('bandwagon effect')";
- potential panellists are very familiar with the mode of communication, and the controlled feedback appeals to them, as it gives them the time and opportunity to participate;
- verification cycles (follow-up messages) to ensure that I correctly understood what each participant meant to communicate boost construct validity;
- the possibility of follow-up interviews, which help gaining a deeper understanding of the issues involved and also add to the validity of the study.

Furthermore, my study explored complex, interpersonal, and often subjective causes, which called for qualitative analysis and open-ended research. I chose the Delphi method because it appears to be an optimal means to achieve my endeavour.

[15]The first four of these reasons are lifted verbatim from [Linstone and Turoff, 1975a, p. 4] and hence quoted.

Variant	Variance	References
Modified Delphi	closed questions	Kerlinger [1973]
Wide-band Delphi	discussion among experts prior to submission of individual responses	Boehm [1981]
Real-time Delphi	immediate availability of responses to all	Gordon and Pease [2006]
Ranking Delphi	brainstorm/augment a set, select, and rank	Dickson et al. [1984], Hsu and Sandford [2007a]
Policy Delphi	establish differing positions instead of consensus	Turoff [1970]
EFTE	committee-like discussion following the study	Nelms and Porter [1985]

Table 5.1.: *Variants of the Delphi method I considered for this research.*

5.3. Variants and modifications

Over the years, the Delphi method has been modified in many ways to create new types of approaches, and new names have been given to existing approaches. Mullen [2003] provides an extensive list of such variations.

The Delphi approach was originally put forth by Dalkey and Helmer [1963]. It is known as the "Delphi method" without further specification. In the light of the many variations that have been developed, a number of adjectives have been used to specify the original Delphi approach, including "classical", "original", "plain", "standard", "conventional", *etc.*. In this text, I limit myself to "classical Delphi".

In this section, I introduce the most notable modified versions of the classical Delphi approach, which are summarised in table 5.1 on this page. The domain of Delphi approaches has been criticised as heavily convoluted [*cf.* Mullen, 2003]. On the other hand, the "numerous variations, advances and so forth on this variety of structured conversation, which have occurred since the early 60's, [...] more effectively address the issues of Delphis as a tool for drawing forth ideas, options, alternatives, diagnosis, etc." [Coates, 1975].

Initially, a variant of the Delphi method existed by the name of "**modified Delphi**," which differed from the original Delphi method in its use of closed questions; the original Delphi used open-ended questions [Kerlinger, 1973]. The openness of Delphi questions seems to have since become just another variable in the choice or design of the particular Delphi approach, as other, more substantially modified approaches have appeared.

Another flavour, the **wide-band Delphi** approach, was developed by Farquhar and Boehm in the 1970s [Boehm, 1981, p. 335]; a wide-band Delphi study incorporates discussions among experts

before each expert submits his/her responses. Stellman and Greene [2005, ch. 3] provide an excellent summary of this technique.

Real-time Delphi studies do not collect and collate responses but instead make them available in anonymised form to all participants immediately [Gordon and Pease, 2006]. Linstone and Turoff [1975a] use the term real-time Delphi study as a synonym for "Delphi conference," which differs from the conventional Delphi approach in that a computer replaces the facilitator. They note that a real-time Delphi approach "require[s] that the characteristics of the communication be well defined before the Delphi is undertaken, whereas in [...] [a conventional Delphi], the monitor team can adjust these characteristics as a function of the group responses" [*ibid.*, p. 5].

Commonly used nowadays are **ranking Delphi** methods, in which participants brainstorm, select, and rank items to facilitate the identification of the most influential issues in a given scenario [*e.g.* Dickson et al., 1984, Hsu and Sandford, 2007a]. Okoli and Pawlowski [2004], Schmidt [1997] provide comprehensive guidelines for this type of study.

A widespread variant of the Delphi approach is the **policy Delphi** method, which is suited to answer questions "for which there are no experts, only advocates and referees," and whose goal is "not so much to obtain a consensus as it is to establish all the differing positions advocated and the principal pro and con arguments for those positions" [Turoff, 1970]. The policy Delphi is often considered to be a whole class of Delphi approaches, rather than a single approach [Rauch, 1979]. For instance, the **decision Delphi** approach, which aims to "prepare, assist, and make decisions," is an instance of a policy Delphi [*ibid.*].

Nelms and Porter [1985] proposed an extended version of the Delphi approach by the name of "**EFTE**", which expands to estimate, feedback, talk, estimate, and incorporates a regular, committee-like discussion round before the final estimation takes place. The literature does not suggest that this method has received much attention or has since been used at all.

Finally, a technique related to the Delphi method is **cross-impact analysis**. This method of analysis "seeks to find the conditional probability of an event given that various other events have or have not occurred" [Gordon and Hayward, 1968]. It thus aims to better take into account the complexity around events or scenarios to be forecast.

5.4. Design considerations

The breadth of modifications to the Delphi method (see section 5.3) is causing methodological difficulties [McKenna, 1994]. One of the strongest critics is Sackman [1974, 1975], who concludes that "conventional Delphi is basically an unreliable and scientifically unvalidated technique in principle and probably in practice," and that the results of a Delphi study are "unreliable and invalid." He dismisses the approach "until its principles, methods, and fundamental applications can be established experimentally as scientifically tenable" [*ibid.*, p. vi].

Sackman's analysis was not well received by the community: Helmer [1977] calls it a "singularly vituperative attack" [footnote 4], and Coates [1975] began a response paper by describing Sackman's work as "a splendid example of trained incapacity diligently applied to spawn a rather attractive irrelevancy." Both authors, as well as many others, fail to see beyond Sackman's alleged attack. Goldschmidt [1975], on the other hand, acknowledges that many of Sackman's criticisms "are valid and should be noted." However, Goldschmidt also refutes most claims and sees Sackman's text as an encouragement to Delphi practitioners "to be more critical of methods, results, and conclusions" [*ibid.*, p. 212].

The important point to gather from Sackman's criticisms is that there is no single, scientifically tenable "Delphi method," but that the Delphi approach is a specific means of communication. Linstone and Turoff [1975a, p. 3f] open their seminal book on the Delphi method stating that Delphi is not "a neatly wrapped package, sitting on the shelf and ready to use," but instead offer "a rich menu of procedures from which [the reader] may select his own repast if he should seek to employ the Delphi technique." That was written in 1975a, but judging from the vast number of variations and "loose interpretations" of the technique available today [*cf.* Mullen, 2003], it is all the more true that the Delphi approach remains a flexible, generic tool which exposes many variables that need to be adjusted for each application individually. However, no matter what combination of variables is chosen, the Delphi technique requires a high degree of methodological precission [Peiró Moreno and Argelaguet Portella, 1993].

Schmidt [1997] picks up on the "lack of a definitive method for conducting the research" as well as the "lack of statistical support for the conclusions drawn by the researcher" and presents a "method, based on nonparametric statistical techniques, to conduct ranking-type Delphi surveys, perform analysis, and report results." Okoli and Pawlowski [2004], use Schmidt's method in an example study, "fill[ing] in many details in the context of Schmidt's framework by providing guidelines on how to conduct a rigorous Delphi study [...]."

Hung et al. [2008] provide a comprehensive overview of strengths and limitations of the Delphi approach, as well as numerous pointers into the literature analysing these aspects.

In anticipation of methodological concerns, and with reference to the sub-objective of this research to help further the use of the Delphi method in the FLOSS context (see section 4.3.2), I detail my design considerations in the following sections. For every variable I could find, I argue for why my interpretation is appropriate in the context of my research. This helps understanding the research approach, which should alleviate methodological concerns, and provide a sound basis for future users of the Delphi method in a FLOSS environment.

5.4.1. Choice of the specific Delphi method

In my research objectives, I propose to identify the salient influences that shape adoptions of innovations by Debian Contributors (DCs). Hence, a ranking Delphi technique initially seemed like the appropriate approach to achieve my objectives. However, ranking Delphi studies are strongly consensus-oriented, and therefore not the best choice for an exploratory study. In the following, I argue why the ranking Delphi technique is not suitable *by itself*. Instead, I propose a policy Delphi method instead, but borrow from the ranking-type approach towards the end of the study.

The policy Delphi method is a technique that can suit any or any combination of the following goals: "to ensure that all possible options have been put on the table for consideration, to estimate impact and consequences of any particular option, and to examine and estimate the acceptability of any particular option" [Turoff, 1975, p. 83]. Murray [1992, p. 18, cited in [Franklin and Hart, 2007]] describes the policy Delphi study as a means to collect a "rich, meaty, stimulating body of opinion" to inform sound decision-making. This study's goal is to provide diffusers of tools and techniques in the Debian Project with substance on which to ground their decisions and strategical choices. Such decision makers are "not interested in having a group generate [their] decision; but rather, have an informed group present all the options and supporting evidence" [Turoff, 1975, p. 80]. This makes the policy Delphi technique an attractive choice for my endeavour.

Nevertheless, the policy Delphi is very strongly focused on "exploring and obtaining the reasons for disagreements," to the point where this becomes an integral part of the technique itself [Turoff, 1975, p. 84]. In the context of my study, the reasons for disagreement are a valuable intermittent vehicle, but not a desired outcome of the study. Rather, disagreements (or divergence) has to be anticipated, because the study explores behavioural decisions made across a

group of people, but as soon as two divergent influences have been established, there is no need to explore why they differ any further.

Furthermore, a policy Delphi is usually followed by a small committee formulating the policy [Turoff, 1975, p. 83]. I explicitly exclude such an activity from my research, as the Debian Policy (see appendix A.1.5) is a document which records established standards, it does not dictate them. Any results from this study will have to stand up to practical scrutiny before they can be formalised.

Seeking the set of salient influences, however much divergent across the Delphi panel, I based my approach mostly on the policy Delphi approach, but incorporated some of the ranking guidelines proposed by Schmidt [1997], in true *bricoleur* spirit [*cf.* Okoli and Pawlowski, 2004]. In the following, I illustrate the design choices I made for my study in the light of common Delphi criticisms.

5.4.2. "Experts"

Sackman [1974, p. 703] criticises the use of the word "expert", claiming that "it is almost impossible to find current psychometric or social science literature on 'experts'." Kaplan [1971, cited in Hsu and Sandford [2007a]] notes that "throughout the Delphi literature, the definition of [Delphi subjects] has remained ambiguous."

Leaving aside the semantics of the word, Linstone [1975] writes that the "specialist is not necessarily the best forecaster. He focuses on a subsystem and frequently takes no account of the larger system."

Policy Delphi studies (see section 5.3) explicitly refrain from the use of experts and invite "informed advocates and referees" instead [Turoff, 1970]. In my study, I avoid the word "experts" and use the term "participants" or "panellists" for members of the Delphi panel to emphasise their participatory and indispensable role in accomplishing the research objective instead.

5.4.3. Participant selection

Selecting the right participants for a Delphi panel is key to a successful study. Okoli and Pawlowski [2004] call panellist selection "perhaps the most important and yet most neglected aspect of the Delphi method," and Judd [1972] claims that deciding "who is an expert" is "the single most confounding factor in panel selection."

It is certainly not an easy task to amass the right people [Mitchell, 1991], which is why experts are often picked on the basis of convenience, rather than qualification. Resulting panels are thus often made up of "(1) participants who are available, (2) other participants whose reputation is known to the experimenter, or (3) those who meet some minimum requirement such as membership in a professional society " [Hill and Fowles, 1975]. The results are diminishing returns [*ibid.*].

According to Hsu and Sandford [2007a, citing Helmer and Rescher [1959], Oh [1974]], "choosing individuals who are simply knowledgeable concerning the target issue is not sufficient nor recommended." They further quote Ludwig [1994]: "solicitation of nominations of well-known and respected individuals from the members within the target groups of experts was recommended." Refering to Kaplan [1971], Ludwig [1994], they note that the selection pool of Delphi subjects will likely include positional leaders. This claim was the basis for my choice of the snowball sampling approach discussed further down.

Delbecq et al. [1975] make substantial recommendations for the selection of qualified panel members for a nominal group technique study, and specifically mention the applicability of their guidelines to the Delphi method [*ibid.*, ch. 4].

5.4.3.1. Diversity

Linstone and Turoff [1975a] maintain that diversity in the panel is necessary to avoid "domination by quantity or by strength of personality." Delbecq et al. [1975] add that members of widely varying personalities and substantially different perspectives on a problem enable higher quality results than homogeneous groups.

Initially, my plan was to invite members from the Debian Project, as well as other projects and disciplines to the panel, but I discarded this idea since it would have exposed the risk of loss of focus of the whole study [*cf.* Mintzberg, 1979]. I deemed it impossible to summarise and feed back the responses to all panellists without a common background or terminology [*cf.* Conboy, 2006, p. 252]. My fear was that a panel whose diversity extended beyond the Debian Project would have made it impossible to discuss the issues in depth, a concern backed up by Powell [2003].

Instead, I sought diversity from within the project, chosing participants equally along four dimensions I had identified to be relevant in discussions on tool adoption in the Debian Project before starting this research:

Cooperation — Is the person a lone worker or a team player? Lone workers are more independent in their choices, whereas team players need to coordinate with others.

Tasks — Does the person work on similar tasks (*e.g.* maintenance of various related packages), or are the tasks diverse? Those working on similar tasks have a harder time trying out new ideas than those who can *e.g.* investigate the performance of a new tool in the context of a small, unrelated task of theirs.

Tools — Does the person work with a uniform set of tools and uses the same techniques across all his/her work, or does day-to-day work involve a large number of different tools? I expect people in the latter group to have been confronted with adoption decisions more often, while members of the former group would be more critical.

Interest — Is the person interested mainly in getting things done, or does s/he expend time thinking about improved ways to achieve goals? The distinction here is between pragmatists and visionaries [Moore, 1991], and the former are going to be slower and more conservative to adopt a new idea, than visionaries, who are the innovators and early adopters.

With reference to Pill [1971]'s claim that "many innovations and real breakthroughs [...] occur from outside a discipline or specialty," I concluded that a group spread between these four dichotomies would have enough differing opinions and views to prevent an instance of "group bias" [Chan, 1982].

5.4.3.2. Sampling strategies

One of the core claims of Sackman [1974] is that Delphi fails to observe properly randomised polling procedures. Helmer [1977] responds that this criticism is void because it takes the Delphi method for something it is not trying to be:

> A Delphi inquiry is not an opinion poll, relying on drawing a random sample from "the population of experts;" rather, once a set of experts has been selected (regardless of how), it provides a communication device for them, that uses the conductor of the exercise as a filter in order to preserve anonymity of responses.

Indeed, as Goodman [1987, cited in Mullen [2003]] notes, the creators of the Delphi method "tend not to advocate a random sample of panellists [...] instead the use of experts or at least informed advocates is recommended." Coates [1975] makes the following comparison: "The

Delphi is in some sense a modern equivalent of the essay. To confuse Delphi with a psychometric test is to confuse an essay with a scientific treatise."

Powell [2003] recommends that panellist be "willing and able to make a valid contribution," and Adler and Ziglio [1996, cited in Skulmoski et al. [2007]] suggest that participants of a Delphi study should meet four requirements: (i) knowledge and experience with the issues under investigation; (ii) capacity and willingness to participate; (iii) sufficient time to participate in the Delphi; and, (iv) effective communication skills. Delbecq et al. [1975, p. 84] see high participant motivation as a criticial condition for a Delphi study.

Coming into this phase of the research, I briefly considered sending a mass invitation to all project contributors and waiting for enough to reply and sign up. I discarded that strategy, however, because that would have populated the Delphi panel primarily with those who are interested in workflow meta-issues, not those who maintain packages and work with others instead. This exemplifies that random sampling is not a viable sampling method for this research, as it would take too long to populate the panel with people who fulfill these requirements. Ludwig [1997] echoes this: "randomly selecting participants is *not* acceptable" (her emphasis).

According to Beech [1999], the "absence of the usual representative sampling techniques" is a common problem in Delphi studies. I therefore considered each of the fifteen purposive sampling strategies put forth by Patton [2001] for the study at hand. For this study, I employed the following techniques:

intensity sampling — I created a list of visible ("intensive") project participants, …

snowball sampling — … and asked them for nominations of participants. This was done in line with the recommendation by Skulmoski et al. [2007] to use a combination of purposive and snowball sampling for Delphi studies.

stratified purposeful sampling — I defined four strata of characteristics of their interests and involvement with the project, …

maximum variation sampling — … and asked the nominees to position themselves to enable me to select the subset with maximum variation.

opportunistic sampling — I kept my eyes open before and during the study for further sampling opportunities.

5.4.3.3. Snowball sampling and the Debian Project

The Debian Project is a fairly close-knit group with a plethora of communication channels and high visibility of its members. A snowball sampling strategy is likely to identify a consistent set of people. The number of nominations a person receives can nicely serve as a measure of his/her competence and help in selecting panellists. Gordon [1994b] warns that this kind of "daisy chaining" exposes cliques. He suggests instead "to form a matrix in which the required skills are listed," which is what I did by selecting participants according to their positions on the four aforementioned strata.

Adler and Ziglio [1996, p. 87f] deem it unrealistic to expect effective participation unless respondents "feel that the aggregation of judgments of a respondent panel will include information which they too value and to which they would not otherwise have access." Even though members of the Debian Project could (and do) discuss tool adoption, these efforts are not rigorous or broadly scoped. As I will discuss in section 8.4, both interest in the subject, the curiousity about the Delphi approach, and the prospect to engage in challenging discussions with potential benefit for the project were important incentives for the panellists to participate.

5.4.4. Panel size

An important variable of a Delphi study is the number of participants. Panel size suggestions range between a handful and several thousand, "with accuracy deteriorating rapidly with smaller sizes and improving more slowly with large numbers" [Linstone, 1978, Mullen, 2003]. Smaller panels amplify problems with decreasing response rates and participant attrition, which are of particular concern with Delphi studies [Hill and Fowles, 1975]. Okoli and Pawlowski [2004] claim that "the literature recommends 10–18 experts on a Delphi panel," while Hsu and Sandford [2007a] state that no consensus on an optimal number exists.

Ludwig [1994, p. 52, cited in [Hsu and Sandford, 2007a]] notes that the number of panellists in a Delphi study is "generally determined by the number required to constitute a representative pooling of judgements and the information processing capability of the research team." I shall inspect these two in turn.

Earlier, I identified four dimensions, or strata, which I used as guideline for the purposive sampling strategy. On the assumption that all 16 combinations of dichotomies on these strata make sense, a panel of 16 would be diverse enough to constitute a representative

pooling. However, it was unlikely to find people with characteristics to match those extremes.

Furthermore, I expected some panel candidates to fit into multiple categories, because they *e.g.* worked in a team as well as individually. Also, other candidates might have very balanced experiences and would be found more towards the centre of the dimensions. Thus, it seemed sensible to keep a few additional slots empty and fill them with exceptional panellists.

I therefore aimed for a panel size of 16–20, which seemed in line with recommendations, leaves enough room for diversity across the four spectra, and is also manageable. Considering that all participants are DCs and thus of rather homogeneous background, the number seemed adequate [Delbecq et al., 1975, Hsu and Sandford, 2007a, Ludwig, 1997]. I organised funding to compensate up to 19 panellists (see section 5.4.9.2).

5.4.5. Degree of anonymity

Anonymity is a core principle of the Delphi approach, but in recent times has become a research approach variable and can vary from total anonymity, where the participants are anonymous even towards the facilitator, to Delphi studies that lose anonymity over their course [Mullen, 2003].

The choice of degree of anonymity of a study deserves a lot of thought. On the one hand, anonymity seems to go against the idea of openness, as cultivated by the FLOSS community, and panellists might try to disclose each other's identity. Furthermore, a number of panellists were actively involved in the conceptualisation and design of new tools for the Debian workflows throughout the Delphi study, which made it hard for members to discuss their own design choices without revealing their identities. Finally, since communication in a FLOSS environment is chiefly written and spans all levels of English fluency, responses will need careful paraphrasing to prevent identity disclosure on the basis of writing style and command of the language.

Also, anonymity between participants of a Delphi study may also impose problems and impact the reliability of the study, as it may result in a lack of responsibility for answers, or encourage quick replies from the panellists [Sackman, 1975]. I cannot see much weight in this claim because the respondents are not anonymous towards the facilitator, who can (and needs to) assure that responses are not careless.

On the other hand, anonymity brings a number of benefits to the study. I expected several panellists to have high social status within the Debian Project, and their opinions could thus be judged and accepted by the other respondents based on status rather than merit [Dalkey and Helmer, 1963, Linstone and Turoff, 1975a]. Anonymity, or rather the role of the facilitator as a relay also serves to reduce noise and the pressure for conformity [*ibid.*], and ensures that the exchange does not become heated and personal [Spinelli, 1983]. Along those lines, an anonymous setting allows individuals to more freely voice their opinions [Armstrong, 1989]. Anonymity does appear to be more conducive to independent thought and helps the formation of considered opinions [Dalkey, 1967].

In case one needs to refer to another panellist, a "quasi-anonymous" study [Keeney et al., 2006] can help: subjects' identities are then aliased with invariant labels (*e.g.* "subject A") where necessary for the duration of the study. I kept the option, but found that it was not necessary to track identities throughout.

Each participant had the choice whether his/her name could be disclosed alongside his/her comments after the study had ended, while the publication of their comments — anonymously or not — was a requirement of participation (see section 4.3.1).

5.4.6. Communication medium

Given the nature of FLOSS and the Debian Project, with individuals in different timezones and globally-spread, it was natural for my study to make use of electronic communication media, as I could expect all participants to be familiar therewith (see section 2.2.7 and *cf.* Delbecq et al. [1975, p. 84]). Witkin and Altschuld [1995, p. 204, cited in [Hsu and Sandford, 2007a]] list several advantages of electronic media: "(1) the storage, processing, and speed of transmission capabilities; (2) the maintenance of respondent anonymity; (3) the potential for rapid feedback." I considered three electronic media: through-the-web, over IRC, or by e-mail.

Interviews over IRC are synchronous and require participants to be on-line to take part, which introduces scheduling problems. Furthermore, since the chat protocol is line-based — text is only transmitted to the peer when the return key is hit — correspondents do not know when the other person is starting to "speak," they only see the entire line once it has been completed and sent. As a result, one typically finds two or three parallel threads of discussion in a single IRC conversation, especially when the discussion is intense or both parties have a lot to say. Such

discussions can be very hard to follow and usually go off on tangents easily; it requires discipline on both sides to maintain focus.

With respect to the Delphi study, IRC is an inappropriate medium to communicate the question(s) at the start of each round, because it is almost impossible for the facilitator to establish context and convey the question without the respondent interrupting — the medium is interactive and the threshold to respond almost nonexistent. However, IRC can be a very useful tool for participants when they seek clarification of the Delphi questions, or for the facilitator to inquire further about a panellists responses.

Web-based approaches suffer from a number of problems, such as the requirement to be connected at the time of participation, rendering problems, frustrations with the user interface, and inability to judge the expected time until completion. In addition, as contributors to the Debian Project, I speculated participants to prefer non-web media for communication, which turned out to be correct (see section 8.4.3.2). An obvious benefit of a web-based representation of the study's findings is the ability to inter-link pages.

[Sheehan and Hoy, 2006] discusses advantages and disadvantages between web- and e-mail-based approaches, and I refrain from reproducing their arguments. The following thus concentrates only on aspects which are specifically relevant to a study among highly skilled computer users, as can be said about members of the Debian Project, who are used to high volumes of e-mail (see section 2.2.7).

E-mail provides solutions to a number of the problems with web-based approaches: participants can use their e-mail programme and editor of choice to participate in the study, and know from the length of the original mail what time commitment is to be expected, at least to read the instructions before being able to gauge the actual effort demands. Furthermore, electronic mail can be signed and thus authenticated, allowing for the automated processing of resubmission of responses without the danger of manipulation. E-mail messages are free-form text, enabling the researcher to provide all needed information, and giving full liberties to the participants. Problems arise from character set and formatting incompatibilities, and spam filters can get in the way, but these are easily alleviated and can even be prevented up front.

E-mail is asynchronous and does not require scheduling. Linstone and Turoff [1975b, p. 483] claim that the benefit of asynchronous communication "with the computer in the loop" is that "each participant is free to choose when he wants to talk (type) or listen (read) and how fast or slowly he wants to engage in the process." They go on to state that "since all the individuals are operating asynchronously, more information can be exchanged within the group in a given

length of time, as opposed to the verbal process where everyone must listen at the rate one person speaks" [*ibid.*].

E-mail replies make it easy for the facilitator to track responses and followup to participants. This comes in handy when a respondent has difficulty getting at the gist of a question [Oppenheim, 2001], a respondent's statement is unclear, or the facilitator asks participants to read over and approve, whether a paraphrased and anonymised text still reflects their opinions [*cf.* Oppermann, 1995]. Similarly, personal contact is exceptionally suitable to exploratory research as it opens the possibility to expand on additional fields of relevance or interest [Oppenheim, 2001, Yin, 2003]. I consulted with participants individually for both reasons at various points throughout my study.

One problem with e-mail is that it does not allow for immediate feedback, *e.g.* to prevent a participant from expending significant amounts of time elaborating a response after having misunderstood the question; this problem is similarly present in web- or regular mail-based surveys, and only marginally addressed when the telephone is used as medium. To prevent misunderstandings arising, I carefully drafted e-mails to be sent to the panel very, and had them reviewed by a number of people, before distributing them. Furthermore, I encouraged participants at all points to get in touch with me in case of any uncertainty (also see section 5.6).

Oppermann [1995] cites many disadvantages of e-mail as a survey medium, which are all outdated (especially in the context of the Debian Project; *cf.* also Sproull [1986]) except for one, which is all the more true today than it ever was: "the increasing interactions with e-mail means that many users find their mailbox 'full,' [and] just delete everything of lesser interest to them." I describe my approach to preventing any problems related to this in section 5.5.

For my study, I chose the e-mail medium as primary means to communicate with the panellists because it let me provide the necessary context and background information, and allowed for easy archival. As previously argued (section 5.2, which refers to section 2.2.7), the panellists came into the study with a great deal of experience with e-mail, which makes the choice all the more natural. Nevertheless, I made sure that the participants could also get back to me using IRC, Voice-over-IP (VoIP), and I issued standing offers to call them on the telephone.

5.4.7. Consensus vs. disagreement

The question of whether the goal of a Delphi study is to reach consensus is heavily debated. Dalkey and Helmer [1963], the creators of the approach, defined it to be a method to reach "consensus of opinion," but Sackman [1974] calls consensus "specious" as it may be a "watered-down version of the best opinion" (which Helmer [1977] does not refute), and Hill and Fowles [1975] claim that consensus may be achieved in the interest of harmony, rather than correctness [*cf.* Powell, 2003, Rennie, 1981]. Linstone [1975] warns his readers that "consensus of experts does not assure good judgment or superior estimates," and Hill and Fowles [1975] identify the pressure for convergence as a dangerous pitfall of the Delphi method.

Furthermore, Linstone and Turoff [1975a] see one of the common reasons of failure of a Delphi study to be "ignoring and not exploring disagreements, so that discouraged dissenters drop out and an artificial consensus is generated." Such consensus could be "collective bias rather than wisdom" [Chan, 1982], and Mitroff and Turoff [1975, p. 22] recommends to maintain a clear distinction between consensus and validity of the study as a whole.

Consensus needs not be the goal of a Delphi study. In fact, Linstone [1975, p. 564] noted that "the ability to expose uncertainty and divergent views is an inherent strength of the Delphi process," and according to Rieger [1986] more attention is being paid to the possibilities of and potentials in disagreement, and the degree of consensus is rather a design variable to be set according to the study's objective [*cf.* also Mullen, 2003]. Furthermore, Armstrong [1989] finds that it can be insightful to explore reasons for failing to achieve consensus.

Turoff [1970, p. 148] defines the Delphi method to be "a method for the systematic solicitation and collation of informed judgements on a particular topic." The policy Delphi approach (see section 5.3) is specifically designed "to establish all the differing positions advocated and the principal pro and con arguments for those positions" [*ibid.*, p. 153].

The goal of my study is not to find consensus on the research question, but to identify salient influences to adoption decisions into a framework. This means that the degree of consensus for each influence determines the influence's salience. A policy Delphi "should be able to serve [the objective] that all possible options have been put on the table for consideration" [*ibid.*], which makes it an appropriate approach for the study at hand.

5.4.8. Question design

Sackman [1974] claims that questions asked in Delphi studies are generally vague, and that combining separate responses made on differing premises or interpretations leads to meaningless results. Gordon and Helmer-Hirschberg [1964] echo this concern. The problem goes both ways: questions which are too ambiguous might not yield meaningful results, but questions that are too specific are quickly perceived as suggestive and permit only a limited amount of interpretation [Salancik et al., 1971]. Scheele [1975a] advocates the former over the latter and proposes to provide anchors in the form of "mini-scenarios, [which] produces a richer set of data and stimulates more insightful interpretations."

Murray [1979] suggests the problem of question ambiguity is "not insignificant even in the more rigorously quantitative methodologies," and that it is not unique to the Delphi approach. He claims that "Delphi does, however, present a danger that unwanted question ambiguity [...] may be masked and not recognised." This danger is closely related to the diversity of the panellists, each of whom looks at the problem from a slightly different angle. While a powerful feature of the Delphi approach — Scheele [1975b] highlights "the influence of the various constructions of reality assumed by the Delphi panellists" as "the most significant result from any Delphi inquiry" — it is almost impossible to distinguish effects from the diversity in the panel from those caused by ambiguity in the questions.

The choice between open-ended and closed questions has some relevance to the ambiguity problem, as open-ended questions seek to explore breadth rather than depth. Delphi studies commonly commence with an open-ended questionaire to "maximize the chance of unearthing the most important issues" [Schmidt, 1997], and are thus often purposely phrased ambiguously [*cf.* Mullen, 2003, p. 44] and broad in scope [Marchant, 1988]. The "modified Delphi" technique is the only Delphi method which explicitly makes use of closed questions from the start, because modified Delphi studies seek very specific answers in domains where a lot of knowledge exists [Kerlinger, 1973].

The first questionaire of a policy Delphi study, on the other hand, is generally developed from the literature and consists of carefully drafted statements ("pre-formulated obvious issues" [Turoff, 1975, p. 84]), which participants rate using *e.g.* a Likert scale [Franklin and Hart, 2007, Hsu and Sandford, 2007b, Keeney et al., 2006, Turoff, 1975]. In fields without existing literature, as was the case for the study at hand, the first round of a policy Delphi study involves exploration, because the domain has to be mapped out and cannot be pulled from literature. Therefore, open-ended questions seem the better choice.

5.4.9. Participant dropout and non-response

Another concern with the Delphi approach is the time and effort that it consumes, and related respondent drop-out issues [Hsu and Sandford, 2007a]. Linstone and Turoff [1975a] include this concern in their list of typical failures of Delphi studies:

> Underestimating the demanding nature of a Delphi and the fact that tire respondents should he recognized as consultants and properly compensated for their time if the Delphi is not an integral part of their job function

Gordon and Helmer-Hirschberg [1964, cited in [Murray, 1979]] note that participants who drop out may be those who most strongly disagree with the growing consensus of the panel. Losing those panellists may shift the result to harmonious, rather than accurate [Hill and Fowles, 1975]. The addition of new members who have not participated in previous discussions may produce unknown results [Murray, 1979]. Delphi studies with a different set of panellists across the iterations damage "the very core of the Delphi procedure, [and] the results that emerge must be suspect" [*ibid.*]. Gordon and Helmer-Hirschberg [1964] advocate to contract panellists in an effort to stablise the panel roster for the duration of the study.

Hsu and Sandford [2007b] deal with the problem of non-response in Delphi studies and provide ample advice on how to counter it. In particular, they note that the correspondence should be regular and frequent to prevent losing participant's interest or context. At the same time, the facilitator needs to process copious amounts of data, which grow with the size of the experts panel. An appropriate communication medium can help to reduce both, the effort and time requirement [Turoff and Hiltz, 1996].

Bardecki [1984] inspects continued participation in Delphi studies from a psychological perspective, and finds a high degree of similarity with the study of attitude change. He notes that "round-to-round dropout rates generally decreases as the Delphi progresses" and draws parallels to the decrease of dissonance in the panel as time goes on.

I chose to counter these concerns in different ways: explicitness of time requirements, incentives, and personal interaction with each participant. I put considerable effort into the writing of cover letters at the various stages of the study, and paid attention to arguments and insights from response behavor theory [Albaum et al., 1998]. Let me address these points in turn.

5.4.9.1. Time requirements

Although it was not always possible to estimate the time requirement of any task, I provided informed approximations, based on feedback from two proof-reading colleagues, and my own view, which I generously rounded up. I wanted to make sure that those who signed up did so knowing what they were getting themselves into.

It turned out that these apprimxations were grossly under-estimated for parts of the study. I reflect upon this in sections 8.4.3.1 and 8.3.2.3.

5.4.9.2. Incentives

Incentives (or compensations) are a popular means to increase response rate in surveys [Anderson et al., 1994, Brehm, 1994, Church, 1993, Deutskens et al., 2004, Dillman, 2000]. While several studies have found that pre-incentives increase survey response more effectively than *post-hoc* (promised) compensation [Berk et al., 1987, Sánchez-Fernández et al., 2008, White and Kaplan, 2002], surveys conducted over electronic media raise additional challenges, since incentives cannot trivially be offered during initial contact [Deutskens et al., 2004].

Hawkins et al. [1988] postulate that incentives, especially those promised at the beginning of a study, can cause a shift in motivation and increase uninformed response error due to people motivated by the incentive, but without the required knowledge to make informed statements. This is in contrast to Goetz et al. [1984], who could not identify a correlation between the use of incentives and response quality. In the context of a Delphi study, and especially with open-ended questions and when participants are selected for their skills, this problem becomes irrelevant.

The choice between material and monetary incentives has also received considerable attention [Church, 1993, Dillman, 2000]. While financial compensation seems generally preferred, Sandford [2002, cited in [Dillman, 2000]] makes a compelling case for material incentives used by creative investigators: "novelty effect can draw panellists' attention as well as lead to respondents thinking that the investigators' efforts are worthwhile and, in return, panellists are more likely to respond to investigators' questionnaires." Given the context of the Debian Project, I discarded the idea to reward participants with money for a number of reasons: (a) it would be impersonal, (b) it would be in stark contrast to the values of a FLOSS project, (c) I speculated that DCs would be more excited by the prospect of electronic toys, and (d) it seemed possible to get more value from sponsors of gadgets, as well as bulk purchases.

I solicited sponsors and was able to reward participants with gadgets of worth €250 per participant, which enabled individual compensation instead of a lottery [*cf.* Anderson et al., 1994]. At an estimated twenty hours of time to invest into the study, €12.5/hour is a low but acceptable reward, especially if the participants have a genuine interest in the study. I still made it explicit that the participants are not paid as consultants [*cf.* Turoff, 1970, p. 248], but recompensated for meeting deadlines to think and write about a topic in which they are interested.

5.4.9.3. Follow-up contacts

Deutskens et al. [2004] note that "follow-up contacts have been consistently reported as being the most powerful technique for increasing response rates," but care must be taken with timing [Dillman, 2000] and frequency, to prevent diminishing returns due to irritation of participants [Solomon, 2001].

I speculated that the use of different communication media (*e.g.* reminding people on IRC) would facilitate the task over always using the same medium for following up. In addition, I made sure that participants knew how to reach me, and encouraged them to get in touch in case of any questions (see section 5.4.6).

In this context, I considered it important to honour and explore everyone's opinion, even though it was difficult at times to get some participants back on track without offending them or discarding what they had to say. Having known the participants prior to the study helped a lot in this regard. As a result, I expected follow-up contacts that would span several exchanged mail messages, or chat discussions of considerable length.

5.4.9.4. Cover letters and behavioural theory

Deutskens et al. [2004] hypothesise that "that longer questionnaires will obtain lower response rates than shorter questionnaires, as they demand more time from the respondent [and that there exists] a negative relation between the response quality and length of the questionnaire." However, they also found mixed results in the literature.

I sent several cover letters throughout the study, which you can find in appendix C.2. I put considerable effort into these letters, often developing them over the course of several days and multiple iterations of proof-reading and feedback from up to four colleagues, who were assisting me throughout the process. This was in part motivated by Brehm [1994], who found that "the

greatest gains in the response rate came from the letter, not the pen or the dollar. In fact it is the letter — the prior contact — that appears to stimulate change in the quality of the respondents' answers." I regarded the letters as a vehicle to excite participants. The response behaviour theory principles identified by Albaum et al. [1998] proved helpful.

Oppenheim [2001] suggests to keep initial contact short, and yet I considered it necessary to provide as much background information as possible, and to answer as many questions about the research and the approach as possible, up front. I will argue this decision further in section 6.1.3.

5.4.10. Bias

The Delphi process is a mode of communication around a central entity, the facilitator, who is responsible for keeping up the anonymity of the respondents. Since the participants do not know each other's identities and lack a sense of the group as a whole, no communication between individual members of the panel takes place (the wide-band Delphi method is the exception, see section 5.3). All exchange happens via the facilitator, who may relay messages between participants, but usually instead collates or summarises them before redistributing the result to the panel [Dalkey, 1967]. Therefore, the facilitator is in a position where s/he can consciously or inadvertedly influence and skew the communication, as "the methodology provides a somewhat more convenient means for deception" [Murray, 1979, citing [Welty, 1972]].

Cyphert and Gant [1970, cited in Linstone [1975]] have shown in an experiment that false information purposely introduced into the second round of a Delphi was accepted by the panellists and caused distorted responses. Linstone [1975] adds that "the Delphi process is not immune to manipulation or propaganda use" and that "the anonymity [...] may even facilitate the deception process."

The facilitator has to consider several dangers that could threaten the value of the panel and the usefulness of the responses s/he receives. Linstone and Turoff [1975a] alert their readers to the following reasons of failure (amongst others):

- Imposing facilitator views and preconceptions of a problem upon the respondent group by overspecifying the structure of the Delphi and not allowing for the contribution of other perspectives related to the problem

- Poor techniques of summarizing and presenting the group response and ensuring common interpretations of the evaluation scales utilized in the exercise
- Ignoring and not exploring disagreements, so that discouraged dissenters drop out and an artificial consensus is generated

It is impossible to rule out all bias in a study like this, which makes it all the more important for the researcher to be aware of the issue. To counter problems related to errors (or intentional misrepresentation) on the part of the facilitator, I kept meticulous records of all communication with the participants, and identified sources of bias in section 8.3.1, and specifically bias in the data processing techniques in section 8.3.1.2. In case of doubt, this information allows for later reproduction of the study at all stages. This level of transparency also allows future researchers to avoid any errors I have made.

5.5. Sending e-mails to Debian contributors

As I discussed in section 2.2.7, e-mail is a ubiquitous medium in the Debian Project. Contributors to the projects are experienced users of the medium and know how to deal with large quantities of e-mail, often by scanning messages instead of actually reading them, and by making ready use of the delete key. To effectively reach a larger number of contributors of the Debian Project by e-mail requires a number of considerations to be made beforehand.

All e-mails sent to multiple recipients as part of this study are shown in appendix C.2.

In section 8.4.3.2, I reflect upon the decision to use e-mail for the study.

5.5.1. Avoiding the bulk-mail character

Messages addressed to a group of people are less likely to be read than messages adressed to one personally, mainly because it takes longer for a reader to establish whether a message is relevant and/or interesting if s/he is not the sole recipient or the message personally appeals to the reader. E-mails with so-called "bulk character" are frowned upon: the Internet provides a gateway for almost instantaneous distribution of a message to tens or thousands of recipients at virtually no cost. Therefore, an e-mail that has obviously been mass-distributed appears to be worth almost nothing to the recipient at first and will likely end up unread in the trash. E-mail spam is an extreme form of such bulk e-mail distribution.

When approaching a group of people with the same concern, most of the messages' contents will be similar, if not equal. Hand-editing is viable when the number of recipients is low, but quickly grows unmanageable and becomes a source of errors. For this study, I thus had to find a way to distribute similar e-mails to dozens of people without arousing suspicion that I was delivering them in bulk.

I ended up using a very simple mail-merge toolset written specifically for this study, because I could not find an existing tool for the job. The toolset operates on a template e-mail message with defined slots as input, which it instantiates once for each recipient. Recipients are stored in separate files, containing variable-value pairs which are filled into the template slots during instantiation. The output is a complete Request For Comments (RFC)-5322-compliant[16] e-mail message ready for transport. Please refer to appendix C.3 for more information about the toolset, which is available under a Free licence.

In addition to the template slots, which allow for personalised messages to be sent, the mail-merge toolset also allows for per-user attachments. I used this feature to attach relevant, additional information to the messages I sent, rather than linking to a web page, which would have required readers to be on-line to gather the information. The toolset uses the `mime-construct` programme to assemble proper MIME-formatted messages.[17]

Finally, the toolset digitally signs each message as an additional step to remove the bulk-mail character, and to add credibility to the messages.

5.5.2. Dealing with spam filters

Unsolicited bulk mail — spam — is a serious problem, especially for members of FLOSS projects with their archived mailing lists, public bug trackers, and public version control systems (VCSs), since all those provide feasts for e-mail address harvesters. Powerful spam filters help alleviate the problem and create a new impediment: false-positives, legitimate messages which are misclassified as spam and thus never get read. Especially messages with bulk-characters are prone to misidentification, even though their content may not be unsolicited at all.

To decrease the chance of false-positives, I took the following precautions:

- full standards-compliance of the messages;

[16] RFC 5322 (the replacement to RFC 2822) standardises the format of Internet e-mail messages.
[17] see RFCs 2045 and 2046, which define the MIME standard.

- tests against common spam filters;
- use of a proper mail transfer agent, instead of a bulk mailer.

Since I also employ a very thorough spam filter, I took extra care not to discard replies to my messages. I created a new e-mail address for the study and exempted messages to this address from being scanned by the spam filter. The address was never published and received no spam at all.

5.6. Deciding against a mock study

Researchers conducting a survey-based study — and in this context, the Delphi method is to be counted as such — should perform a test study (also often referred to as pre-test or mock study), which is "the only way to evaluate in advance whether a questionnaire causes problems for interviewers or respondents" [Presser et al., 2004]. Conversely, the "universally acknowledged importance [of pre-testing] has been honored more in the breach than in the practice, and not a great deal is known about many aspects of pretesting, including the extent to which pretests serve their intended purpose and lead to improved questionnaires" [*ibid.*].

The most prevalent form of pre-testing is known as "conventional pre-testing", in which "interviewers receive training [...] for the main survey and administer the questionnaire as they would during the survey proper. After each interviewer completes a handful of interviews, response distributions may be tallied, and there is a debriefing in which the interviewers relate their experiences with the questionnaire and offer their views about the questionnaire's problems" [Presser et al., 2004]. Other pre-testing strategies are "behaviour coding" and "cognitive," both of which require very elaborate subject analysis which is often not possible in a remote context [Converse and Presser, 1986].

The conventional mock study usually takes the "undeclared" form, in which subjects are not told that they are participating in a mock survey but led to believe that they are part of a real study.

Undeclared pre-testing has a number of drawbacks and problems. First of all, it is one-sided because only the interviewer knows about its real purpose. As such, evaluation of the questions is only possibly by looking at and correlating the answers received, or questions that arise among the participants. The participants are not encouraged to approach the questioning itself critically, or provide information about the interpretive process engaged by the questions. In so-called

"participating" pre-tests, subjects are told that this is a practise round and asked to provide feedback on the questions themselves, in addition to answering them (although the latter is often optional).

If pre-tests are declared an important part of the study as a whole, then participating in a pre-test is an important contribution. From the perspective of the participant, who defines "the study" as the part to which s/he is exposed — the test itself, however, the pre-test is not "the real thing" and thus deemed secondary and less important. In its own way, this goes against recommendations to convey to respondents the feeling that their participation is an important part of the study [*e.g.* Salkind, 2006].

Furthermore, I compensated the participants of the Delphi panel for their time and effort. In a voluntary context (the Debian Project), compensations help heighten the professionalism and importance of an endeavour, causing participants to take their role more seriously and prioritise their involvement over purely voluntary tasks [*cf.* Berdou, 2007]. I did not have enough funding available to compensate pre-testers for their efforts to ensure that the tests would be conducted under the same stipulation as the live study.

In the context of this study and the diversity among members of the Debian Project, there is also no guarantee that test subjects' understanding and correct interpretation of the questions guarantees the same level of comprehension among the subjects of the live study. As Presser et al. [2004] note, "surveys of special populations [. . .] pose special challenges for pre-testing." Even an elaborate mock study would not have guaranteed flawless questions, or their precise comprehension by the subjects. Heraclitus is credited with saying something akin to "you cannot step into the same river twice", which seems appropriate in this case.

Instead of a conventional pre-test, I chose to consult with a number of colleagues, some of whom were potential or real candidates for the panel at the time, and asked them to provide feedback on the questions I would ask. The colleagues were asked independently (rather than addressed as a panel), but the strategy is nevertheless reminiscent of the expert panels discussed in Presser and Blair [1994]. I was aware of the risk associated with possibly drawing real candidates into the design phase of the study, but the colleagues were themselves researchers and well aware of my endeavours anyhow. They affirmed that they would be glad to help out, and capable of participating nonetheless.

Another reason for skipping a pre-test round lies in the close contact between myself and the candidates of the panel, which offered the possibility to all participants to obtain clarifications after receiving the questionnaire, and thus considerably alleviated the need for pre-testing. As active contributors to the Debian Project, we are in close collaboration on a daily basis, spanning

multiple media. Anticipating questions of clarification in response to my questions, I frequently communicated my ready availability in IRC channels, by e-mail, as well as the telephone in case of any questions. I explicitly asked participants to refrain from contacting me in public forums, though, due to the anonymity requirements of the Delphi study.

Chapter 6

Research approach details

In chapter 5, I introduced the Delphi method, and argued for it's suitability for the study at hand and the context of the Debian Project, which I discussed in-depth in chapter 2. I deemed the Delphi approach suitable to research in a FLOSS project in section 5.2. As the Delphi method is more of an idea and a box of tools than a cookbook method, and has been implemented and used in sundry different ways, I devoted section 5.4 to identifying those tools and variables, and designed the approach for this study.

In section 4.1, I established the philosophical basis for my research, which is chiefly of the *interpretive* nature. The approach I detail in the following sections is hence *qualitative* and *exploratory*, which I conducted *in the field* to create knowledge by *induction*.

In the following two sections, I report on the exact process, providing detailed information about the various steps at different stages, to allow for the reproduction (or replay) of the study. Furthermore, by providing an account of the process, I seek to facilitate the use of the Delphi method in a FLOSS context for future researchers (*cf.* section 4.3.2). Since the approach includes steps that are not necessarily part of the Delphi method (albeit related), such as sampling, those are described as well.

As noted in section 5.4.10, bias is an important consideration in a Delphi study — and qualitative research *per se*. It is practically impossible to rule out bias. Given my previous involvement with the participants, as well as the core topics related to workflows and tool adoption in the Debian Project, it was especially important for me to be aware of possible sources of bias throughout the study. I took meticulous care to (1) identify all potential sources of bias in this thesis, and (2) document all steps and make all records available. In case of an influence of bias

Date	Activity
24–30 Sep 2008	sent calls for nominations to 162 selected peers
8 Oct 2008	deadline for nominations (incl. extension)
12 Oct 2008	sent call for applications to nominees
27 Oct 2008	deadline for applications (incl. extension)
31 Oct 2008	sent declining/acceptance notices

Table 6.1.: *The time-line of the participant selection process*

threatening the validity of the results of the study, those precautions should make it possible to (1) compartmentalise the bias and identify its reach and effect on the results, and (2) aid future researchers in avoiding the same trap.

6.1. Details of the participant selection process

To avoid convenience sampling and any selection bias from my side, I used the following steps to amass the Delphi panel for the study at hand:

1. identify project members who take part in team efforts, or who otherwise cooperate with several others involved in the project;
2. ask these people to nominate colleagues whom they deemed to have insights into the adoption behaviour of Debian Contributors (DCs), along with a short reason for each nomination (snowball sampling); I explicitly mentioned the possibility of self-nominations.
3. write to each of these nominees and ask them to participate in the discussion, and to answer a number of questions;
4. select the panel members according to the answers received.

In the following sections, I shall elaborate on each of the four steps. Table 6.1 on the current page shows the time-line of the participant selection process.

6.1.1. Picking nominators

I could have easily populated the Delphi panel with people I know. However, as a workflow pioneer in the Debian Project, my selection would have been strongly biased towards those

with whom I've discussed the issues before; I would have been unable to assemble a balanced panel, but instead would have ended up with a group of people in my immediate circle.

The task was thus to ensure that the panel could include people I barely knew if they were knowledgeable in the field, and that the panel should be representative of the major packaging-related camps in the Debian Project.

I opted for a snowball sampling approach, in which I would ask a broad group of people to name those deemed knowledgeable in the matter. I obtained the list of people to ask by scanning:

- several mailing list archives (mainly `debian-devel`, `vcs-pkg-discuss`, `dpkg-dev`, and `debian-mentors`)
- logs of Internet relay chat (IRC) channels (`#debian-devel` and `#vcs-pkg`)
- the `qa.debian.org` package maintainer database
- notes from various meetings and discussions at conferences

and assembled an initial list of 143 well-connected, active contributors, who have driven or been subjected to workflow changes in their teams, who are part of well-functioning teams in general, or who have participated substantially in discussions on improving the cooperation between contributors to the project.

6.1.2. Calling for nominations

My intent was to ask each of the people identified in the previous step for names of people who they thought would have interesting things to say on the topic of my research. Specifically, I asked them to nominate "people who have interesting things to say on the topic of why some techniques have been widely adopted while others have not, because they work in teams, employ cool tools, have good reasons to keep using the same old tools as always, or for some other cause" (see listing C.1 on page 336).

These criteria are soft and each nomination could be considered biased. However, to ensure that the respondents knew what kind of nominations I sought, I had to avoid being too rigorous when specifying the criteria. Instead, I opted for a formulation that did not require much thought by the reader, but ideally elicited an immediate reaction. Since this snowball sampling approach shifts the sampling body from the set of people I could find through on-line searches to the set

of people containing well-connected and experienced contributors who may not interact with the media I searched, I considered it appropriate.

To prevent quick, thoughtless responses, I wanted everyone to explain why each nominee had been chosen. I encouraged correspondents to nominate themselves if they were interested in the study, because by the criteria I used to identify them, they could potentially be good candidates. I also wanted to prevent giving the impression that they were only stepping stones. E-mail was the natural choice as medium for this call for nominations (*cf.* section 2.2.7).

To increase response rates, I wrote one or two personalised sentences in which I explained to each recipient why s/he was chosen, heeding to common advice to make the respondent feel like an important part of the study Salkind [*cf.* 2006]). Even though the rest of the message I sent was the same for everyone, this personalised hook removed much of the bulk-character of the e-mail (*cf.* section 5.5). Where appropriate, I added a personalised closing paragraph, *e.g.* wishing someone well if I knew s/he was sick, or commenting on their work if related. In addition, I also made an effort to greet the recipient in his/her own language, and used the nickname in cases where I knew the recipient preferred it over the first name. The majority of the replies I received suggest that these measures were successful and only very few recipients identified the bulk character of the call for nominations.

Furthermore, I wrote the message to be engaging, using an active voice and cutting straight to the chase, but avoiding jargon throughout to improve the clarity and comprehensibility of the text. To draw the reader in, I appealed to his/her motivation to work on Debian and mentioned the benefits of the endeavour to the project at the very start [*cf.* Albaum et al., 1998]. I tried to estimate the time requirements to reply, and stated the deadline at the beginning and again at the end of the message, encouraging readers to let me know if they knew they would not be able to make it.

Since I was asking people to refer others, I explicitly stated the consequences of nominating someone to alleviate concerns about signing people up for something they might not want. Several respondents explicitly appreciated that.

Finally, to increase the transparency and help answer further questions, I noted that the endeavour is part of my research and included a link to the accompanying website for more information. I also attached the most relevant and interesting bits from the website for people reading the message off-line.

Listing C.1 on page 336 is the template of the message I sent to the recipients. The message had been proof-read by six colleagues, four of them with a good grasp of the Debian Project, while two only read for grammatical correctness and clarity. Two of the four with a good grasp were likely to become candidates for the panel themselves, and agreed to supervise the selection process while regularly participating therein. We explicitly considered the problem of the two roles conflicting. However, short of extra time involvement, we did not identify any dangers therein. In hindsight, it proved very valuable to have their feedback, especially in later rounds.

As an additional measure of care, I sent the message to only 14 of the 143 recipients, whom I knew very well (and who would be merciful in case of problems), on 24 September 2008. After soliciting feedback from them to make sure that the text was clear and I had not overlooked any problems, I posted the remaining 129 messages two days later.

By 27 September, I had 29 e-mails with nominations. From the set of nominated people, I extracted three additional recipients and sent them the (semi-personalised) call for nominations. Patton [2001] calls this flexibility in the sampling approach "opportunistic". By 30 September, 43 respondents had sent in their nominations. I identified a further 16 recipients and sent them the (semi-personalised) call for nominations in a final round.

In total, I sent 162 calls for nomination messages. By the deadline on 5 October, 69 people had responded. I sent a reminder to the 93 others on 6 October, and gave them a two day deadline extension. In response to the reminder, another 29 sent in their nominations by 8 October, totalling in 98 responses (60.5% response rate) and 429 nominations (including self-nominations if they provided a reason).[1]

6.1.3. Soliciting applications to the panel

Using a simple sqlite3 database, I identified nominees with three or more nominations, weighing them all equally: a self-nomination like "I am interested because … and would like to be part of the discussion" counted as much as a nomination like "Ashley J. Developer is a strong proponent of … in the team, he would be a good candidate". However, I did not count self-nominations without a reason ("I am interested, let me join"). This process yielded 48 nominees, of which I removed 10 I knew were indisposed, had previously expressed disinterest, or stated that they didn't feel comfortable talking about the issue.

[1] The response rate could have been higher, had I used "snowball sampling" instead of "snowballing" (a sexual practice, as I've come to learn) in the text; several respondents alerted me to this.

From the 50 people who had two or fewer nominations, I manually selected 5 whose nominations made them particularly interesting candidates (see section 8.3.1). The final set spanned 43 people.

I have already argued that members of the target group are used to dealing with large amounts of e-mail. Hence, I deemed it appropriate to send each an e-mail asking him/her to apply to the panel by answering a set of questions. This also allowed me to determine a person's interest in the study, and his/her time availability prior to selecting the panel. Again, I asked the same six colleagues as before to proof-read the message, and incorporated their valuable feedback. Listing C.2 on page 337 shows the final result, which I sent to the 43 nominees I selected earlier.

As [Brehm, 1994] notes, the initial contact is paramount in motivating participation, and Oppenheim [2001] suggests to keep the initial contact short and concise. Nevertheless, the e-mail I sent was rather long (~140 lines[2]). I deliberately chose to be explicit and provide enough information, suspecting that the recipients would appreciate details when deciding whether to participate or not. The decision for detail and length was in part motivated by the opinion of three peers: due to my involvement with the Debian Project, I have the advantage that people would read my text anyway, which meant a unique opportunity to effectively convey the necessary information in an enticing manner. I reflect upon this choice in section 8.4.3.2.

With the message, I included a short summary of the Delphi process, a time frame of the entire study, and mentioned the rules of compensation. The message also highlighted transparency as a goal of the study. I provided a visual anchor in the middle of the message text to help people skip the informational content to get straight to the important part of the message, where I ask the questions.

I then explained why the respondents would have to answer a set of questions to be considered for the discussion panel and repeated the deadline. I asked potential participants to be discreet about the endeavour due to the anonymity requirements, and I outlined the motivation for that. Just before presenting the questions, I again stated the objective to help the respondents focus.

[2] In accordance with established guidelines, a line of body text in an e-mail message is defined to be no more than 68 characters.

6.1.4. Questioning potential participants

The Delphi discussion panel was to be comprised of 16–20 people (see section 5.4.4), selected for maximum diversity in the panel along the four strata identified in section 5.4.3. Part of the panel selection was thus to place each candidate at the appropriate position on the strata. Rather than to gather background information on each candidate, I chose to obtain the necessary information directly from each candidate.

Initially, I had considered a Likert-style approach, using a scale of 5 or 7 items representing each stratum. However, in eight short tests with fellow Debian Project members, I found that the results were not interesting, and that respondents did not appreciate the constraints of such a scale (*cf.* section 8.4.3.4). Despite the obvious benefit of being able to quantitatively analyse and use the answers, I opted for a qualitative approach instead and solicited free-form answers.

I sought to determine more specifically the nature of each candidate's work for the project, and collaboration within the project. The questions corresponding to the four strata were part of this set (see below). I found that I could not translate the fourth strata directly into unambiguous questions and ended up having to split it across two questions:

1. How much of your work is in teams, and how much do you work alone?
2. How many of your packages are related to each other, and how many are unrelated?
3. Do you use the same packaging tools and techniques for most of your packages, or do you use (or deal with) various approaches? How many of them, approximately?
4. To what degree do you experiment with new packaging tools and techniques?
5. Are you more interested in improving the way in which you work, or do you mainly concentrate on getting your work done?

I also used the opportunity to obtain additional information about the respondents' involvement with the Debian project and included a number of other questions, such as "what is your role within the project?", "how many hours per week do you spend on Debian?" or "with how many people have you collaborated in the past year?". The complete message is shown in listing C.2 on page 337.

At the end, I asked the respondents about the amount of time per week they could devote to the study, and sought permission to publish their responses to these questions.

By the deadline, 27 of the 43 people had responded to all of the questions, while one had excused himself due to lack of time to participate. I sent a reminder to the remaining 15 and gave them another two days to respond, yielding five more responses and two declines. A minor technical problem, someone on vacation, and the final freeze of Ubuntu 8.10 caused another four responses to arrive with a delay, but still in time. In total, 36 respondents applied to the discussion panel, and the response rate was 90.7% with only three candidates not replying at all. A small number replied with questions, which I answered immediately in each case.

To keep up the pace and rhythm, I replied to each respondent, thanking him/her for the time and giving a peek of the next steps. The message is shown in listing C.3 on page 340 (*cf.* section 5.4.9).

6.1.5. Selecting the panellists

For the panel selection, I created a knowledge resource nomination worksheet (KRNW) [*cf.* Okoli and Pawlowski, 2004] and populated it with the nominees who had replied to my questions. For each candidate, I recorded six data: a pseudo-percentage position on each of the four strata according to which I wanted to select, the estimated number of people with whom the candidate regularly interacts, and the estimated available time for each round of this study. The result is shown in table 6.2 on page 114.

Given the general level of diversity among members of the Debian Project, selecting for diversity is a nigh-impossible task, much harder than selecting for a given criteria. I chose to deal with this challenge by abstracting the problem into a quantitative space, knowing full well the impossibility to translate qualitative into quantitative data.

The selection is based on a *tesseract*, a theoretical, four-dimensional "hypercube", whose axes correspond to the four sampling strata, ranging from 0 to 1. For each candidate, I had to determine the coordinates within the cube, where a value of 1 in any of the four columns meant that the candidate (a) is a team player, (b) has a uniform set of tasks, (c) uses uniform tools, (d) is interested in workflow improvement. Figure 6.1 on the facing page depicts a simplified, three-dimensional cube covering only the first three of the strata; a four-dimensional cube cannot easily be drawn on paper.

Aware of the general uncertainty inherent in the translation, I restricted myself to single significant figures. To minimise the error, I translated each candidate's responses to the strata-related questions into numbers three times, with two days of time between each iteration. I averaged

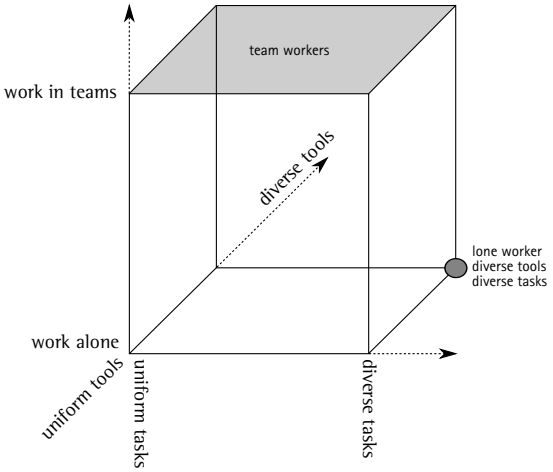

Figure 6.1.: *A three-dimensional cube, exemplifying the stratified selection process. The axes correspond to strata, and individuals are positioned inside the cube according to their position in each strata. The circle represents the position of a pure lone worker who works on diverse tasks with diverse tools. The top plane represents the set of team workers, and serves only the purpose of clarification; planes are not used in this research.*

the three numbers. I also asked two colleagues experienced with Debian packaging to do the same. With their results, I compared each number to the average I obtained and revisited a coordinate when either of the two's value was 0.2 or more away from my number. This resulted in a small number of corrections, which were rarely larger than 0.1 and never higher than 0.2.

To select the panel members from the list of candidates, I went on to label the 16 corners in ascending binary order: the origin, which corresponds to lone workers who use diverse tools to achieve diverse tasks and are purely interested in getting work done (0,0,0,0) became A, while the J corner (1,0,0,1) would be more populated with team workers who use diverse tools for diverse tasks, and are interested in how to improve their workflow. I used the first sixteen letters of the alphabet, A–P, and mapped them in increasing order of the corner coordinates when interpreted as binary numbers: (1,0,0,1) is a decimal 9, which is the tenth number and thus translates to J, the tenth letter (*cf.* table 6.3 on page 115).

For the record: I identified my own position to be close to the B corner (0,0,0,1), since I work in and with too many teams to be a real team player, use diverse tools for diverse tasks, and have a strong interest in workflow improvement.

In a first step, I picked the obvious "corner cases" from table 6.2 on the following page: candidates whose coordinates were no further than 0.2 from the extreme. The second step allowed for some fuzz, and in the third step, I was trying to find the best representative for yet unfilled

	Position on strata					
Candidate	Team	Tasks	Tools	Interests	People	Time
1	0.9	0.1	0.9	0.1	8	"a couple of hours"
2	0.4	0.7	0.9	1.0	5	1h
3	0.6	0.1	0.6	0.5	15	"enough"
4	0.6	0.0	0.9	0.0	20–50	1h
5	0.9	0.3	0.9	0.3	4	1+h
6	0.5	0.1	0.4	0.7	5	1h*
7	0.3	0.3	0.3	0.4	30–40	1+h
8	0.9	0.8	0.9	0.9	15	1h
9	0.4	0.4	0.9	0.9	10–15	"enough"
10	0.2	0.6	0.8	0.4	4	1h
11	0.4	0.5	0.4	0.6	10	1–2h
12	0.6	0.1	0.4	0.4	8–13	1–2h
13	1.0	0.8	1.0	0.6	20–30	1–2h
14	0.3	0.8	0.7	0.4	10	1+h
15	0.8	0.9	0.9	0.3	20	1h
16	0.9	0.1	0.8	0.4	4–5	1–2h
17	0.4	0.1	0.6	0.6	5	1+h
18	0.5	0.3	0.9	0.7	10	1–1.5h
19	1.0	0.9	1.0	0.3	5	1+h
20	0.0	0.8	0.8	0.7	0–10	1+h
21	0.0	0.4	0.6	0.1	1	"a couple of hours"
22	0.8	0.8	0.4	0.5	50	"some hours"
23	0.7	0.7	0.7	0.6	5–6	2+h
24	0.7	0.2	0.2	0.8	30+	1h
25	0.4	0.6	0.4	0.3	5–10	2–3h*
26	0.7	0.4	0.9	0.8	4	"plenty of time"
27	0.2	0.2	0.5	0.2	5–10	1–2h
28	0.5	0.5	0.5	0.6	5–10	2h
29	0.9	0.4	0.8	0.1	5–10	1h
30	0.8	0.6	0.8	0.8	12	5h
31	0.0	0.1	0.9	0.9	0	4h
32	0.0	0.2	1.0	0.9	0	4–5h
33	0.1	0.8	0.9	0.2	5	1–1.5h
34	0.8	0.0	0.8	0.0	20	1+h
35	0.8	0.8	0.7	0.4	5	3h*
36	0.6	0.4	0.3	0.4	10	2h
37	0.8	0.6	0.2	0.9	20	2–3h

Table 6.2.: *The KRNW used to select the panellists. The four strata are (0 to 1 in each case): lone worker* vs. *team player; diverse* vs. *similar tasks; diverse* vs. *similar tools; interest in getting work done* vs. *workflow improvements. The last two columns are estimated number of regular collaborators and available time per round, as estimated by the candidate him/herself. Time entries with an asterisk mark tentative estimates because of pending events with unforeseeable consequences for the candidate's availability.*

		Selection step			
Profile	Corner	1	2	3	Final pick
0000	A			**7**	7
0001	B			**6**	6
0010	C			**4**	4
0011	D	31,32	**9**		9
0100	E			**25**	25
0101	F			**11**	11
0110	G	33	**14**		14
0111	H		**20**		20
1000	I			**36**	36
1001	J		**24**		24
1010	K	1,34	**5**,29		5
1011	L		**16**	18	16
1100	M		**22**		22
1101	N		**37**		37
1110	O		**15**,19		15
1111	P	**8**			8
xxxx	X			**18, 28, 29**	18, 28, 29

Table 6.3.: *Associating panellists with classes describing the strata extremes. The first selection step consisted of almost exact matches; in the second step, a little bit of fuzz was tolerated; the third column holds picks which are not obvious from the data, but which can be qualitatively backed up. Deciding which candidate to pick for each row (result in the last column) involved looking at the number of collaborators and time available shown in table 6.2 on the previous page. The last row X holds panellists close to the centre of the hypercube, further adding diversity separate from the strata to the panel.*

corners, often going back to the candidates' responses and gathering more information about them. Table 6.3 on the current page illustrates the selection process. In rows D, G, and K, I replaced the "obvious candidates" in a later step, due to time constraints, or low number of collaborators. I also selected three panellists close to the centre (row X) due to their diverse background, in order to bring in alternative viewpoints (*cf.* section 8.3.1). The final column shows the 19 chosen panellists.

6.1.6. Declining applications

After selecting the panel, I sent the message shown in listing C.4 on page 341 to the 18 applicants who did not get chosen for the compensated panel positions. One purpose of the message was to be polite and appreciative of the time those people had already invested into answering the application questions. Another purpose was to identify candidates who are interested enough to participate in the study without compensation. This was somewhat of a pre-emptive

step to obtain names of people who might be able to help out, or jump in for candidates who dropped out at a later stage.

Two candidates replied, and I accepted them onto the panel as backups. One of the two (#34) very clearly fit into corner K already occupied by a close collaborator of his. The other backup was #23, who was more central with a slight tendency towards corner P, and maybe O. This introduced a slight tendency towards team-maintenance and uniform tools, which I felt was passable in the context of the Debian Project; I discuss problems related to this source of bias in section 8.3.1.1.

6.2. Details of the Delphi study

With the panel assembled as described in section 6.1.5, the actual Delphi study could begin. Initially, I planned a policy Delphi approach with elements of ranking, but had to amend the strategy slightly between rounds, because the data provided by the participants was unexpectedly rich, and called for further exploration (rather than concentration into a ranking) to avoid cropping them to fit into a limiting structure.

The Delphi study took three rounds, of the following nature:

1. exploration of the domain and collecting influences
2. discussing the influences, and identifying relations between them
3. describing real-world instances of what participants considered to be the most salient influences to the Debian Project.

The study took several months, which exceeded my expectations. The extra time was necessary due to the data and the efforts expended by the participants. Table 6.4 on the next page shows the milestones.

6.2.1. The first round: brainstorming influences

In section 6.1.5, I described the selection process with which I chose 19 panellists. I sent the message shown in listing C.5 on page 342 to these panellists. Furthermore, I forwarded the message to two further candidates who had responded to the declining notice (see section 6.1.6), expressing their desire to participate without compensation. Unfortunately, I failed to make it

Date	Activity
31 Oct 2008	sent first round Delphi question to 21 panellists
09 Nov 2008	received responses from all panellists in time
19 Nov 2008	sent second round Delphi instructions to 21 panellists
08 Dec 2008	received responses from all participants
11 Dec 2008	beginning of bilateral discussion and exploration of divergences
08 Apr 2008	sent third round Delphi instructions to 21 panellists
23 May 2009	acknowledged receipt of responses and concluded the study
19 Jun 2009	sent out call for feedback

Table 6.4.: *The time-line of the Delphi study*

clear that I was not soliciting their full participation; I will return to this point at the end of the discussion of round one.

The message started out with a bit of subtle flattery to make the reader feel like an important part of the study [*cf.* Salkind, 2006]. It then gave a glimpse of the selection process, and the diversity and balance of the panel. I mentioned the compensation offer, noted that it is bound to completion of the whole process (*cf.* section 5.4.9.2), and encouraged people to step down if they were not confident to be able to follow through to the end. I also reminded the reader of the anonymity requirement.

I asked panellists to assume the position of representatives and speak for their fellow contributors to the project and the Debian Project at large. This was not a trivial decision. Several people whom I consulted during the design of the Delphi study feared that participants might feel uncomfortable speaking for anyone but themselves, or would simply not be able to step out of their own role to be able to do so. Nevertheless, the panel was not large enough to be representative of the entire project, and answers that only concerned personal experiences of the panellists would not be of much use to the study. The participants had been nominated by their peers as people who would be able to provide insights into the matter. I therefore deemed it appropriate to ask them to represent their peers, but to provide personal input if desired, clearly identifying it as such.

Due to the danger of interviewer-related survey errors, I considered it significant to be aware of my own roles in asking these questions, and conducting the study in general. Even though "the role of the interviewer in contributing to error in survey data has not been appreciated generally" [Fowler, 1993, p. 135] and "survey error is undetectable" [*ibid.*, p. 133], [Fowler and Mangione, 1990, p. 13] note that "the wording of the specific questions has a direct effect on the quality of answers" and that "interviewers do affect the response rate". However, they argue that the data collection process is more error-prone than the preparation and the conducting of

the interview. Nonetheless, I carefully designed the question I was going to ask the panellists in the first round.

I wanted to obtain a broad sense of what shapes package maintainers' adoption and rejection decisions. Initially, I planned to present the panel with my original research question: under what circumstances do Debian package maintainers incorporate new tools or techniques into their workflow?

In various discussions following presentations I gave at conferences, I found that very few people had an idea about the depth of answers I was expecting. Especially the word "circumstance" seemed to instil in people a sense of passiveness, as if circumstances linger but do not play an active role in decisions. Another word I toyed with, "factors", apparently caused some to think the exact opposite, while most could not make sense of the word's precise meaning — many suggested that factors was too overloaded a word to be meaningful in a context as critical as this study. In trying to explain what I meant, I found myself using the word "influence" repeatedly, and that seemed to be received much better.

Since my goal was an exhaustive search for influences, I needed to encourage respondents to provide multiple influences. I thus took the advice by Schmidt [1997] to ask for a minimum number of responses, and came up with the following formulation, turning the question into an instruction: Describe at least six influences to a Debian package maintainer's decision for or against a new tool or technique for his/her packaging work.

I consulted with seven colleagues (two of whom would later end up on the panel, but had agreed to provide feedback independently at this stage), and, with their help, identified numerous vague and imprecise aspects of the formulation. The instruction as written lacked context, and also left it undefined what exactly constitutes an influence.

Realising that the Delphi question did not need to be as succinct as a research question, I set out to break it into three parts, in which I would set the scene by example, establish the focus, and finally deliver the instruction. To emphasise that I was looking for common, project-wide influences — as opposed to personal experiences — I asked the respondents to identify influences they expected to recur in the project.

> Debian package maintainers occasionally find new tools or techniques that could change their packaging work. At times, they might decide in favour of one technique and adopt it; at other times, they might reject a technique without further thought.

> While deciding whether or not to adopt a tool or technique, people normally weigh many options, and take various points into consideration which influence their decision.
>
> Please describe at least six such influences you have witnessed in the Debian project, and which you expect to witness again on future occasions.

Following the question, I hinted at the kind of information that I am looking for, and tried to alleviate concerns about language and writing style by encouraging people to approach the instruction like a "brain dump," promising that I would faithfully rephrase and anonymise their answers. To those who were not comfortable with writing, I offered to call and listen.

6.2.2. Wrapping up the first round

All 21 panellists sent their responses in time for the deadline. Of these responses, 18 were exactly what I had had in mind (one respondent contacted me shortly after the start of the first round to verify that he had understood the task, which I confirmed). Three respondents took a different approach.

I realised at this point that the two uncompensated had also replied, which put me in a somewhat awkward position: they had expended effort, but were not compensated for their work, and they threatened to bias the diversity of the panel. As it turned out, however, both of their replies were among the three different approaches:

One uncompensated respondent identified six tools or techniques (instead of influences) and analysed their diffusion in the Debian Project. Studying his reply, it was obvious though that he had identified the sought influences, but presented them in an unexpected way, which made his input no less useful to me.

The other uncompensated panellist identified six new tools and techniques and summarised their origins and the discussions revolving around them when they appeared. His response constituted input I expected to become useful later in the study, but it didn't fit in with the Delphi discussion. I conferred with him, and we agreed that he would stay on the panel and resume in the next round.

Another respondent chose to identify six ways in which DCs discover new tools or techniques, and how they arrive at their adoption or rejection decisions. Exposure or discovery

being a commonnly cited influence, I realised later that his response exemplified this influence.

I followed up to every response, seeking clarification or asking additional questions in an attempt to drill further and obtain more faceted, fine-grained responses. In most cases, additional discussion with each individual helped break up unspecific, large influences into smaller, more specific components or related influences.

One respondent did not reply to my follow-up, and his response was very brisk and ambiguous in places. Suspecting scheduling difficulties, I approached him to find out whether he would be able to dedicate more time to the study in a future round, or whether I should offer his compensated panel position to one of the two uncompensated participants. In the e-mail exchange that followed, the cause was identified to be a simple misunderstanding, and the participant elaborated on his earlier reply and re-qualified the response for use in the next round.

6.2.3. Preparing the second round

Overall, the responses were overwhelming, both in terms of quality and quantity. The goal of the second round was to discuss the broad set of influences that resulted from the first round. I wanted to let the participants argue against influences they did not agree with, identify links between the comments made by the panellists, and provide additional information wherever needed.

I entered the second discussion round with around 3,500 lines of responses.[3] To manage this vast amount of text, I attached keywords to paragraphs and sentences that identified influences. I did not design a taxonomy for these keywords, but instead used the authors' own words or *ad-hoc* keywords that came to mind, with the intent to consolidate the set later.

One of the two non-compensated respondents' replies was not suitable for inclusion in the second round. The other's responses I carefully analysed and found that they were almost entirely overlapping with what the remaining panellists had said, which allowed me to include his replies. This allowed me to enter the second round with two backup candidates, still anticipating possible participant drop-out.

[3] The line-width was capped at 80 characters.

I used the `vim` editor for all of the data processing. I had briefly investigated other data mining applications and approaches but in the end opted for a tool I knew very well, which allowed me to keep an overview of the data, and which didn't require me to expend any time learning it.

This process yielded a total of 104 keywords. Using a concept-mapping approach [*cf.* Novak and Cañas, 2006], I identified 40 groupings of keywords that applied to the same underlying concept, using the respondents' descriptions to determine the similarity, instead of the keyword semantics. For instance, the two keywords "simplicity" and "complexity" seem to relate to the same underlying concept, when looking only at the words themselves. In the context of the arguments put forth by the panellists, however, it the two referred to disjoint ideas; I thus placed them into separate groups. I labelled each group with a collection of the most salient keywords or concepts the group represented, but expended no further effort on the titles.

For each of the 40 groups, I then collected the corresponding quotes from the responses, and added them to the group together with the author names. When multiple keywords were attached to the same paragraph or sentence, I made a note about the cross-references to preserve those relationships. During the processing, 14 of those groups merged naturally with others to yield a set of 26 groups, each containing a collection of related quotes. Table 6.5 on the next page lists those groups and the number of lines of text contained in each at the end of the round.

Looking over the 26 groups, I found myself wanting to fold groups that seemed alike together. Schmidt [1997] suggests to use up to 20 categories for a ranking Delphi approach, and even though I had decided not to develop a ranking in the second round of the Delphi study, 26 felt too large.

I invited three colleagues to an exercise known as "card-sorting" [Rugg and Mcgeorge, 2005]: given 26 pieces of paper, I asked each to arrange them according to their own will. I made it clear that there was no need to assign every piece of paper to a group, or to reduce the group, but encouraged an "anything-goes" approach. I did emphasise that the actual meaning of the words on the paper in the context of my study wasn't as important as their gut feeling when arranging them. The result of the card-sorting was a strong indication that groups such as "Need, fills a niche, scratches an itch" and "Motivation, curiosity, pragmatism", or the pair of "Aftermarket, derivative tools" and "Popularity, consensus, critical mass, support" were closely related. Altogether, there were five such pairs and two triplets, consolidating the number of

Title	Lines
Exposure, visibility, availability	117
First impression, gut feeling	72
Need, fills a niche, scratches an itch	60
Perceived advantage, cost-benefit	104
Quality benefits	67
Productivity, efficiency	108
Sustainability	191
Language chosen	30
Aftermarket, derivative tools	72
KISS, simplicity	33
Genericity, flexibility, extensibility, adaptability	141
Unix philosophy, well-defined tasks, modularity	39
Transparency	50
Cargo cult	42
Heritage, lifetime record, prejudice against developers	71
Implementation, technical excellence, elegance, aesthetics, feels right, fits	162
Documentation, examples	71
Motivation, curiosity, pragmatism	49
Complexity, conceptual change, learning curve	117
Compatibility, revolution vs. evolution, gradual adoption	155
Inertia, never touch a running system, laziness	35
Network effects, team adoption	64
Convergence, reuse, consistency, diversity	46
Recommendations, advocacy, publicity, marketing, peers, lead users, buzz	252
Popularity, consensus, critical mass, support	111
Top-down, integration with tools, infrastructure support, formal documentation	214

Table 6.5.: *The 26 groups that emerged during the first-round response processing.*

groups to 17. The Delphi approach to card-sorting used by Paul [2008] did not seem reasonable in this context; I discuss its shortcomings briefly in appendix C.1.2.

Before fusing any groups, however, I proceeded to weed out quotes that made the same point, collated statements, and sorted the resulting collection into a somewhat coherent order. This process required a lot of care to minimise the introduction of bias. In most cases, one statement clearly included the other, and I could simply leave out the latter. When both statements brought additional perspectives to the same point, I collated them. I kept records of which statements were used where so that I would be able to unmerge them later, in case of any doubt.

Quite often, the remaining statements in a group had a similar cross-reference to another group, or related groups ended up condensed to merely 2–3 statements in each, both of which suggested the merging of groups.

Title	Lines
Exposure, visibility, marketing, publicity, availability	77
First impression, gut feeling, heritage, prejudice	117
Motivation, curiosity, need	55
Perceived advantage, cost-benefit, efficiency, productivity	132
Quality, transparency	79
Sustainability, future-proof investment	91
Genericity, flexibility, adaptability, modularity, KISS, Unix	113
Technical excellence, elegance, aesthetics, rightness feeling	91
Documentation, examples	55
Complexity, conceptual change, learning curve	90
Compatibility, gradual adoption, inertia, laziness	93
Network effects, team adoption, cooperation	44
Recommendations, advocacy, peers, lead users, buzz	113
Popularity, standards, critical mass, support, derivative tools	71
Top-down diffusion, infrastructure support, formal docs	115

Table 6.6.: *The final set of groups of statements for the second round.*

Between the card-sorting exercise and processing the responses, I reduced the number of groups to 15, which are shown in table 6.6 on this page, along with the number of lines of statements within each. In collating and grouping, I condensed the 3,500 lines of combined responses to about a third, or 1,300 lines.

I involved two colleagues into the process, who had previously agreed to supervise my study. To minimise the chances of leaving out important statements, and to prevent my own bias from colouring the selection, I kept them updated on the process, and asked them to look over the final summary. Both confirmed that they had not identified a problem with completeness or bias.

It occurred to me later that the 15 groups appear in an order slightly correlated with the five stages of Rogers' innovation-decision process. This realisation formed the basis for the structure of the results I develop in section 7.1.

The processsing took twice as long as expected and I was forced to delay the second round by five days. I sent an appropriate message to the panellists to inform them of the delay while keeping up the pace.

6.2.4. Round two: discussing influences

The purpose of the second Delphi discussion round was to enrich the data by seeking further qualifications from the panellists, finding out with which statements participants commonly agreed, and identifying discrepancies between the panellists' judgements.

The message I sent to the panellists to start the second round is shown in listing C.6 on page 344. It started out with a bit of background information and gives reasons for the delay.

After I introduceed the "15 chunks of related statements" and give an idea of how these came to be, I presented the task for the next round. I phrased the instruction as follows (original emphasis[4]):

> Go through and read this collection, and comment whenever you disagree or would like to set something straight. If you leave a statement uncommented, I will assume that you *generally* agree.

The instructions encouraged the respondent to read critically because his/her agreement would be assumed by default. By asking only for qualifications and refutations, I intended to broaden the data gathered in the first round without burdening the participants excessively.

Anticipating a greater time requirement than in the first round, I extended the deadline to 12 days, instead of the 7 originally planned. In the message to the panellists, I apologised for the size of the message and the additional time requirement, noting that I had not expected it.

Before presenting the chunks of statements, I briefly discussed a couple of aspects of these data, their creation, and their presentation:

- As the final dataset was roughly a third of the size of the combined responses I received, participants were unlikely to find their statements verbatim. I assured them that I worked with utmost care not to leave out anything, but encouraged the respondents to let me know in case they missed anything.
- In the interest of brevity, I refrained from glueing statements together, or even reformulating them, which would have taken a lot more time and introduced further delays. I thus informed the respondents that statements made in the first person are statements by another panellist and do not necessarily reflect my opinion.

[4] For lack of formatting in plain-text e-mail, I used the common convention of _underscores_ to emphasise the word.

- Selecting statements, and even reordering, can introduce bias on many levels. I identified my genuine interest to avoid bias and explicitly state that I did not amend or engineer a statement to make my own point.
- Even though I had put a great amount of effort into anonymising the responses, I discouraged the panellists from trying to guess any statement's author's identity.
- The 15 chunks existed to make the whole collection manageable and readable. I explicitly stated that they do not represent or hint at a higher-level structure. In case a panellist sees a structuring, I encouraged him/her to share his/her ideas.

The use of fold markers grew out of the size of the data set. Fold markers allow supporting editors to collapse chunks of text, which can help improve the presentation, allow the reader to maintain an overview of the whole, and make it possible to concentrate on each chunk in as much isolation as desired.

Since the chunks were purely structural in the context of the message, and respondents needed to be able to easily switch between or reference across them, I included all chunks in one large message, rather than to split the chunks into separate attachments.

Drawing inspiration from guidelines for ranking-type Delphi studies (which are the only Delphi studies which deal with collections of statements or concepts), I considered randomising the order of the chunks, but decided to leave them in their original order because it would make it easier for the panellists to follow along. Similarly, I decided to keep the chunks' titles, reasoning that the potential influence or confusion that could stem from single words in the title of a section would be outweighed by the potential gain in terms of manageability and overview of the large dataset.

6.2.5. Wrapping up round two

Given a four-day deadline extension, all participants sent their second round responses in time. Only one participant did not provide any additional input, who explained that he had failed to allocate enough time to give the statements more than a brief glance due to other project involvements. Fortunately, one of the two uncompensated participants was able to take his place, as he had a very similar KRNW entry.

I retrieved anything noteworthy from the replies and inserted responses at appropriate places into the text sent out at the beginning of this round. As before, I kept names with statements,

and I also identified replies through indentation. The result were 116 pages, or about 6,500 lines of text. Again, I was still working exclusively with `vim`.

In addition, following-up to their replies, I exchanged just under 400 e-mails with the panellists, who readily responded. Each e-mail focused on one specific point in their reply, and sought clarification, or presented an opposing or related statement by another panellist, asking for further qualification, agreement, or refutation. I inserted the relevant bits of these follow-up statements into the collection and finished the second round with 156 pages or about 8,700 lines of text (*ca.* 270 kilobytes).

At this stage, it became increasingly more difficult to keep the two non-compensated panellists separate. However, both were enthusiastically participating, which made it basically impossible to continue without them. I continued to monitor their statements very closely and could not identify anything in their points of view that was in contrast to what other people said, so I moved on. I return to this issue in section 8.3.1.1.

6.2.6. Preparing the third and final round

Contrary to my initial estimate of about an hour time involvement for each of the 3–4 rounds, some participants reported spending up to 10 hours to complete the initial reply in the second round. Yet, they all readily responded to follow-up emails with further input. I considered it of utmost importance to ensure not exceeding the time estimate for the third round to avoid driving away participants.

This requirement, in combination with difficulties processing the large volume of data, were the main reasons for the long delay of the third round. Given the Christmas/New Year holiday 2008/09, as well as the release of Debian 5.0 "lenny" on 14 February, the delay suited most participants, who were busy with their projects. I sent regular status updates to keep the Delphi study on their minds.

For the third round, I analysed the results from round 2 (which included the results from round 1) and identified non-obvious and insightful statements by the panellists. Long statements, I shortened while paying special attention to capture context and essence without losing any of the detail. The result was a list of 281 such statements.

It should be obvious that reducing 8,700 lines of text to 281 is potentially a giant source of bias. I paid very close attention to this danger throughout the process, but probably failed to erradicate all bias. Thus, I considered it of utmost importance to keep meticulous records of the

process and to publish all data for future researchers. I will return to the concern of bias in section 8.3.1.2.

Even spanning 281 statements, the data set was still too big; given one hour, this large number would have meant that a participant could spend no more than 12 seconds on each statement, which would have made any higher-level analysis impossible. I thus investigated numerous strategies to deal with those data, and especially to reduce them further. Those that were unsuitable are briefly described in appendix C.1. I shall describe the process I used to reduce the data in the following:

Reducing the data set I started by picking a long-list of no more than half the statements. During this process, I gained a good idea of the amount of redundancy in the statements and had little trouble selecting 152 statements — more than planned, but not significantly less expressive than the full set of statements thanks to the reduction in redundancy.

My next goal was to create a short-list of 15–30 statements or statement groups, a process that suddenly appeared surprisingly simple. I found 33 such groups and went on to further reduce that number through merging and reorganisation.

At this point, I had to concede that the data were still too complex and inaccessible: when I returned to the groups after some time away, it would take me a few moments of scanning the statements to gather the commonalities among them and recreate the group in my head. I reasoned that if it took me considerable time to get back into the dataset with which I was already familiar, the participants would probably have even larger difficulties, and I had to keep the time restrictions in mind.

I attached keywords to each group but found that those were either ambiguous or not expressive enough. I proceeded to write concise, full-sentence descriptions of the aspects that caused the statements to reside alongside each other in any given group, taking special care not to leave out facets of statements that could be relevant. Instead, I concentrated on capturing the context of each group, which kept the individual statements together.

This approach made the individual statements secondary. I thus chose to present the participants primarily with those paragraphs, but also include the supporting statements in case anyone wanted more information on the constituents of each group. Writing those descriptive paragraphs helped improve the groupings in much the same way as putting thoughts to paper can give structure to ideas.

#	Title	Short description
1	Return-on-investment	value of future benefits
2	"Peercolation"	people networks and flow of information
3	Maturity	usability and stability without tracking development
4	Trialability	ease of trying out a tool/technique
5	Genericity	re-use of a tool/technique for other tasks
6	Quality assurance	automated tests
7	Transparency	automation and associated loss of control
8	Uniformity	benefits of and needs for uniformity
9	Quality documentation	maintenance, target audience, and verbosity
10	Scaled use	various degrees of use and gradual migrations
11	Consensus	just the right amount of discussion and consensus
12	Cost-benefit	investing time vs. getting things done
13	Marketing	increase awareness of a tool/technique
14	Compatibility	minimal effort for change
15	Elegance	personal preferences and irrationality
16	Examples vs. documentation	role of examples and templates
17	Sedimentation	the nature of spread of revolutionary ideas
18	Resistance	benefits of, and dealing with resistance
19	Modularity	granularity vs. monolithic solutions
20	Sustainability	confidence in the decision for a tool/technique
21	Network effects	dampening and enabling effects of teams
22	Chunking	revolutions and evolutions, and peace-wise adoption
23	Standards	creation and effects of standards
24	First impression	make sure the first impression is good

Table 6.7.: *The 24 categories distilled from the second-round responses.*

At this point I took the 129 statements that did not make the long-list and tried to re-integrate them. My theory was that redundant statements would fit trivially into the categories, while other statements would challenge the grouping. Reintegrating the previously discarded items seemed to reduce the bias introduced during the selection process. This approach allowed me to include even statements that were very specific to the Debian Project, but I did not refer to those in the descriptive paragraphs, keeping them generic.

By this process, I was able to reduce the 33 categories from the short-list to 24, which were supported by 204 of the 281 statements. Those 24 groups are shown together with one-line summaries in table 6.7 on this page. I shall explain the use of those short descriptions when I discuss the design of the next e-mail sent to the panellists in the next section. The descriptive paragraphs to each category, along with the supporting statements can be found in listing C.7 on page 370.

The descriptive paragraphs considerably improved the accessibility of the data, which three colleagues confirmed after proof-reading them. However, they constitute an unquantifiable source

of bias because they introduce additional levels of interpretation. On the other hand, presenting a set of statements does not come without similar problems: a reader who has to process a large dataset with limited time will not have the ability to view each statement independently of the group. Thus, the context in which a statement is presented has an unquantifiable influence on its interpretation, as well. I deemed it preferable to make the context explicit by providing the paragraphs, thereby reducing the focus on the individual statements.

6.2.7. Round three: identifying salient influences

The goal of the third round was to identify the salient influences to adoption or reject decisions among DCs. Initially, I had envisioned a ranking exercise for this round, but given the richness of the data, I didn't see a strategy for that. Also, I had begun asking the question over the worth of agreements between panellists on the relevance of individual influences, assuming that such an agreement can be found in the first place. With a diverse panel (see section 6.1.5), any ranking and thus selection based on relevance would be very weak, meaning that many categories would rank close to each other. If, on the other hand, the panel had not been diverse enough, then the ranking would not have been representative of the project. Furthermore, a meaningful, strong ranking would require additional rounds, which were outside of the capacity of the assembled panel. Instead, I sought to augment the existing influences with real-world instances of the underlying influences.

Nonetheless, I did not have to completely forego the identification of salient influences, despite the choice against a ranking exercise. I sought the proverbial stories from the trenches, and it seemed logical that every one of the participants would have more to say on influences that s/he had experienced before and was expecting to experience again. By asking each participant to pick a small number of influences and share details about instances in which those influences manifested themselves, I was able to gather more in-depth data, while also obtaining rough numbers on which influences were generally perceived more salient and thus nominated more frequently than others.

During participant selection (see section 6.1.5), each panellist was selected onto the panel according to his/her position on four strata; this selection process laid the basis for the diversity among the participants. Yet, my goal was always the identification of project-wide influences. There existed a chasm between each participant's immediate experience in the type of environment which s/he represents on the panel, and the project as a whole. I chose to approach this by asking the participants to identify two separate sets of influences: first, those

Listing 6.1: *Presentation of example influences in the email as displayed by the vim editor. The statements supporting the first point are folded, while the second set of statements have been expanded.*

```
 1. Some descriptive title

    The paragraphs on which you will concentrate are just like this
    one. It can consist of multiple sentences, but there is a common
    theme in those: the paragraph will represent a single influence
    you and your fellow panellists have identified, and your task is
    to determine how important this influence is.

+-- 11 lines: supporting statements--------------------------------

 2. Another title

    This is another paragraph...

    {{{ supporting statements
      - ... with supporting statements. Just one, for brevity of
        this example.
    }}}
```

that had immediate, personal relevance, and those that were speculated to be relevant on a project level.

6.2.8. Design of the third-round email

The e-mail I sent to kick off the third round (see listing C.7 on page 370) started with a recount of the study so far, which was motivated by the long time that elapsed between the second and third rounds.

The e-mail went on to introduce the format, in which the influences and their supporting statements would be presented. Again, I chose fold markers (*cf.* section 6.2.4) to make it easier to focus on the primary content. The result looked similar to listing 6.1 on the current page, depending on the editor used by the recipient.

Next, I presented the task, along with estimated time requirements, and noted that the participants would probably take slightly longer, but that I was unable to further reduce the data set without negatively impacting the study.

The task description encouraged participants to read the paragraphs, take notes, and let them settle. Then, I instructed the participants to

1. select the three paragraphs which reflect the three strongest influences (positive or negative) they have experienced in the context of their packaging work in the Debian Project, and share details about how these influences have previously manifested themselves and are expected to do so again in their immediate environments.
2. shift their perspectives away from their own work, consider the Debian Project as a whole, and repeat the exercise.

I explained that I needed the participants to be conscious about the difference between these two perspectives, but also noted that two sets could well overlap.

Finally, I allowed participants to address multiple influences as one, in case they experienced difficulty choosing between them. This decision was motivated by considerable overlaps between influences, and the difficulty experienced by my two colleagues, who inspected the instructions and influence set before they were distributed. Given that the desired results were in-the-wild experiences of salient influences, rather than a ranking, I deemed the impact on the usefulness of the results as negligible, while facilitating the job for the participants. Furthermore, I considered the possibility that this approach could expose clusters of influences I had not previously seen.

6.2.9. Wrapping up round three

I had initially asked the participants to respond within 12 days, but it quickly became apparent that this was not doable for some. In the end, the third round took 6 weeks, but every participant responded:

- Nineteen of the 21 respondents followed the instructions.
- one non-compensated respondent chose to assign between 0 and 5 points for each of the two perspectives to every influence, an alternative approach that would later be suggested during the feedback round (see section 8.3.2.4).
- another respondent described an alternative approach using a web-based surveying tool that implemented a similar scoring method to identify the most salient influences by concentrating on each influence in isolation (see section 8.3.2.4). He also assumed a total of five perspectives, rather than just two, to factor out those scenarios that he deemed too dissimilar from other tool adoptions.

Upon receipt of each response, I thanked the author for his/her participation and concluded the study, but not without keeping the option open to get back in touch with followup questions. In the end, 40 further e-mails were exchanged, seeking clarification, or asking for further specification of one's comment in the context of another participant's statement, or circumstance in the Debian Project.

Part III.

Analysis

Chapter 7

Results

Chapter 6 detailed the research approach I used to obtain the data to answer my research objectives:

1. to determine the salient influences to Debian Contributors (DCs)' tool/technique adoption or rejection decisions;
2. to propose a terminology of labels for these influences, and present them in a meaningful, accessible way;
3. to crystallise a number of implications for practice from these influences, to help increase the rate of diffusion of improved tools and techniques in the Debian Project, to foster competition and progress, and to enable the project to scale better with its growth.

The underlying research question was: What are the influences that shape DCs' adoptions of innovations?

Initially, my plan had been to design a framework of influences that were orthogonal to each other (*cf.* section 3.2). In section 3.2.2.1, I have claimed that the lack of orthogonality in the "elements of diffusion" [Rogers, 2003] may be a result of the generality of his work (see section 3.2.2.1). Since I approached my research question in a very general manner,[1] it quickly became apparent that the input received from the study participants was too multi-faceted to allow for the identification of principal components and thus proper orthogonality.

[1] Due to the lack of previous work on the issue, I took an exploratory approach, similar (but more broader in scope) to the first set of Delphi studies (see chapter 5).

When I did not see a framework emerge, I considered several approaches to structure the data, reduce it, and involve the study participants in the design of a categorisation or framework (see appendix C.1). None of these approaches proved viable.

I returned to the diffusions frameworks I researched prior to the Delphi study (see section 3.2) to determine if it would be possible to retrofit one of them to the data received. All attempts fell short, either because a category of influences would not fit well, or only a subset of each framework could be populated with influences. This seems to be due to their focus on individual decisions with little degree of interdependency on the decisions of others, the static definition of attributes (see section 3.2.2.1), and peculiarities of a FLOSS social system, and specifically the Debian Project.

For instance, while Rogers [2003, p. 183] notes that "a higher degree of re-invention leads to a faster rate of adoption [and] a higher degree of sustainability of an innovation," this view does not take into account the desire to minimise divergence from an innovation (see section 2.2.4.5), or allow treating "re-invent-ability" (or flexibility) as an attribute of an innovation, rather than a possible outcome of the innovation-decision process (see section 3.1.3).

Rogers claims that "the five attributes of innovations [...] may not always be the five most important perceived characteristics for a particular set of respondents." He refers to Kearns [1992] who found that most variance in the rate of adoption in this study could be explained by Rogers' five "elements of diffusion" (see appendices B.1 and B.1.1), but also established 20 other attributes that were guaranteed to be grounded in the respondents' own frames of reference [Rogers, 2003, p. 226]. Kearns [1992] summarises several empirical studies carried out before 1992 and noted that "the evidence suggests that a single set of innovation attributes (at least those studied thus far) are not very reliable as predictors of adoption, particularly when applied across multiple institutional contexts."

Studies in which the researcher decides on a set of criteria, or a framework before conducting the study, with the intention to fit the data into this framework, are flawed because they are "based on the *a priori* assumption that these are criteria in fact subjectively meaningful and relevant to potential adopters" [*ibid.*, his emphasis]:

> Attempts to correlate these "imposed" criteria with observed patterns of innovative behavior are, in effect, an effort to "force fit" observed behavior within the confines of preconceived theoretical frameworks. Naturally, all empirical research is theory driven, but in this case the claim can be made that researchers have apparently fallen short in the theory development phase of their methodology by failing

> to "ground" [Glaser and Strauss, 1967] the evaluative criteria that are used as independent variables; that is, there is no evidence in this stream of literature that the evaluative criteria have been derived from the underlying assumptions, values, and belief systems of decision makers.

In line with Rogers' encouragement to "[create] new scale items for each set of innovations to be adopted by a particular set of individuals, [rather than to] utilize existing scale items already developed by other investigators" [*ibid.*, p. 225], I chose to move on without a specific framework, but rather to find a way in which the results could be coherently presented. I knowingly opted against an existing, validated framework, because I could not find one that suited; I address validity through iterative development, and communicative validation with the participants of my study.

With the framework to be discussed in the following, I sought to build a categorisation of influences applicable to the task at hand, and to develop a definitive terminology for them, allowing future work to reference the influences in a way that their meaning is clear without further explanation.

7.1. Framing the results

A major challenge of this research has been the volume of data that was generated and had to be processed (see section 8.3). The resulting data are similarly copious, and in the following, I develop a structure to allow me to present them in a meaningful way.

7.1.1. Individual and organisational adoption timelines

The data from the Delphi study (see section 6.2) are the result of the inquiry for influences that shape Debian package maintainers' decisions to adopt or reject tools or techniques (see section 4.2).

If one regards adoption as a process (*cf.* section 3.1.3), then one possible structuring is to align the influences along a timeline, or a time-based stage model. Two classes of such models can be found in the literature on the diffusion of innovations: one in which the adopter is an individual, and another where the focus is on the organisation as a whole (see section 3.1.3). Rogers [2003] provides two five-stage models for the processes of individual and

organisational innovation, and Kwon and Zmud [1987] break the organisational process into six stages.

The existence of stages in the innovation process has not been empirically proven, but enough evidence exists to allow Rogers [2003, p. 195] to generalise that "stages exist in the innovation decision process" [Generalisation 5-12[2]]. Nevertheless, he purports that "individuals passing through the stages may or may not recognize when one stage ends and another stage begins," and that "we should not expect a sharp distinction between each stage".

Regardless of whether adopters go through distinct stages, stage-based models are "a means of simplifying a complex reality [...]. Perhaps we should think of stages as a social construction, a mental framework that we have created and generally agreed to" [*ibid.*]. I described my research as constructivist in section 4.1, and I am aware of the inherent limitation and danger of using a simplifying structure (see the discussion of the "frame problem" in section 3.2.1). However, the structure I am seeking is for presentation, not data gathering, and while any structuring necessarily affects (because it orders) the discussion of results, it does not preclude or obscure facets during the data collection.

My research focused on the Debian Project and its set of contributors as a whole, rather than looking at individuals. As a result, the individual innovation-decision process model by Rogers is not a reasonable basis for laying out the results of the study, and I ought to consider a model of organisational innovation instead. On the other hand, in a voluntary context, each individual plays an important role in the organisational adoption process, and the variables relevant at the individual level also gain significance in the organisational context. This suggests that a combination of stage models will be needed.

Between the two organisational adoption models presented in section 3.1.3, the model by Kwon and Zmud [1987] is a little bit more granular. In addition, the names of the stages in the model put forth by Rogers [2003] are derived from verbs, which have a more active connotation and thus lean more towards authoritarian organisations. Arguably, this is a minor, possibly subjective reason, but the former model nevertheless seems more appropriate for my study.

[2] Rogers [2003] includes numerous generalisations throughout the text, which are numbered c—n, where c is the chapter number, and n the index within the chapter.

Organisational stages	Individual stages	Description
Initiation	Knowledge	perceived need or pressure to change evolves
Adoption	Persuasion	interest or disinterest is formed
	Decision	the innovation is investigated and adopted or rejected
Adaptation	(Re-invention)	the innovation is adapted to local needs
Acceptance	*Implementation*	the innovation is used
Use-performance-satisfaction	*Confirmation*	a state of comfort is maintained
Incorporation	n/a	the innovation is standardised

Table 7.1.: *Comparison of the organisational innovation stage-model by Kwon and Zmud [1987] and the individual innovation-decision process by Rogers [2003]. The individual stages tend towards the earlier stages in the organisational model, and the mapping is less definitive further down the table (where the individual stages are emphasised). Re-invention is not a stage in the individual model, but occurs at the decision and implementation stages, or in-between, according to Rogers [p. 180ff].*

7.1.2. A stage model for adoptions in the Debian Project

In this section, I would like to develop a combination of two stage models to create a structure that is suitable as presentation framework of adoption decision influences in the Debian Project.

Table 7.1 on this page is a crude juxtaposition of the organisational innovation stage-model by Kwon and Zmud [1987] and the individual innovation-decision process by Rogers [2003], in an attempt to highlight the parallels. The first three stages of the individual stage model cluster alongside the first two stages of the organisational stage model. Then, the individual stages seem to fade, and the mapping between the two models becomes less clear. For instance, individual implementation could well be seen to precede organisational adaptation, but there are also reasons why implementation is more of a symptom of acceptance.

A naïve linear combination of two stage models might start with the individual model and end with the organisational model. Confirmation, which is an optional stage in Rogers' model, seems to fit nicely with "use-performance-satisfaction" and one might be inclined to fuse the two, to obtain a ten-stage model.

At first glance, this model fits well: in an organisation like the Debian Project, innovations are introduced by individuals in local contexts, rather than through authoritarian "agenda-setting" or "matching". Therefore, an innovation needs to have been adopted at the individual level before organisational adoption can begin.

However, even if a considerable number of individual adoptions precede an organisational adoption, the model would be inadequate to illustrate any influences on later adopters that are a result of the previous organisational adoption. For instance, following the majority adoption of a technique to the point where it has become incorporated into daily practice (and thus the organisational adoption process has come to an end), the ten-stage model could not explain the behaviour of later adopters striving to conform with the rest of the project.

FLOSS adoption is cyclical One logical step to deal with this shortcoming would be to tie the ends together and think of the process as a cycle, instead of a line. Firstly, this addresses the aforementioned shortcoming of a linear model to cater for different adopter classes (early adopters *vs.* late adopters); secondly, a cycle seems to fit in nicely with the open nature of innovations in FLOSS, where innovations are often the result of previous adoptions.

However, simply tying ends together fails to take into account that individual adoptions are essentially *part of* the organisational adoption process. Looking for ways to link the two processes more tightly, I noticed two additional parallels:

First, I took note that the first stages of both models, the initiation stage, as well as the individual knowledge stage are concerned with the exchange of ideas, information and needs, and specifically the question whether a need follows a technology, or a technology is developed to satisfy a need. This suggests the fusion or juxtaposition of the two, to which I shall return at the very end of this section.

The second parallel between the two different models shown in table 7.1 on the previous page is between organisational adaptation and individual re-invention. The latter is not really a stage in Rogers' innovation-decision process, but he considers it part of the decision and implementation stages [p. 180ff]. According to Kwon and Zmud [1987], adaptation is nothing more than organisational implementation, which requires individual implementations as part of the process of convergence between organisation and innovation.

Given these two corresponding points in the two stage models, I let the individual stages between knowledge and implementation coincide with (or replace) the organisational adoption

stage. According to Kwon and Zmud [1987], at the adoption stage, the organisation decides to allocate resources towards the implementation of an innovation. Such a step is unheard of in the Debian Project, where no entity can allocate the resources necessary to carry through an adoption process, namely the individual adopters themselves.

In the Debian Project, no distinction exists between "technicians" designing tools and techniques, and "users" employing them, as one might find in traditional organisational settings; instead, the users are the ones developing the techniques, by way of implementing and re-inventing them themselves. Thus, sticking with the general idea that individual adoption precedes organisational adoption, it seems sensible to place individual implementation separate from but feeding into organisational adaptation.

With the individual adoption stages in place of the organisational adoption phase, the model is still mostly sequential in that the four main individual adoption stages precede the organisational adaptation stage without any back reference. The missing link is a feedback loop from adopters back to the knowledge stage, which represents the so-called "peercolation".[3]

The adoption cycle just developed is depicted in figure 7.1 on the facing page. With the feedback loop in place, it is now possible to trace any number of individual adoptions required to enter into the organisational adoption track.

Critical mass and consensus Adopters of an innovation may get involved in adapting the innovation to the organisation, or even prepare the organisation (*e.g.* its infrastructure) for an innovation. As acceptance grows among peers, the point of critical mass may be reached, after which the diffusion of the innovation is said to be irreversible in this social system (see section 3.1.2).

With critical mass, an innovation will most likely become a *de facto* standard. Alternatively, at this point it is conceivable even in the Debian Project that a decision is made and an innovation chosen, as long as it happens on the basis of discussion, or is backed up with arguments so that the decision appears considered.

The adopters may experience confirmation of their choice as use of the innovation continues to grow, and benefits originate in improved collaboration, re-inventions, additional documentation, bug fixes, and advanced features. The exact point at which this confirmation takes place

[3] The term "peercolation" was coined by Christian Perrier at the start of the Delphi study to describe the percolation of information (and particularly knowledge of innovations) through networks of trusting peers [<20081104174209.GU4145@mykerinos.kheops.frmug.org>].

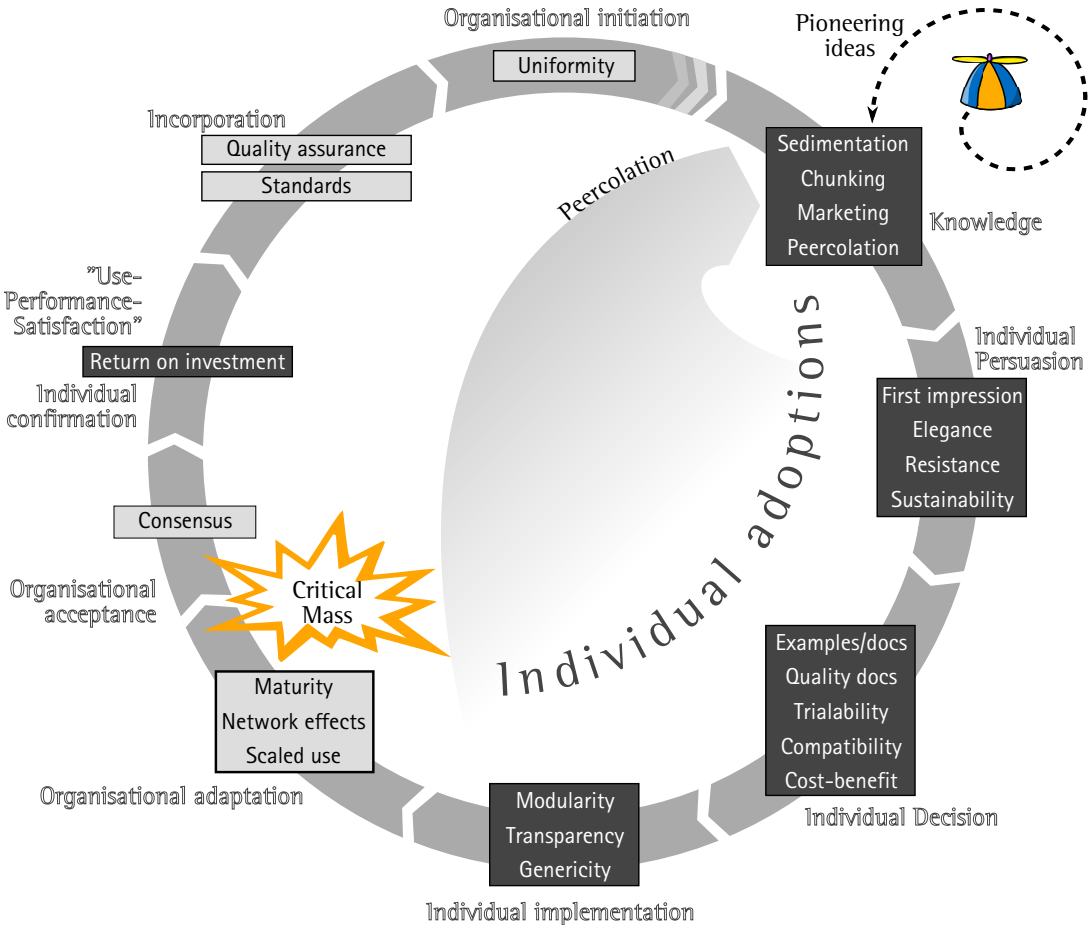

Figure 7.1.: *The combination of individual and organisational adoption stage models arranged in a cycle, which will be used to present the results from the Delphi study. Around the edge with an outlined font are the names of the stages. Words inside rectangles on the cycle refer to the influences. The dark rectangles represent individual adoption stages and may occur many times in parallel. The light, curved arrow through the centre of the circle visualises the influence of peers talking about their own adoption experience onto peers at the start of the cycle. Light rectangles on the cycle hold influences relevant at organisational stages.*

— if it takes place at all — remains undetermined. Therefore, the use-performance-satisfaction-confirmation stage assumes a somewhat special place in the model. I considered displaying it alongside the cycle, starting at the individual implementation stage and extending all the way through organisational initiation, but that would have cluttered the figure more, without a real benefit. I therefore chose to keep the stage at the position where confirmation is logical: after the point where an adoption becomes explicitly or implicitly standard across the organisation.

The final stage of organisational adoption is the incorporation of the innovation. This may involve the definition of standards, the codification of best practices, or an innovation may simply become widely advocated.

Organisational initiation comes last In figure 7.1 on the previous page, the three shaded arrows at the top right of the cycle, as well as the arrow from pioneers indicate the starting point for the discussion of influences that is to follow. This may seem strange as it puts organisational initiation last, which appears at least a contradiction in terms. However, in the context of the Debian Project, there is no authoritarian body that can dictate practices or mandate adoptions. It may be that changes are proposed through executive decisions, but those will only be effective if they build on previous adoptions, and consensus or competency among the decision makers.

Organisational initiation comes last, but stays at the top of the cycle, which is to suggest that it is an important stage in the organisational context, but to reach it, at least one organisational adoption cycle has to have been completed. Organisational initiation thus does not begin the initial iteration of stages, but precedes subsequent revolutions of the cycle.

7.2. Influences to Debian Contributors' adoption decisions

Out of the second Delphi round (see section 6.2.3) emerged a set of 24 categories of influences which explain adoption decisions amongst DCs *in theory*. On the following pages, I present these categories (henceforth just referred to as "influences") along the nine stages of the adoption timeline as described in section 7.1.2, which can be briefly summarised as follows.

A discussion on the role of pioneers will precede the other stages in section 7.2.0,[4] because pioneers are necessary to bootstrap the cycle. At this point it suffices to say that there are pioneers who think in organisational terms and investigate project-level improvements and organisational streamlining. However, these remain individual efforts coming out of a wide range of motivations, they are not initiated at the organisational level.

0. Pioneers — Not a stage in the model, but since innovations originate among pioneers, pioneers are a prerequisite for the adoption cycle. See footnote 4 on this page for the choice of '0' in this list and the number of the section on pioneers.

1. Knowledge — Ideas need to spread before they can be adopted. Diffusion of knowledge happens through a number of channels, and several factors affect the spread.

2. Individual persuasion — Individual adopters form an opinion about an innovation before they are ready to decide for or against it.

3. Individual decision — A positive attitude towards an innovation can lead to practical investigation of the innovation, with the intention to make an adoption decision.

4. Individual implementation — Individuals who decided for an innovation now put the innovation to use, possibly re-inventing it to better suit their use cases.

5. Organisational adaptation — Through several individual adoptions, the innovation and the organisation converge, and both might change to better accomodate each other.

6. Organisational acceptance — After a while, an innovation might become widely accepted, loses its novelty, and consensus and/or critical mass is formed.

7. Use-performance-satisfaction, individual confirmation — Somewhat of a special stage in that it could well start earlier in the cycle, the adopters start to reap benefits herein.

8. Incorporation — The use of an innovation becomes routine, and the use is formalised.

9. Organisational initiation — An innovation that has been incorporated provides an incentive to adopt for those who have not yet done so.

The lead paragraphs in each influence section

- summarise the gist of the influence;

[4] This unusual section number was chosen to allow stage one to be section 7.2.1, and because pioneering activity precedes the stage model.

- show how I summarised the influence for the panellists between the second and third rounds.

The discussion will be interlaced with quotes gleaned from the Delphi study, presented in indented fashion like the following paragraph. These quotes are by panellists of the study (and reproduced with permission) and will not be introduced as such each time, but attributed as shown:

> *This is an example quote.*
>
> — *Panelist name*, *context* <Example-Message-ID>

Occasionally, I may ascribe arguments or claims to participants' statements without an explicit quotation stanza; such an attribution appears like this: [*Panelist name*, *context*[5]].

I have done my best to repeat the participants' opinions in a coherent way. However, where no coherence could be found between claims categorised under the same influence, I refrained from creating segues. Therefore, not all influences are presented as contiguous stories, to minimise bias through artificial contexts, and last but not least, in the interest of brevity.

Anonymity As I discussed in section 5.4.5, I chose to maintain panellist anonymity throughout the Delphi study, because I wanted to give the participants the opportunity to focus on content and merit of statements, without being subjected to possible preconceptions about a statement's author. In section 8.4.3.3, I reflect upon this choice, which proved to be a good decision.

Nevertheless, in the following presentation of results, I include the full author attributions with statements, even though this may affect the ability of the Debian-immersed reader to judge statements for their worth. My decision was helped by feedback from participants, and in particular the following statement:

> *If I were you, I would present the data to the Debian audience in the way they are familiar with, namely "assertions" stated in mails which prominently show identities.*
>
> — Stefano Zacchiroli, post-study feedback
> <20090920134246.GA17553@usha.takhisis.invalid>

To ensure the validity and reliability of the study, I make all discourse from the study available (see section 4.3.1), and I provide direct references to the e-mail messages from which I sourced the statements, as shown above. Therefore, anonymising statements would only constitute an

[5] <Example-Message-ID>

inconvenient obstacle for those seeking to verify the origin of claims, but not prevent anyone from discovering the authors' identities. Therefore, attributing statements increases the worth of the data.

Rephrasing Statements may not appear verbatim. I encouraged the panellists not to worry about language and writing style throughout the entire process, but would like to present their views in a way that does not to suggest carelessness on their part. By extension, grammatical and orthographical errors are mine by default. Thanks to the availability of the data (see section 4.3.1) and the message-IDs, it is possible to locate the source of each statement for verification of my rephrasing.

To ensure that I correctly represented the panellists' views, I made this chapter available to them two weeks prior to finalisation, explicitly asking them to check for misrepresentation. I received no change requests.

Delphi round numbers I include the Delphi round number in the context of which any statement was given. This serves two purposes: first, it puts the statement into temporal context of the whole Delphi process; second, it gives an indication about the level of consensus on which the statement is based:

1. Statements made in the first round are purely subjective and are not in any way based on the input or opinions of other participants. They may have been contested in the second round discussion, and I took every effort to make sure that contested statements are followed shortly by such counter-arguments.
2. Statements from the second round are based on the collective input gathered in the first round. They were not followed by a discussion round like the first round statements, but I did exchange almost 400 e-mails with participants and put statements into context with each other, or asked participants to comment on refutations or claims made by others, where I saw a link.

   There is, undoubtedly, a level of bias in this, but given the amount of data collected by the end of the second round (see section 6.2.3), it was not possible to engage in further group discussion (see section 8.3.1).
3. Those statements made during the third round are based on the experiences and insights gained throughout the study from the participants, as well as the condensed 24 influences I presented at the start of the round. Around 40 e-mails were exchanged *post-hoc*, but

I fed neither those statements nor the rankings back to the participants. Therefore, they are mostly unchallenged, but the third round brought few arguments to the discussion, which alleviates this concern a bit.

7.2.0. On the role of pioneers

Before presenting influences, the cycle depicted in figure 7.1 on page 141 has to be bootstrapped. Due in large part to the open nature of innovations, development in FLOSS environments is inherently cyclical. In particular, many ideas and solutions build upon previous ideas and tools. A spiral might be an even better metaphor, but the cycle is more convenient to illustrate the process.

Nevertheless, not all ideas build upon existing ones. In fact, if we go back to the beginnings of the Debian Project, a sizeable amount of pioneering work laid the foundations on which the project was built. These foundations include *e.g.* the package management system and the source package format, the Bug Tracking System (BTS), and the Debian archive. Even nowadays, new ideas enter the project with the potential to revolutionise the processes therein. Recent examples might include packaging with distributed version control system (DVCS), but also the isolated build environments, to which I shall return shortly.

Pioneering ideas are usually born inside the project, or carried into the project by a cosmopolite member,[6] who picked up a concept externally in order to apply it within the Debian Project. Furthermore, ideas might also be transferred within the project, picked up in one area and henceforth adapted and applied in other contexts.

For instance, isolated build environments[7] have their roots in the Debian archive "autobuilders" — the scripts that rebuild packages on the various supported architectures. At some point, members of the project deemed it inconsistent that the project infrastructure rebuilt packages in clean, isolated environments, while the developers used their "unclean" workstations[8] to create them. This gave rise to a number of tools that facilitated the maintenance and use of such clean environments, and integrated them into the regular build process. In this case, both the idea to use isolated build environments for package development, as well as the tools that implemented this technique were innovations.

[6] Cosmopoliteness is the degree to which an individual is oriented outside a social system [Rogers, 2003, p. 290].

[7] These are also called "chroots", which derives from the name of the command `chroot` used to "change" the system "root" for a process to a clean environment.

[8] Workstations are considered unclean in this context, because they have packages installed that may not be installed elsewhere. The packager may hence easily forget to declare a build dependency, causing remote builds to fail.

Over time, the technique has been widely adopted and become highly recommended, if not mandated as much as is possible without being specified in the definitive standards documents (see section 7.2.8.1):

> *A pbuilder-like test environment*[9] *prior to upload is now so strongly suggested that someone could say "imposed". If someone nowadays is not in a position to do a pbuilder test of an upload and hits a missing build dependency bug, they'll be very likely to have some really nasty remarks in the bug reports.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

Initially, the innovation started out with a need on the side of the developers who implemented the tools to enable clean building. At the time, other developers may have also perceived a need for the technique but saw no way to satisfy it, and did not want to implement the tools themselves. The majority, however, probably did not know about such a need at the time, and did not fathom use of the technique in this way.

At some point, a DC decided to cross the chasm and pioneered the idea of using isolated build environments for local package builds. The outcome was an invention, a tool that facilitated the task, and its adoption cycle could start with people discovering the tool.

7.2.1. Stage one: knowledge

The first stage of the Debian adoption process is a fusion of the first stages of the individual and organisational models combined in section 7.1.2. It is the beginning of an individual adoption process, and also the beginning of an organisational adoption process in the absence of authoritarian agenda-setting.

The knowledge stage starts with the first exposure of an individual to an innovation, and usually involves a need. Zmud [1984] differentiates between "need-pull" and "technology-push" knowledge: does a need precede an innovation (*i.e.* an individual seeks a solution to an existing problem), or does a technology create or instill a need? This question has not been adequately resolved in the literature. Zmud [1984] hypothesises that an innovation will be more likely to diffuse successfully if need and availability coincide, suggesting that mere availability does not necessarily drive adoption (or create needs). In the case of the isolated build environments mentioned in section 7.2.0, need and availability were brought together by a pioneer.

[9] pbuilder is one of a family of tools that facilitate package building in isolated environments.

Central to the knowledge stage is thus a perceived need on the side of the potential adopter, and this need has its origins in any of a number of sources. These influences are the subject of this section, and the common theme is the flow of information, and how knowledge about an innovation reaches an individual.

At this early stage, knowledge about the innovation is purely cognitive: the individual becomes aware of the innovation (awareness-knowledge, see section 3.1.2), might perceive how an innovation can be used (how-to knowledge), and possibly even starts to understand some of the principles behind it (principles knowledge) [Rogers, 2003, p. 172ff]. The latter two types of knowledge, how-to and principles knowledge, gain importance as the adoption progresses, and I shall return to them in section 7.2.3.2.

The focus at this stage is on how information about an innovation spreads and reaches the individual. The first stage ends when the individual has enough knowledge to be able to form an opinion about an innovation, and moves forward to do so.

Alternatively, the individual might simply forget about the innovation, although this is unlikely given the tightly connected communcation network at the core of the Debian Project [Luca Capello, round two[10]]. It is more likely for DCs to postpone further evaluation of a tool, due to lack of time, or to avoid being too early an adopter [Joey Hess, round one[11]].

7.2.1.1. Influence 1: Sedimentation

Gist: Knowledge takes time to spread.

Summary used during study: Ground-breaking ideas need time to spread. People reject ideas until they understand the underlying problems, are able to formulate them in their head, and identify the benefits of a solution. In order for everyone to understand the principles, explanations must cover multiple perspectives.

Sedimentation covers the spread and settling of ideas and innovations among a group of people. The concept is related to gaps introduced into the social system as consequences of diffusions. Those gaps have been extensively studied in the context of classical diffusion [Rogers, 2003, p. 456ff], and are mostly of socioeconomic nature. Thus, they have little to no bearing in the context of the Debian Project. The underlying principles hold, however.

[10] <87k5aksbpl.fsf@gismo.pca.it>
[11] <20081106024658.GA10322@kodama.kitenet.net>

The diffusion of an innovation partitions the members of a social system into adopter innovativeness categories, from early adopters to the so-called "laggards" [Rogers, 2003, ch. 7]. A multitude of reasons for this partitioning exist, ranging from personality traits to issues of understanding. While there is little to be done about personality traits and the excitement of those who "cannot be bothered" will likely remain where it is, effective communication can help decrease the gap, at least in terms of understanding.

For awareness about an innovation to grow, the innovation must be understood. This process can take substantial amount of time.

> *DVCSs have been around for years, and it's only now (last 2 years) that we see a real growth in users.*
>
> — Pierre Habouzit, round two <20081122152202.GA30285@artemis.corp>

> *What I've seen is that usually something revolutionary appears, the group is not yet ready, so only few people get into it, but in general it is completely ignored. Then the idea starts pouring into the group slowly (it might even take years!) through the "sedimenting" process. At some point most of the people in the group have already heard of that new thing, which no longer seems like something completely alien. Then people might feel comfortable playing with it, and might start liking it, and even using it for real work.*
>
> — Guillem Jover, round one <20081112061945.GA26186@zulo.hadrons.org>

The inertia referenced in this statement is the result of several possible reasons.

First, Rogers [2003, p. 171] identifies "selective exposure" as the tendency "to attend [only] to communication messages that are consistent with the individual's existing attitudes and beliefs." This means that even though an individual has been exposed to an innovation, s/he may not consciously notice or downright neglect the innovation due to conflicts with existing predispositions.

A related tendency, "selective perception", is based on the claim that individuals mostly expose themselves to messages about an innovation for which they perceive a need [Hassinger, 1959, cited in [Rogers, 2003]], and then perceive and interpret them according to existing attitudes and believes.

> *Most of the time, new solutions try to solve problems people have not yet formulated in their heads, or which seem irrelevant compared with other currently more important problems, or the advantages do not outweigh the perceived disadvantages.*
>
> — Guillem Jover, round one <20081109225614.GA8728@pulsar.hadrons.org>

Potentially complex knowledge also needs time to spread. An innovation may be available, but the potential adopter might not understand it enough to the point to perceive a need, or identify how the innovation can be put to use to meet an existing need, for lack of understanding.

[Rogers, 2003, p. 464f] suggests to focus less on the innovators and early adopters, but make messages primarily accessible to less innovative people.[12] There is an inherent network effect in this: the more people who understand an innovation, the more people can talk about it and spread the information further. This creates redundancy and introduces diversity, which helps spread the knowledge even further.

> *Some techniques are just hard or a major mindset shift, and for those I think they're the most likely to be successful if they're explained by different people from different directions until there's an explanation that happens to match each different way of thinking about the problem space.*
>
> — Russ Allbery, round one <87skq26i7c.fsf@windlord.stanford.edu>

One participant suggested that this aspect is not limited to the diffusion of innovations, but could even be beneficial in the context of discussions:

> *The "sedimentation" idea made me think of mailing list discussions in a different way. I used to think that the usual endless circles there achieve nothing and mostly happen because people don't read what the others write. However, maybe there is more worth than I thought in rehashing the same idea from all sides, if only because it takes some time to collectively sink in.*
>
> — Niko Tyni, round three <20090420113239.GA25346@kuusama.it.helsinki.fi>

The bottom line of this influence is that information can spread slowly, especially when it is about complex concepts, or addresses needs that are not yet explicit.

[12] Rogers [2003, p. 460] suggests to target messages at the so-called "downs". He segments the "downs" from the "ups" and explicitly notes that the segmentation could happen at the level of adopter categories (which is how I used it in talking about "less innovative people"), level of information, or socio-economic status, but that certain regularities prevailed.

7.2.1.2. Influence 2: Chunking

Gist: Smaller tools and techniques are easier to fit into existing workflows.

Summary used during study: People have established workflows and adapt tools to fit those. They (might) want to improve/evolve those workflows, rather than replace/revolutionise them. Free time is fragmented and the adoption of a new tool/technique often requires committing oneself over larger chunks of time. Revolutionary ideas are hard to spot initially, but if they can be identified, their supporters should try to break them down into an evolution with individual components that can be adopted piecewise.

Chunking is about the difference between a revolution and a series of evolutions. Small chunks are easier to digest than large ones, and the same applies to ideas and innovations: tangible ideas are easier to grasp, and piecemeal adoption easier to achieve, not only because free time is often fragmented, allowing for small tasks to be done, but not leaving room for larger, coherent tasks [Charles Plessy, round two[13]]. This also relates to a later influence, trialability (see section 7.2.3.3).

One participant put it succinctly:

> *Large, monolithic changes to processes tend to take a lot of time, require a lot of debugging, and be disruptive to a general goal of getting things done, rather than working on tools. It's therefore much easier to adopt a tool or technique that can be applied in small chunks or in a self-contained area, or slowly over time.*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

What seems to be quite evident needs not be taken into account, for instance when "people unconsciously try to shove radical ideas [into the group] and expect everyone to appreciate them as obvious improvements" [Guillem Jover, round one[14]]. A better approach may be to chop an innovation into pieces [*ibid.*]:

1. Create something revolutionary, but do not push yet.
2. Split it in smaller unrelated parts.
3. Prepare the people with the simple concepts.
4. Deploy the simple tools/techniques implementing those simple concepts.
5. After some time, present the unified, no longer alien solution.

[13] <20081204143515.GA8588@kunpuu.plessy.org>
[14] <20081112061945.GA26186@zulo.hadrons.org>

If each chunk brings about only a minor change to existing practice, then there is little in the way of an innovation's diffusion. Even though subject of a later discussion on compatibility (see section 7.2.3.4), the following question by Rogers [2003, p. 245] is appropriate in this context: "The more compatible an innovation is, the less of a change of behavior it represents. How useful, then, is the introduction of a very highly compatible innovation? Quite useful, perhaps, if the compatible innovation is seen as the first step in a cluster of innovations that are to be introduced sequentially."

Chunking an innovation into individually "compatible" pieces and make the adoption process unconscious sounds convincing, but it might not be easy, because revolutions are not always easy to spot up-front:

> *Chunking is difficult because at times, [the effects of] new tools that introduce revolutions are initially not well understood by their authors, and details are worked out while one is chasing a vision that becomes more and more clear with time.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

Furthermore, the extent of a revolution cannot be judged beforehand, and it is thus difficult to decide whether the hard work of chunking will be worth it [Raphaël Hertzog, round two[15]].

Lastly, for certain classes of tools or techniques, the chunking is simply not possible, unless they are implemented in unconventional ways:

> *I dislike using a wiki because the editor environment is horrible without going to extra work to set up some sort of external editor linkage with a web browser. It's a good example of tools that force a monolithic environment rather than letting one easily swap in the tool of their choice for a common operation, like editing text.*
>
> *IkiWiki is the first wiki I've ever been able to use sufficiently comfortably to want to put a lot in it, precisely because it doesn't re-invent its own separate environment but instead integrates with standard editors, standard formatting languages, standard version control systems (VCSs) for history,* etc.
>
> — Russ Allbery, round two <87wsep9a04.fsf@windlord.stanford.edu>

Awareness of the necessity and possibility of chunking, as well as the effects fosters thinking in terms of manageable chunks, and progressive, evolutionary change.

7.2.1.3. Influence 3: Marketing

Gist: Active promotion helps overcome information overload if done right.

[15] <20081122155833.GY2212@ouaza.com>

> **Summary used during study:** Buzz and excitement increase the exposure of a tool or technique, and can drive evaluation, though not necessarily adoption. It is important to use the right channels, and to provide repeated exposure to the tool/technique one is promoting, not just a single glimpse. Success stories and a positive attitude are stimulating, especially when needs are met instead of created. A corporate link, premature promotion, and TV-style marketing, on the other hand, can have negative effects and ought to be avoided.

According to Wikipedia, "marketing is an integrated communications-based process through which individuals and communities discover that existing and newly-identified needs and wants may be satisfied by the products and services of others."[16].

It is my impression that the term has gained a negative connotation, mostly due to the flood of consumer innovations today, in which needs are generated for commercial purposes [*cf.* Rogers, 2003, p. 172]. For instance, the mobile phone market of today is, in many ways, defined by needs-creation: consumers are infiltrated to believe that the ability to transmit low-quality picture and video snapshots instantaneously is an essential luxury of life. A commercial background is undeniable, and this is perceived as a negative trait by some.

On the other hand, marketing is also about identifying and catering to *existing* needs [Enrico Zini, round one[17]], and the "integrated communications-based process" as Wikipedia has it. Knowledge of the workings of this process helps to support the diffusion of an innovation.

The importance of marketing

> *One challenge we have is how to get information about new tools and techniques to the people.*
>
> — Niko Tyni, round one <20081107183929.GA5087@rebekka>

The somewhat abundant and diverse collection of communication channels available in the context of Debian can facilitate the spread of information, and may speed up adoptions if used right (*cf.* section 3.2.2.1). The main problem — if there is one — in the Debian Project seems not to lie in the lack of communication media, but rather the number thereof, which forces people to select a subset of media and concentrate on those:

[16] http://en.wikipedia.org/wiki/Marketing [24 Jul 2009]
[17] <20081205140625.GA22710@enricozini.org>

> *Our communication media are numerous, but the available time is not extensible. We probably all tend to focus on* some *information media (which vary from one person to another). This potentially forms smaller, well-connected groups who might not interact with each other enough.*
>
> — Christian Perrier (his emphasis), round one
> <20081107193431.GQ4145@mykerinos.kheops.frmug.org>

Such cliques present a challenge when trying to get information about tools and techniques out to the contributor collective, although they also further the spread of information through network effects (see section 7.2.5.1). However, there seem to be sensible channels that currently appear underused:

> *Reaching people is not the problem, as we have the Developer News sent to* `debian-devel-announce`[18]*... we rather need to make tool developers aware that they can spread their ideas there. In my opinion, they should make a detailed announcement to the* `debian-devel` *mailing list with the goals of the tool, and then submit a short summary to the Developer News.*
>
> *It's just a pity that so many people are not aware that it's an important part of their job to reach out to other people. We were better at this when we were smaller.*
>
> — Raphaël Hertzog, round two <20081122155833.GY2212@ouaza.com>

On the topic of why more marketing initiative would be essential, this participant had the following to say:

> *Getting the word out makes other feel part of a wider community. By sharing knowledge, we can avoid duplication of efforts, and since it's (almost) the only (effective) way to recruit new people interested in what you do, it's essential for the long-term survival of the (sub-)project. It lets the rest of the free software world know that the project is alive and kicking.*
>
> — Raphaël Hertzog, round two <20081122155833.GY2212@ouaza.com>

I will return to long-term aspects at later stages, *e.g.* when discussing sustainability in section 7.2.2.4, and return-on-investment in section 7.2.7.1.

Communication media and their use The main communication media of the project are, in descending order of breadth of reach: mailing lists, web-log posts aggregrated on Planet Debian[19], Internet relay chat (IRC) channels, and occasional real-life interactions, which are useful for less focused discussion and controversial topics [Guillem Jover, round three[20]]. This would suggest that mailing lists are the first choice to spread information.

[18] This list is the only list that is considered to be mandatory for project members and contributors.
[19] `http://planet.debian.org`
[20] <20090422045458.GA14006@gluon.hadrons.org>

However, when compared to the main Debian development mailinglist, `debian-devel`, "the blogosphere[21] is more appropriate for 'advertising' personal technical choices" [Stefano Zacchiroli, round two[22]], because it does not render itself to tangential discussion or loss of focus[23].

Web-logs are part of the fragmented media space, and the majority of DCs do not publish their views on Planet Debian[24]; one participant remarked that advertising on blogs is like "hiding Camel advertising inside Camel cigarette packs" [Andreas Tille (a non-smoker), round two[25]]. Nevertheless, web-logs appear to have a huge impact on contributors of the project:

> *Buzz on Planet Debian seems to me to have a huge impact on what people pick up and start using. The project's huge adoption of Git, for example, seemed to me based heavily on that sort of buzz, particularly given that there was quite a bit of resistance and negative publicity over its ease of use to overcome. Despite those obstacles, the amount of enthusiasm and success stories have made it far more popular than either Mercurial or Bazaar, both of which have a reputation for being easier to use and the latter of which has more direct connections with Debian.*
>
> — Russ Allbery, round three <87d4b9o4m7.fsf@windlord.stanford.edu>

As more people start using web-logs to advertise and announce, the effectiveness of the medium will decline with the signal-noise-ratio, and information may drown in volume [Charles Plessy, round one[26]]. Other, more focused media may have to be sought:

[21] "Blogosphere" is the world of web-logs ("blogs") and their authors ("bloggers")

[22] <20081130181416.GA13001@usha.takhisis.invalid>

[23] While some web-log authors welcome comments to their posts, others do not. In any case, comments are not as visible and somewhat secondary. Therefore, web-logs are rather one-way communication media, and even though people have used their own web-logs to reply to the articles of others, this practice is frowned upon and not really effective.

[24] At time of writing (18 Sep 2009), 342 web-logs were aggregated on Planet Debian, while Steinlin [2009] identified 503 core developers. Given that some web-logs do not belong to core developers, it might be a reasonable assumption that one in two core developers publishes to Planet Debian, and that number decreases a lot when posting frequency is taken into account.

[25] <alpine.DEB.2.00.0812050821200.9613@wr-linux02>

[26] <20081112055226.GD27041@kunpuu.plessy.org>

> *Just like we have a "package of the day" concept, we could have a "packaging tip of the month" or wiki pages on case studies for this or that particular packaging situation or meta-packaging situation (e.g. moving to a different svn repository, packaging a lot of small packages, uploading a bunch of packages effectively* etc.*).*
>
> — Loïc Minier, round three <20090505203417.GA25853@bee.dooz.org>

Maybe a central advertising platform, like the aforementioned Developer News can bridge the gap and push a collection of relevant articles, rather than expecting the project contributors to poll an even larger number of sources for the input.

Inexperienced people can have a hard time diffusing information in the project due to not knowing how to achieve that [Stefano Zacchiroli, round two[27]], or because their priorities are different, and time is short:

> *I have more fun coding than announcing, and especially after a long coding burst I have little energy left to dedicate to promoting the code, or […] prefer to use it to fix the code even more.*
>
> *Today, I would spend less time writing code to cover corner cases and more time talking with friends about how to make useful stuff out of it* right now *rather than in the future.*
>
> — Enrico Zini (his emphasis), post-study follow-up of round two statement <20090817113214.GA32630@enricozini.org>

In ways, this leads back into sedimentation (section 7.2.1.1): getting the word out to multiple people, in combination with providing something that can be used early on, increases the number of people who would spread the word, and also increase the spectrum of perspectives on a new innovation, effectively increasing the speed of sedimentation.

Similarly, obtaining feedback, while beneficial *per se*, may also lead to testers talking about an innovation in public discussions, possibly even recommending it [Enrico Zini, round two[28]].

Bootstrapping adoptions by way of peers, in combination with the multi-tier nature of communication in a FLOSS project can establish a base-level of "buzz" about an innovation, which, logically, will only be sustained if the innovation has sufficient merit. As another participant suggested, the diversity and high level of interconnectedness of the Debian community ensures that even if someone is not interested in or against a particular tool or technique, s/he is still continuously exposed to it, kept aware of improvements, and able to change his/her mind [Luca Capello, round two[29]].

[27] <20081217102151.GB10133@usha.takhisis.invalid>
[28] <20081205140625.GA22710@enricozini.org>
[29] <87k5aksbpl.fsf@gismo.pca.it>

Peers are an integral component of marketing in the Debian Project, and section 7.2.1.4 concentrates entirely on the effect of communication in people networks in the project, and further discussion shall be postponed to there.

Repeated exposure All but the most innovative people (the earliest adopters) are unlikely to consider an innovation immediately following the first exposure. Multiple glimpses of a concept are necessary to create a kind of "familiarity" among the recipients of the marketing message:

> *Advertising is not just about providing that first glimpse, but providing a second and third exposure to something, to the point of familiarity. Many of us are early adopters and only need the first glimpse, but many more of us are not early adopters and need repeated exposure. My feeling is we probably have more early adopters than average, but they're substantially not the majority. So, one blog post, one personal recommendation, or one demo in a talk are generally not enough to get most people using a new thing.*
>
> — Joey Hess, round two <20081120001750.GA19544@kodama.kitenet.net>

A panellist drew the line between repeated exposure and the sedimentation process discussed in section 7.2.1.1: if multiple exposures are required, then the process of attaining familiarity takes longer [Raphaël Hertzog, round three[30]], due to the necessity for different perspectives of the same innovation to suit everyone and make it possible that each individual attains that level of familiarity with an innovation required to consider it (*cf.* selective perception).

Repeated exposure to an innovation may be even more effective if multiple sources are involved. The "buzz" referenced in the following quote ensures a wider reach due to better social network coverage. In addition, a larger number of sources increases the likelihood for each individual to receive a message from a trusted peer (*cf.* section 7.2.1.4) or from a familiar perspective (*cf.* "sedimentation", section 7.2.1.1).

[30] <20090413102934.GC23906@rivendell>

> *Having some buzz and excitement around a new tool or technique seems to help. If several people are blogging about using something, lots of other people will become aware of it, and start thinking about using it.*
>
> — Joey Hess, round one <20081106024658.GA10322@kodama.kitenet.net>

The alleged buzz can be grounds for a positive feedback loop: it is not always the best solution that gains popularity, but the solution which first grows more popular receives more attention and grows even more [Enrico Zini, round one[31]].

On the other side, buzz can come across as hype, which might yield opposite effects:

> *I think we're all allergic to overhype, mostly because it is and it has been such a common thing in the software industry. For this reason we have grown some "antibodies" against it, like a generally cynical attitude. This probably hurts a lot software that is too enthusiastically recommended by only few people.*
>
> — Enrico Zini, round one <20081205140625.GA22710@enricozini.org>

Content of marketing messages When composing marketing messages, the human nature of each volunteer should be remembered — everyone approaches an innovation from a subjective angle, interprets messages in the context of their own values, attitude and experience, and passes information along in a non-objective way.

> *One benefit [of an innovation] may be emphasised and some of the associated costs ignored. When a discussion happens in public this effect can be emphasised by some inflating the estimation of the costs, and others playing down the costs and pretending that the new tool will solve world hunger.*
>
> — James Westby, round one <1226589634.10403.29.camel@flash>

Even though a cost-benefit analysis is an influence of its own and subject of section 7.2.3.5, the aspect is paramount when it comes the message of marketing:

> *Initial perceived benefit is going to be a big factor in each developer's ongoing decisions; Once they invest in something it takes rather more to cause them to switch. The lesson for tool designers is (as I see it, unfortunately …) to get the marketing right early!*
>
> — Colin Watson, round two <20081201223929.GG4735@riva.ucam.org>

Maybe due to their voluntary involvement, or as a means to protect oneself from information overload, DCs tend to be sceptical of announcements that are not sufficiently factual.

[31] <20081104173445.GA508@enricozini.org>

> *When your project is still unknown, do not market it like you see on TV: it may be better to just plainly state what it does, what problems it solves, and back this with examples and tutorials of how to do it.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

A positive attitude in a discussion (or announcement) seems to send similarly encouraging messages:

> *Techniques can either be explicitly advertised, or simply discussed in a way that is stimulating the curiosity of potential users. Success stories and a positive attitude are stimulating to me.*
>
> — Charles Plessy, round one <20081112055226.GD27041@kunpuu.plessy.org>

> *Public discussions tend to be quite productive, both to increase developer awareness of a tool and to find its real goals, strengths and shortcomings — unless, of course, this discussion degenerates into a flame fest.*
>
> — Gunnar Wolf, round two <20081201014042.GB6395@cajita.gateway.2wire.net>

Related to this discussion may be the balance between "history and concision" [Charles Plessy, round two[32]]: are there (upper and lower) limits to the amount of background information that influence the forming of the first impression? For instance, if tool B is an improved version of tool A, should it be introduced to new users as such, or should they just be referred to tool B without the background information? It seems that too much background information could overwhelm, while too little information might prevent the recipient from forming an impression at all, or yield the tangential deviations alluded to in the last statement.

The right time for marketing Premature promotion can have counter-productive effects. As I shall argue in the context of my discussion on the influence of the first impression (see section 7.2.2.1), it is paramount to avoid such negative promotion.

> *If the developer is promoting something while it's in pre-alpha state, that reflects poorly on the developer's judgement and yes, would make me more reluctant to revisit the tool later.*
>
> — Steve Langasek, round two <20081130093607.GA21630@dario.dodds.net>

Similarly, going public with an idea may confront the audience with too many open questions and dilute the effect of the publicity:

[32] <20081204143515.GA8588@kunpuu.plessy.org>

> *Public discussions can also descend into a never-ending series of tangential point scoring, particularly if not enough preparation has been done. It can be difficult to identify the point at which tool development should go public.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

While a basic principle of FLOSS development is to release early and often [Raymond, 1999], it pays to prepare. The participant I had interviewed long before the study had caught my attention due to his track record of diffusing software into Debian with ease, and I asked him for advice:

> *Before releasing the code, it is important to write some kind of document/manifesto explaining what's wrong with the current situation, in detail, and exploring how your ideas can solve it. I think this works better than just dropping code in people's laps.*
>
> — Joey Hess, personal interview <20070621081621.GA2869@kitenet.net>

It seems plausible that a manifesto of this sort can help prevent discussions from degrading, because it is a foundation document on which discussions can be based. If it does not outright answer a given question, it might specify the direction such that the question will be identified as tangential beforehand and hence withheld.

7.2.1.4. Influence 4: Peercolation

Gist: Information that flows between peers is usually weighed stronger.

Summary used during study: Information spreads through people networks, and team dynamics carry ideas beyond team boundaries. Those with significant experience in an area, and who can clearly explain a tool's benefits, get more respect. People tend to favour peers they trust, or with whom they have overlaps in interest or heritage; similarly, large teams and those in charge of important packages are more influential. The cost of investigating something new cannot be avoided, but those who present the facts which we could not research ourselves are the most persuasive as they appear to help and save us time. Popularity and critical mass seem more important than individual people using a tool/technique, however. With increasing experience, people become less reliant on the opinions of others. Somewhat unexpectedly, we apply less scrutiny to projects stemming from outside Debian.

The term "peercolation" was coined by a participant at the start of the Delphi study to describe the percolation of information (and particularly knowledge of innovations) through networks of trusting peers [Christian Perrier, round one[33]].

Rogers [2003] does not mention "peers" at the knowledge stage, but only in his discussion of the persuasion stage [p. 175f] in the context of the influence of peers on the subjective evaluation of an innovation. However, this discounts the influence of information flow in peer networks, which is particularly strong in FLOSS projects, where it has been called the "primary way that ideas spread through the project" [Russ Allbery, round three[34]]. The light arrow in figure 7.1 on page 141 from the later individual adoption stages through the middle, which is labelled "peercolation" represent exactly this: the flow of information from people in the middle of the adoption cycle to those individuals at the beginning of the cycle.

I treat the influence under the heading of stage one, but it should be clear that cognitive and affective knowledge (which is primarily found at the persuasion stage, section 7.2.2) are convoluted, and that messages by peers achieve more than mere exposure. Therefore, much of what I will discuss in what is to follow also applies (or better applies) to the persuasion stage (section 7.2.2), but in FLOSS context, peer-to-peer information exchange influences adoption behaviour right from the start.

Spreading the word In some ways, peercolation is "just a form of marketing which tends to work very well among active members of the community" [Stefano Zacchiroli, round three[35]], and it is not easy to keep the two apart (*cf.* section 8.3.1.4). One participant used the following separation, which is exactly what I had in mind when splitting the two:

> *I think of peercolation as the spread of tools and techniques through normal use of them for one's work and normal discussion, whereas marketing is instead the conscious attempt to spread a particular tool or technique.*
>
> — Russ Allbery, round three <87tz1opja2.fsf@windlord.stanford.edu>

In social networks, and especially tightly interwoven ones, most information spreads between peers [Allen, 1970], even though communication in groups can decline over time if membership stays constant [Katz and Allen, 1982]. By extension, fluctuations in the membership set increase the rate of flow of information. This ties in with the homophily of peers (see appendix B.1.2), as well as the characteristics of the communication channels used.

[33] <20081104174209.GU4145@mykerinos.kheops.frmug.org>
[34] <87d4b9o4m7.fsf@windlord.stanford.edu>
[35] <20090427090909.GA10691@usha.takhisis.invalid>

When assessing the effect of peer-to-peer communication on individual adoption behaviour, it helps to distinguish between the three cases (a) where a new tool is needed to move on, (b) where an innovation is so obvious an improvement that its adoption warrants no explicit decision, and (c) where a tool is merely sought or considered. The first two of those seem naturally related, because the *status quo* is inadequate and progress is required, and the decision is mainly how to address the need. The latter case is not only concerned with a choice between innovations, but also the choice for an innovation (and against the current practice) in the first place, and a parallel to so-called preventative innovations can be identified, suggesting lower adoption rates [Rogers, 2003].

When an adoption is inevitable for one reason or another, the individual might well confer to the group as a whole, or maybe get on board of a perceived momentum, to guide decisions. Looking to trusted and respected people for inspiration helps to make and justify decisions [Enrico Zini, round three[36]].

> *A significant proportion of the Debian community — I'll even go out on a limb and say most — just want to use whatever is standard. I think a lot of people have the (fairly reasonable, overall) perception that there are many good ways of doing things, and that anything that's broadly used is probably good enough to work once they learn it. They therefore base their decision-making on what seems to have the most momentum and mass-adoption, since that's the method that they're most likely able to get support for.*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

At times, there may simply not be enough time for individual evaluation of a tool, and people tend to refer to their peers for advice, using their trust to shortcut evaluation and save the time for the implementation of an idea:

> *I see people look to others that they trust and respect for ideas on what new 'cool' tool they should be looking into next. This trust is a combination of overall visibility in the project and direct experience with people. There isn't enough time in the day to look at everything and knowing what people you trust are using or interested in is a major factor.*
>
> — Scott Kitterman, round three <200905232049.11003.scott@kitterman.com>

Following a trend leads into later influences sustainability (section 7.2.2.4) and return-on-investment (section 7.2.7.1), but also exposes the social properties of peer-to-peer transmission as constitutive component of information flow:

[36] <20090421104533.GA5652@enricozini.org>

> *Even though individually we all have our own reasons for preferring one tool or another, when looked at as a whole, peer-to-peer transmission of tools comes to dominate because it's a social property, and Debian is very much a social organization.*
>
> — Steve Langasek, round three <20090427073700.GA5755@dario.dodds.net>

This is not to say that peers provide an influence that causes unconditional adoption, but that exposure to messages about a tool by peers leads to increased evaluation [Steve Langasek, round two[37]].

> *Other people's advice is important, but in the end I will test the new tool, which means that I will be able to evaluate it by myself.*
>
> — Luca Capello, round two <87k5aksbpl.fsf@gismo.pca.it>

Centrality and exposure Those in central positions in the project are more exposed, and their perceived (meritocratic) authority does have an impact on the credibility of their messages:

> *The peercolation effect is much stronger with developers of core packages and people who are considered central to the project, since more people read their changelogs, their blog posts (even casual entries that aren't really pushing something) get more attention, and their mailing list posts are viewed as more authoritative.*
>
> — Russ Allbery, round three <87tz1opja2.fsf@windlord.stanford.edu>

Apart from explicit communication, actual use of a tool or technique in packages exerts an influence upon those investigating [Joey Hess, round one[38]], provided that the tool is visible in the package [Russ Allbery, round two[39]]. Use in important packages is relevant, as is frequent use of the tool [Luca Capello, round two[40]]. This is possibly how the multi-author format convention in changelog files spread through the project [Russ Allbery, round two[41]].

Credibility of peers Even though the concept of "thought leaders" (*cf.* lead users [von Hippel, 1986]) may be perceived as too explicit or exaggerated [Niko Tyni, round two[42]], the participants mostly agreed that messages by respected peers are weighed more than others. In places,

[37] <20081130093607.GA21630@dario.dodds.net>
[38] <20081106024658.GA10322@kodama.kitenet.net>
[39] <87wsep9a04.fsf@windlord.stanford.edu>
[40] <87k5aksbpl.fsf@gismo.pca.it>
[41] <87wsep9a04.fsf@windlord.stanford.edu>
[42] <20081130213055.GA19290@rebekka>

the group of respected peers has been referred to as "core developers", which is not an optimal term for lack of a clear definition, and a name-clash with *e.g.* Ubuntu [Loïc Minier, round two[43]].

However, it is questionable whether the set of core developers and trusted peers are congruent. An overlap is probable, but other factors come into play:

> *I have a set of people who I'd consider core developers because of the work they do. It overlaps but is not the same as the set of people whose judgement I trust. Being a core developer can be as much a function of available time or energy as good judgement.*
>
> — Joey Hess, round two <20081120001750.GA19544@kodama.kitenet.net>

We did not establish to what extent the set of trusted peers defines the set of people whose messages are weighed more than others. In places, perceived experience of the peers was the preferred metric. The following statement also adds meritocratic values to the credibility of a peer: it matters less who a peer is than what s/he has to say, and how s/he communicates it:

> *The most persuasive advocates are the ones who can clearly explain why a new tool or technique is better, who are perceived as having significant experience within the area affected by that tool or technique, and who seem to have done all of the research and experimentation that we'd all like to do but usually don't have time for.*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

This may be what enables non-central contributors to push messages out as well, although they may have to engage in more active marketing than others:

> *I've seen "this is really cool, use this!" blog posts in Planet Debian or messages to mailing lists work fairly well even if the person isn't a "core" developer or a maintainer of a major package. But I think that if one isn't perceived as core, one has to go farther on the marketing side and intentionally push stuff, not just mention it casually.*
>
> — Russ Allbery, round three <87tz1opja2.fsf@windlord.stanford.edu>

Impact and abuse The participant goes on to claim that experienced developers have a significant impact on adoptions, although that may be more a feature of care and activity than of notoriety — experience is what has taught them how to spread their ideas:

[43] <20081202215204.GA21578@bee.dooz.org>

> *The "senior" developers who post well-thought-out discussions to Planet Debian or who frequently answer questions on the `debian-devel` and `debian-mentors` mailinglists have a significant impact on what tools and techniques people use. I think the mentoring system increases this substantially; anyone who comes to project packaging through the mentor system will probably be heavily influenced by the tools and techniques used by mentors.*
>
> — Russ Allbery, round three <87d4b9o4m7.fsf@windlord.stanford.edu>

Participants have witnessed abuse of this influence towards new developers on the `debian-mentors` mailinglist [Charles Plessy, round one[44]], which calls for more eyes to prevent such instances [Colin Watson, round two[45]]. While a certain level of guidance on the use of tools and techniques is beneficial for new developers, it might also squander the capacity of new contributors to innovate:

> *I suspect that this is making it harder for new techniques to get a foothold in the project. New maintainers should be one of the groups most open to new ideas, since they have few preconceptions, but if they're forced to conform to get anything done, that capacity is squandered. I think that the project is less accepting of new ideas than it was 10 or even 5 years ago. Partly because we have good enough solutions for a lot of things we didn't have then, but partly for other, less good reasons.*
>
> — Joey Hess, round two <20081120001750.GA19544@kodama.kitenet.net>

Another component in the evaluation of peer messages seems to be related to a "magpie tendency" [Colin Watson, round two[46]] among contributors who take joy in experimenting with the new and shiny. This is not inherently a bad trait, as it is one basis for pioneering ideas.

> *Experimenting with new tools is just thrilling. This you'll find easily in people that are passionate about computer stuff. Some will get crazy about hardware, some about new algorithms, some about new tools, it all depends on your center of interest, but in the end, people crazy about computers will always have hunger for new concepts at some level.*
>
> — Pierre Habouzit, round one <20081101165828.GA26229@artemis.corp>

However, when perceived in this way, a peer's messages may not be influential to adoption decisions:

[44] <20081108080522.GA28360@kunpuu.plessy.org>
[45] <20081201223929.GG4735@riva.ucam.org>
[46] <20081201223929.GG4735@riva.ucam.org>

> *These people do not so much exert a direct persuasive influence on other developers, but rather act as a first-pass filter for bad ideas and so save the rest of the project from having to worry about them!*
>
> — Colin Watson, round two <20081215130011.GW4735@riva.ucam.org>

Conversely, one participant noticed that in Debian, it is not unheard of for a new tool to be disregarded for being "trendy" [Damyan Ivanov, round two[47]]. This tendency will be treated as part of the discussion on resistance in section 7.2.2.3.

An innovation's pedigree A question that arose during the study was in what ways an idea's pedigree affected its adoption rate.

> *Tools can gain a perception of bias from the initial development targets of the project or affiliations of the main developers. Why and by whom the tool was developed can become more important than the actual performance of the tool.*
>
> — Neil Williams, round one <1225790047.31771.202.camel@holly.codehelp>

For example, one participant claims that the DVCS Bazaar had "a bad start in Debian", because it was developed by Canonical, the corporate entity sponsoring Ubuntu (see section 2.2.6) [Pierre Habouzit, round one[48]]. A Canonical employee did not feel comfortable accepting this without pointing out that there are "many views opposing this in Debian among people who don't work for Canonical too", but also acknowledged a level of anti-corporate feeling in the project:

> *People who are paid to work on Debian-related projects are sometimes reticent to promote their company's preferred tools too heavily, knowing the strong streak of rather anti-corporate feeling in the project.*
>
> — Colin Watson, round two <20081201223929.GG4735@riva.ucam.org>

It does not seem far fetched to assume a certain level of anti-corporate feeling in the Debian Project. Debian's non-hierarchical organisation and the distributed power structure of the Debian Project is in stark contrast to the corporate world, and independence is a core principle of the Debian Project (see section 2.2.5). The same participant employed Occam's Razor to argue for the accumulation of an anti-corporate feeling in the Debian Project:

[47] <20081130213033.GA10462@pc1.creditreform.bg>
[48] <20081102170804.GA4149@artemis.corp>

> *Let us assume that there is some fraction of GNU is not Unix (GNU)/Linux users and developers with a strong distrust of corporations. Those people will naturally tend to gravitate towards non-corporate distributions, of which by far the most competent and complete is Debian. As a result Debian can hardly avoid accumulating some anti-corporate bias, simply by the fact of its existence and position.*
>
> — Colin Watson, post-study follow-up to round two statement <20090726154705.GA16966@riva.ucam.org>

Another participant responded in a way that suggests such ideological concerns were secondary: "the tool quality (and features) will always win [Luca Capello, round two[49]].

I could not determine to what degree anti-corporate feelings had an effect on adoption behaviour, nor could I find other examples where a corporate link negatively influenced a tool's diffusion.

Pedigree does not need to involve corporate involvement, but can play a role with tools of a FLOSS origin too:

> *It is the very aura of the kernel that puts people off Git. The perception is that a tool designed for kernel development would be overkill for simple userspace tasks.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

Moreover, solutions from unknown people may be regarded with scepticism, for lack of experience on the side of their authors, who may be making the same old mistakes as have been previously encountered:

> *It's not uncommon in free software to see someone new to a problem re-invent the wheel, often in an inferior way. I therefore tend to lean away from tools developed by people who are brand new to a particular problem space, and that's often strongly correlated with being unknown.*
>
> — Russ Allbery, round two <87wsep9a04.fsf@windlord.stanford.edu>

On the other hand, a credible pedigree could make it possible for a project to come "out of the blue" and get adopted [Enrico Zini, round two[50]].

There seems to be a distinction between innovations that originated in the Debian Project and those stemming from an external source. This may be because projects from the outside are likely to be further in their development cycle by the time they reach Debian, and thus come with a larger amount of inertia, and DC are more willing to accept them as they are. Innovations that come out of Debian's own ranks are likely to get noticed earlier, and are grounded

[49] <87k5aksbpl.fsf@gismo.pca.it>
[50] <<20081205140625.GA22710@enricozini.org>>

in issues with which DCs are directly familiar, which makes them more prone to "bikeshedding".[51]

The participants unanimously agreed that "the perceived merit of a developer is affected by the amount of overlap of previous ventures with the priorities and preferences of the potential adopter" [Neil Williams, round one[52]]. This could be related to the tendency of selective perception (see "sedimentation", section 7.2.1.1) and is supported by the following claim about spread of ideas among team members, who have large overlaps between each other:

> *Developers often learn new techniques through exposure to what other team members use, either through necessity or by recommendation, and new tools for collaborative development are often forged through experience in teams.*
>
> — Colin Watson, round one <20081109010317.GK4735@riva.ucam.org>

It works the other way, too: teams can also limit choice, an effect I will discuss in greater depth during the presentation of the network effects influence in section 7.2.5.1.

> *Interactions within a team are interesting, because one early adopter in a team can be enough to get a whole team exposed to and using something new. At the same time, an early adopter in a team can get frustrated when the team doesn't want to adopt something and instead wants to waste time looking at other approaches, or not change at all.*
>
> — Joey Hess, round two <20081120001750.GA19544@kodama.kitenet.net>

7.2.2. Stage two: individual persuasion

The persuasion stage begins when an individual starts to form an opinion about an innovation. Persuasion does not refer to the act of someone persuading someone else, but rather to an individual forming an attitude and considering change him/herself.

The type of knowledge at this stage is affective and subjective, as the individual puts the innovation into context of his/her own work, and engages in hypothetical and counterfactual thinking about the future [Rogers, 2003, p. 175]. S/he becomes more psychologically involved and starts actively seeking more information to reduce uncertainty about the innovation and its consequences, and gathers evaluation information [*ibid.*].

[51] *cf.* Parkinson's Law of Triviality [Parkinson, 1957] is known in Debian as the "bikeshed principle": given the technical knowledge required to discuss the building of a nuclear plant, people are happy to defer judgements to experts, but when discussing the building of a bikeshed, which is so trivial that *everyone* could do it, *everyone* wants to have a word in the discussion. http://en.wikipedia.org/wiki/Bikeshed [08 Sep 2009]

[52] <1225790047.31771.202.camel@holly.codehelp>

Social reinforcement helps reduce this uncertainty, and individuals at this stage tend towards interpersonal communication channels to find answers, rather than absorbing mass communication channels that mainly disperse information. The previously discussed peercolation influence (section 7.2.1.4) continues to have an impact at this stage, but was placed into the previous stage for the property of social networks to transmit information and knowledge, especially in FLOSS projects.

The focus in this section is on subjective perception, an important consideration given that "the individual's attitudes or beliefs about an innovation have much to say about his or her passage through the stages" [Rogers, 2003, p. 174]. Furthermore, forward-planning is central to this stage and will be dealt with as part of the discussion of the sustainability influence in section 7.2.2.4.

In Rogers' theory of diffusion, the "attributes of innovation" (see appendix B.1.1) impact at this stage [Figure 5-1, p. 170], and not at all at the later decision stage. Unfortunately, in the discussion that follows [p. 174ff], this distinction is not as clearly cut. Since the central theme of the persuasion stage is perception, and individuals only "engage in activities" in stage 3 [p. 177], I postpone discussion of influences that involve action to the treatment of the decision stage in section 7.2.3, even though some of them are related to Rogers' classical attributes (*e.g.* trialability), and would belong in the discussion of the persuasion stage, according to Rogers.

The outcome of the persuasion stage is a favourable or unfavourable attitude of an innovation. Similar to how individuals may consciously or inadvertedly linger in the knowledge phase and postpone the forming an opinion (see section 7.2.1), the formation of an attitude toward an innovation needs not directly lead into the next stage.

7.2.2.1. Influence 5: First impression

Gist: The first impression usually establishes inertia for or against an innovation.

Summary used during study: The first impression could yield further investigation by the potential adopter. Stating plainly what a tool's purpose is and what it does, documenting its benefits, and providing examples and tutorials facilitates the ensuing investigation. A concise description, which does not assume high levels of technical knowledge or skills, can help form a positive first impression.

The first impression is often a long-lasting base-line for both positive and negative judgements. It is the first subjective encounter of an individual with an innovation and hence the appropriate influence to appear first in the discussion of the second stage influences of the adoption process.

Whether good or bad, the first impression is difficult to overcome [Rabin and Schrag, 1999]. While a positive impression can elicit blinding enthusiasm, a negative impression may also postpone or prevent a proper opinion from being formed and halt or postpone the adoption process. Furthermore, the first impression can ripple through a social system and slow down adoption processes (see "peercolation", section 7.2.1.4).

Even though the following statement references an "initial experience", which was gained long after the first conscious encounter, the underlying message nevertheless exemplifies the difficulty to overcome a negative impression:

> *In one project I work on they attempted to move to DVCS-style package management to make team wide coordination easier. Unfortunately the combination of the size of the packages, the DVCS chosen, and the infrastructure used to host the repository produced a system that was so slow it was extremely painful to use. This negative initial experience has made me very reluctant to use that system again even though many people describe it as "much faster now".*
>
> — Scott Kitterman, round one <200811081804.30371.scott@kitterman.com>

One participant found this behaviour curious:

> *At first glance this reluctance to re-evaluate things seems to be somewhat at odds with the "release early, release often" spirit commonly found in free software.*
>
> — James Westby, round two <1228587563.4490.29.camel@flash>

Even though the "release early, release often" spirit is wide-spread among FLOSS community members — it is one of the defining traits of the community after all [Raymond, 1999] — the first impression is anything but rational [Rabin and Schrag, 1999], as stated by another panellist:

> *Even knowing that the first impression can colour later judgement, developers can take significant amounts of persuasion to reassess the first impression against later releases or problems, depending on other factors.*
>
> — Neil Williams, round one <1225790047.31771.202.camel@holly.codehelp>

One explanation for this may again be found in the scarcity of free time: at this early stage in the adoption process, not a lot of time was expended to form a first impression, but potential adopters are aware of the time requirements of adoptions in general (see sections 7.2.1.2 and

7.2.3.5). With a negative first impression, additional time or input is required to reassess the initial opinion.

Just like a negative first impression can cause ripples through the social system, so can a positive one. However, it seems that positive first impressions have a lesser impact, since fresh enthusiasm about a tool or technique is taken with a grain of salt: a proponent acting on the basis of nothing more than a first impression will have low credibility, since s/he cannot provide the detailed information to excite potential adopters, or his/her "magpie behaviour" may even deter (see "peercolation", section 7.2.1.4).

The bottom line seems to be that the first impression is an important milestone in the adoption process, and marketing in all aspects should aim to prevent a negative first impression, while a positive first impression is less influential and should thus not necessarily be a priority goal. This is echoed in the following:

> *Nevertheless, I think that Debian is a place where there are less possibilities for a second chance, even when there is good will to make a new start, probably because it does not take much energy to write a few negative lines and send them to a mailing list, which makes the bad opinions always come first, and discourages other persons to enter into a discussion.*
>
> — Charles Plessy, round three <20090712142503.GI29671@kunpuu.plessy.org>

Unfortunately, this might prove difficult, as a first impression can be formed inadvertedly, as described in the following statement:

> *Working on someone's package in an non-maintainer upload (NMU) scenario can have a negative impact on your perception of a tool. In an NMU you may just be interested in fixing an release-critical (RC) bug or similar, and not wanting to have to learn something new, so anything you have to do that is outside of your comfort zone can prejudice you against that thing.*
>
> — James Westby, round two <1228587563.4490.29.camel@flash>

This specific problem should be somewhat alleviated by Debian's recent policy change, which requires maintainers to document non-standard processes in the `README.source` file inside the package, providing a standardised location for documentation and instructions for how to work on a given package, and therefore improving anyone's first encounter with the tools employed.

Mission statements A clearly defined purpose or mission statement, which is understandable by all potential users seems to be an important prerequisite in the forming of a first impression, or awareness might not yield further investigation:

> *It should not take more than a few minutes to understand the overall rationale of a new tool, as well as its principles. This is helped by a good and concise description, targeted at all potential users. Often, the designers of tools tend to assume that all future users will have their knowledge, skills and wisdom, which is a fallacy.*
>
> — Christian Perrier, round one <20081107193431.GQ4145@mykerinos.kheops.frmug.org>

A mission statement may not be the first exposure to an innovation, but knowledge of the innovation might come from elsewhere. The first impression still bears significance in (individual) innovation-decision process:

> *I start using a tool* after *I gained an idea of what the tool does (the idea has usually been formed reading blog posts or documentation). Based on the idea, I already know whether I want to use the tool or not.*
>
> — Stefano Zacchiroli (his emphasis), round two
> <20081130181416.GA13001@usha.takhisis.invalid>

It seems that the benefits of an innovation need to be clear, even though a considered cost-benefit analysis does not yet happen at this stage (see section 7.2.3.5):

> *Tools for which the benefit is obvious enough (or documented very clearly) are much more likely to be widely adopted than tools or techniques where the benefit is more vague.*
>
> — Christian Perrier, round two <20090102163012.GO22308@mykerinos.kheops.frmug.org>

Several panellists also hinted at the ability to try out an innovation in this context, which will be subject of section 7.2.3.3.

A good example of a positive first impression that led to a wide-spread and quick adoption was the technique of embedding version control information into packages' metadata. There was never an official announcement[53], but one of the inventors implemented support for the idea in a core infrastructural component of the project (the package tracking system), and *de facto* standardised the idea that way. As a result, knowledge of the idea spread very quickly, and the new control information became officially supported. Later, the main driver of the technology introduced a tool which made use of the idea by adding it to an existing and wide-spread toolset, which helped its diffusion and the standardisation of the whole approach[54]. In the eyes of a panellist, everything was done right in diffusing the idea and the tool, which appeared to him as simultaneous:

[53] only http://upsilon.cc/~zack/blog/posts/2006/09/xs-x-vcs-XXX/ [23 Jul 2009], which referenced another message whose author pioneered the idea: http://lists.debian.org/msgid-search/20060729022633.GB20620@chistera.yi.org [23 Jul 2009]

[54] http://upsilon.cc/~zack/blog/posts/2007/08/debcheckout/ [23 Jul 2009]

> *Not only were the benefits clearly detailed and everyone seemed to understand that this adoption would be a big improvement, but a simple tool that used the idea was available at the same time as the proposal was made.*
>
> — Luca Capello, round three <87hc0ks9sd.fsf@gismo.pca.it>

7.2.2.2. Influence 6: Elegance

> **Gist:** Elegance is a subjective reward, but members of a community expose some common preferences.
>
> **Summary used during study:** Working on something that pleases can help increase one's efficiency. The design, quality of implementation, and technical correctness of a tool/technique can be important factors to some, but there is also the "feeling", a personal preference which cannot always be qualified, and can result in irrational behaviour.

To start, I should note that one participant questioned the neutrality of my phrasing of the above summary.

> *While I agree with the statement that intuitive analysis can result in irrational behaviour, I think the paragraph as a whole comes close to implying that anything not based on strict logical deduction is irrational, which is an interesting and rather controversial philosophical hypothesis not very strongly borne out by your supporting statements. I don't think you necessarily meant to give this implication. Perhaps simply separating the last clause out into a separate sentence would make it clear that it isn't a necessary association.*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

It is unfortunate that I inadvertedly conveyed such an implication to some or all participants. Not a single participant picked elegance as one of the most salient influences in the third round, which may have been a result of the misrepresentation.

Elegance is possibly the most subjective of all influences, as it is about aesthetics and preferences, and sometimes even perfectionism. This also makes it hard to define and has many synonyms and related words, including "aesthetics", "rightness", "cleanliness", "feeling", and "excellence".

> *Being technically correct is an obvious requirement for an idea to gain popularity. But there is also one aspect that is not very easy to clearly define: "rightness". To some people it just feels right to use debhelper instead of Common Debian Build System (CDBS). Others feel the opposite. Both camps feel that their choice is "right".*
>
> — Damyan Ivanov, round one <20081107203114.GV8178@pc1.creditreform.bg>

Even though one participant claimed that such elegance was secondary to productivity for himself [Christian Perrier, round one[55]], he also admitted that the strive for technical excellence is common in the project:

> *Many DCs are very keen about technical excellence. So, for some of us, the quality of implementation might sometimes be an important factor to decide about adopting a tool or not.*
>
> — Christian Perrier, round one <20081107193431.GQ4145@mykerinos.kheops.frmug.org>

According to another participant, this preference is related to perfectionism among contributors, who are not told what to do, do not work by deadlines, and simply want to properly maintain their packages [Stefano Zacchiroli, round one[56]]. In fact, perfectionism has been identified as a core cultural traits of the Debian Project [Scott Kitterman, round two[57]].

A similar claim was made by another panellist, who draws the line towards professional appearance and pride in his work:

> *One of the most significant factors for tool adoption for me is a perception of "cleanliness." A lot of problems have clean and less clean solutions, and even if it takes a bit more time, I tend to gravitate towards methods that feel "clean." Usually that's strongly related to efficiency, but it can involve other things as well, such as the appearance and quality of the output (of debian/changelog files, for instance, or commit messages in a VCS). I want to take pride in my work, and I look for tools that help craft a professional product even more than I look for whatever is the most efficient or productive.*
>
> — Russ Allbery, round two <87wsep9a04.fsf@windlord.stanford.edu>

It is not hard to imagine how this translates into efficiency and motivation, and people prefer to engage in work they enjoy:

[55] <20081107193431.GQ4145@mykerinos.kheops.frmug.org>
[56] <20081109101430.GA11190@usha.takhisis.invalid>
[57] <13222-SnapperMsgD8DB99B6C56F29A6@[75.196.134.29]>

> *Aesthetics is part of the efficiency. I'm more prone to be efficient and willing to modify something that pleases me, than something horrible and broken.*
>
> — Pierre Habouzit, round one <20081102170804.GA4149@artemis.corp>

Elegance and aesthetics are unquestionably highly subjective and vague. All the more surprising is that Debian exists around an elegant body, which has evolved and become iteratively enshrined in the Debian Policy:

> *To a large extent, this kind of thing tends to rest on evolutionary development, at least in terms of style. Perception of elegance and familiarity have something in common. Finding the minimum interoperability requirements that need to be expressed in policy is a useful exercise which often improves the simplicity and robustness of the end result.*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

The comparison between elegance and familiarity is an interesting one, and can be interpreted in two ways: something that seems familiar can appear elegant (unless it is notoriously familiar; *cf.* also the discussion on compatibility in section 7.2.3.4), while familiarity with a domain may also allow to "hit the nail on the head" in terms of elegance:

> *Debian Developers (DDs), particularly longer-standing ones, generally seem to have a notion of what kinds of approaches fit well into the Debian system. Joey Hess has a particularly good track record of hitting this nail on the head (debhelper, some of `devscripts`, the general design of the installer, etc.), although there are others with a similar talent. Some of the important themes here seem to be modularity, adaptability, and a general idea of enforcing just the right amount of policy (very tricky) and leaving the rest up to the developer.*
>
> — Colin Watson, round one <20081109010317.GK4735@riva.ucam.org>

Nowhere are the specifications of what is considered elegant in Debian written down, but their existence cannot be denied. For instance, the Debian control file format seems widely preferred over *e.g.* Extensible Markup Language (XML) [Colin Watson, round one[58]], and abusing behaviours/results for unrelated behaviours/results is frowned upon — the recent example here might be the inclusion of the homepage in the long package description, which was considered wrong and yielded the institution of a proper homepage data field, which seemed "clearly right" [Stefano Zacchiroli, round one[59]], even though consensus for that had previously not been found [Joey Hess, round two[60]] (*cf.* section 7.2.8.1).

[58] <20081109010317.GK4735@riva.ucam.org>
[59] <20081109101430.GA11190@usha.takhisis.invalid>
[60] <20081120001750.GA19544@kodama.kitenet.net>

Due to the high subjectivity of perceptions of elegance, impressions can turn into what is commonly called "religious belief": clinging to one's opinion about a tool or technique and defending it against all forms of criticism, often without any facts to back up the claims. This is especially common in cases where two or more tools or techniques are more or less equivalent in their features and possibilities (such as most DVCSs and text editors), but their usage paradigms differ sufficiently to force choice of one or another, which is thenceforth often considered to be more elegant, and religiously defended. Herein can lie the epicentre of resistance, topic of the next section.

7.2.2.3. Influence 7: Resistance

Gist: Resistance helps to improve ideas, but may have to be actively overcome.

Summary used during study: While initial resistance to an idea can help separate good from bad ideas, pushing too hard, or taking away the option to continue with the old method, will yield further resistance. Resistance can only be met with conversion instructions, support offers, and patience. Lack of interest is closely related to resistance.

Resistance to change is a common negative influence to adoption behaviour, but not without positive aspects. It is responsible for a level of inertia among individuals and the project as a whole, which is hardly surprising, given the size and complexity of the Debian Project.

> *I think it is inertia: you have settled on a workflow that "does the job" and even if it has some glitches, it is generally okay, and the corner cases happen not that often. Even if a change would improve things, the improvements have to be significant, in order to justify the efforts.*
>
> — Damyan Ivanov, round one <20081107203114.GV8178@pc1.creditreform.bg>

It seems well understood among project contributors that tools and techniques can always be improved, but at some level, cost and benefit diverge.

> *There is too much process/tool churn. Doing things a slightly better way is nice, but it's probably best not to distract others for minor changes as each time we stop to learn/investigate something new, we aren't doing actually productive work.*
>
> — Scott Kitterman, round two <200812070057.25291.scott@kitterman.com>

The cost-benefit ratio will be subject of section 7.2.3.5, because an explicit cost-benefit consideration is made as part of a decision for or against a tool, if such a consideration is made at all. However, a projected cost-benefit ratio may be made already at this stage and result in resistance.

Reasons for resistance other than cost-benefit analyses (but still in part related) include time scarcity, a preference to get "actual work" done, lack of understanding of the proposed new concepts, and a categorical unwillingness to part with existing approaches [Pierre Habouzit, round three[61]]. Sometimes resistance can also be grounded in an outdated or simply static perception of a tool or technique (*cf.* sections 7.2.2.1 and 7.2.2.2).

Antagonism is hard (if not impossible) to overcome and can discourage those trying, if too much time is spent discussing (and voicing resistance) than accepting a solution and working to bring it up to an acceptable standard [Pierre Habouzit, round three[62]] (*cf.* section 7.2.6.1).

On the other hand, if resistance can decrease with time, it helps separate the good from the bad ideas — bad ideas are unlikely to withstand resistance for longer periods of time [Damyan Ivanov, round two[63]]. As a result, the project does not lose time with tools or techniques that will not survive, and can avoid having to recover from problems caused by the premature adoption of such approaches [Guillem Jover, round three[64]].

Overcoming resistance In general, change for the sake of change does not make sense [Niko Tyni, round two[65]]. If change is sought, this may translate into a need for active contributions by those desiring the change, such as communicating the reasons for change, facilitating the adoption with support options and demonstrations, or chunking change into smaller steps, (*cf.* section 7.2.1.2).

> *To defeat resistance, you have to write documentation to translate from the old to the new set of commands, provide support/help in the early stages of adoption, and be patient. If you can demonstrate the new toolset — how it does the job, pros and cons —* before *the migration, that's a plus.*
>
> — Damyan Ivanov, round two <20081215094128.GU8107@pc1.creditreform.bg>

Inadequately preparing or supporting for change can cause loss of interest. One participant argued that the disorganised state of the Wiki page tracking the discussion on machine-readable copyright files made the process so inaccessible that potential participants turned away and moved on [Charles Plessy, round three[66]]. Conversely, proposals whose drivers took care to main-

[61] <20090424103539.GA26076@artemis.corp>
[62] <20090424103539.GA26076@artemis.corp>
[63] <20081215094128.GU8107@pc1.creditreform.bg>
[64] <20090422045458.GA14006@gluon.hadrons.org>
[65] <20081130213055.GA19290@rebekka>
[66] <20090712142503.GI29671@kunpuu.plessy.org>

tain the available information and actively managed the discussion, have processed quickly in the project (*cf.* Debian Enhancement Proposals (DEP)1[67]).

When providing adequate information is not enough, and change is not met with resistance, but disinterest cannot be overcome, it may become necessary to implement migrations for those unwilling (or unable) to do it themselves. I witnessed several instances of this form of change driving in the context of the Python transition in Debian, where drivers provided step-by-step migration instructions, or simply offered to implement the necessary changes themselves.

Getting the amount of active driving of change right is hard, but paramount. Ideas need time to sediment and be understood by the majority (see section 7.2.1.1). Other reasons for resistance add to this and increase the level of patience required by those trying to overcome the resistance. Not allocating enough time and patience may increase resistance even more [Damyan Ivanov, round two[68]]. Ideally, change is optional and forestalls resistance:

> *I find I'm very resistant to changes that I think will force me to change my workflow to continue to participate. I'd like the option to change and do better, but what I have now doesn't entirely suck and I'd like the option to not change too.*
>
> — Scott Kitterman, round two <200812070057.25291.scott@kitterman.com>

Although the topic of the later discussion on consensus in section 7.2.6.1, obtaining feedback on controversial topics can keep resistance low.

> *I wouldn't fancy my chances of propagating any sort of potentially-controversial technique unless I had already canvassed opinions of some other well-known developers and made sure that they thought I was doing the right thing.*
>
> — Colin Watson, round one <20081109010317.GK4735@riva.ucam.org>

Resistance as self-protection Resistance can also be a protective measure. Inadequately trying to overcome such resistance might cause people to retract (or resist) even further.

[67] `http://dep.debian.net/deps/dep1.html` [28 Aug 2009]
[68] <20081130213033.GA10462@pc1.creditreform.bg>

> *None of us have as much time to spend on learning new tools as we'd like, and I think we all therefore feel at least a little guilty about not "keeping up" and being behind the times on tool adoption. We know there are better things out there and just haven't had time to process them. New recommendations therefore add more onto an already overloaded queue and can produce feelings of guilt, and people usually react with a bit of hostility to things that make them feel guilty.*
>
> — Russ Allbery, round two <87wsep9a04.fsf@windlord.stanford.edu>

Other instances of self-protection can often be witnessed in the context of project-wide transitions or changes, which might affect maintainers of large numbers of packages more than the majority. Therefore, those might put up resistance in order to defend themselves of an increased workload due to a migration [Charles Plessy, round three[69]]. Such inertia needs to be taken into consideration when driving change.

Similarly, even though a new approach may constitute an improvement over previous ways, it may not result in greater efficiency for the individual [Neil Williams, post-study follow-up to round two statement[70]]. Scalability, or better collaboration may call for solutions that increase the net time requirement on the part of the adopter, and that may create grounds for resistance:

> *Everyone tries to work as much as possible in the limited free time s/he has. This means that a new tool/technique increasing the time needed to fulfill a task will not be adopted, no matter how better coded, elegant or scalable it is.*
>
> — Luca Capello, round three <87hc0ks9sd.fsf@gismo.pca.it>

This relates to the later discussion on cost-benefit considerations and specifically the four combinations of individualist/collectivist cost/benefit (see section 7.2.3.5).

7.2.2.4. Influence 8: Sustainability

> **Gist:** Faith in the development direction and future of a tool or technique makes it a sustainable choice.
>
> **Summary used during study:** We want to put tools/techniques to use in our own ways, rather than change our ways to fit them. We want confidence that a tool/technique develops in the right direction, and that its maintainers incorporate feedback, allowing users to influence that direction; otherwise, time invested now might be lost later. Personal friction with/dislike of a tool's author can get in the way of cooperation.

[69] <20090712142503.GI29671@kunpuu.plessy.org>
[70] <20090821174258.79cd8f4f.codehelp@debian.org>

With the exception of users chasing "the new and shiny" (*cf.* "peercolation", section 7.2.1.4), DCs tend to seek tools and techniques that will not disappear, become neglected, or radically change shortly after having been implemented, because they could not always continue maintenance in such events [Stefano Zacchiroli, round one[71]]. This tendency appears to grow with experience as potential users gain a more long-term view and know better what they are looking for [Pierre Habouzit, round one[72]] and is related to the fact that contributors may use tools in not-quite-expected ways to suit their workflows [Luca Capello, round two[73]].

Under the heading of sustainability fall activities related to forward planning, an activity common to the persuasion stage [Rogers, 2003, p. 175]. Before trying out a tool, potential adopters already mentally apply innovations in the context of their work, and implicitly consider many of the influences that I present in later stages, specifically in stages 3 and 4 (see sections 7.2.3 and 7.2.4). Considerations about future influences already have an impact at this stage, but need not explicitly happen until later [*e.g.* Raphaël Hertzog, round one[74]].

Long-term planning There are good reasons for a long-term view on adoptions: on the one hand, no one wants to invest time that later turns out to have been wasted; on the other hand, discontinued or neglected tools end up being more work to use in the long run, possibly forcing people to find and integrate new means to achieve something.

Sustainable approaches are those that are available and operational until they are no longer used. No certain method of prediction of a tool's future exists, but a number of considerations are made at this stage to evaluate an innovation in this respect. For instance, the way a tool or technique is maintained, and the community that has formed around it, are standard points of evaluation:

[71] <20081109101430.GA11190@usha.takhisis.invalid>
[72] <20081101165828.GA26229@artemis.corp>
[73] <87k5aksbpl.fsf@gismo.pca.it>
[74] <20081107094111.GA25888@ouaza.com>

> *The point of a tool, put very broadly, is to leverage other people's work so that you don't have to re-invent the wheel. Since Debian packages change and software changes and requirements change, the tool needs to evolve. This requires an active development community. Tools that aren't being actively developed end up being more work to use, since the problems that crop up aren't spread across a large user-base and hence fixed in a way that feels almost magical.*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

As a consequence, popular tools are often considered safer investments [*ibid.*], although the many perfectionists in the project may not let that be a factor [Stefano Zacchiroli, round two[75]].

There is also the danger that the developers of a tool or technique might take it into an undesirable direction [Colin Watson, round two[76]], which would require a code fork or an alternative to be sought, both of which could defeat the benefits of using the tool in the first place. A strong sense of direction, as well as the awareness of feedback from users can help alleviate this concern [Enrico Zini, round two[77]]. Consensus, to which I will also return in section 7.2.6.1, is perhaps a better guideline than the satisfaction of individual requests:

> *The percolation of experiments and ideas in our collective packaging practices has been vital in building good packaging helpers; individual requests for changes there may not be satisfied directly or immediately, but it's clear to me that the people developing such tools are trying to synthesise a wide variety of views from their users. Clear consensus, whether that be on mailing lists or bug reports or elsewhere, is a very useful indicator.*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

While it is somewhat contrary to the diversity of FLOSS, the lack of alternatives can be positively related to sustainability, which is reflected in the following statement about libraries, and should similarly apply to tools and techniques:

> *I believe that one advantage of the Python "there is only one way to do it [except where we are not looking]" philosophy over the Perl "there is more than one way of doing it" approach is in libraries: you do not want five libraries to do the same thing, because you will risk choosing the one that will end up being unmaintained, and having to learn a new one, and rewrite part of your code.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

[75] <20081130181416.GA13001@usha.takhisis.invalid>
[76] <20081201223929.GG4735@riva.ucam.org>
[77] <20081205140625.GA22710@enricozini.org>

Maintainers' role in adoptions Knowing that evaluation of maintenance of a tool happens prior to its adoption, maintainers can do their part to help the adoption.

> *A responsive maintainer gives the impression that he/she is "working for you", which is quite a motivation for becoming "faithful" to that project.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

An example of maintainer participation in adoption was recounted to me in an interview with Delphi participant Joey Hess long before this study. Joey was publishing a series of articles about his involvement with FLOSS, and in one he stated that he had "spent a lot of time tracking the market share of debhelper versus its competitors and working on improving it."[78]:

> *I used to crawl the entire source archive to figure out how many packages were using debhelper* vs. *alternatives, and I had a graph for that for several (maybe 5) years. I also had a graph of which commands were used how much. Improving market share was mostly a matter of figuring out why people were not using it and adding the features they needed, and of reponding to bugs and feature requests quickly and uploading frequently.*
>
> — Joey Hess, personal interview <20070621081621.GA2869@kitenet.net>

Heritage and influence One of the benefits of FLOSS development is the ability to get involved. An overlap in heritage (*cf.* "peercolation", section 7.2.1.4) can be seen favourably:

> *If the heritage matches the interests of the developer, significant flaws can be ignored in the hope of future fixes. If the developer has no wish to be involved in the projects that gave rise to the tool, the tool can be dismissed as "specialist", even if some functionality exists to broaden the support.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

A tool's heritage, and more specifically its author(s), have an influence on choice, possibly more than DCs are willing to admit [Joey Hess, round one[79]], but the reasons seem to be pragmatic, rather than personal, and the ability to influence a project's direction, or modify a tool's capabilities appear to be in the foreground:

[78] http://joey.kitenet.net/blog/entry/ten_years_of_free_software_--_part_5_debhelper/ [26 Aug 2009]

[79] <20081106024658.GA10322@kodama.kitenet.net>

> *I don't think "personal opinions about the author" are really what's at work. If we know an author is not likely to listen to feedback about his work, or if we find it difficult to work with the author for other reasons, then we are less likely to invest time in learning a tool that author maintains. This is really a practical assessment of how effective we (in general) will be at getting our changes and improvements rolled back into the tool. Tools that can't be improved in response to changing circumstances are developmental dead-ends.*
>
> — Steve Langasek, round two <20081130093607.GA21630@dario.dodds.net>

The ability to influence — and continue to be able to influence — a software is another important consideration:

> *Don't forget that we are free software developers, and we like to shape the things we use. If too many decisions have already been taken then we will feel limited in how much we can do that.*
>
> — James Westby, round two <1228587563.4490.29.camel@flash>

The degree to which an innovation is a community effort and peer review has happened plays into this desire [Guillem Jover, round two[80]]. Projects that have not "smoothed off through encounters with other people" will be less trustworthy, similar to "how a Wikipedia page that's only had one editor tends not to be as trusted as one that might have started out very bad but has had a lot of people work on it" [Joey Hess, round one[81]]. This puts a challenging twist on the question of when to go public with an idea (see "marketing", section 7.2.1.3).

> *I think we look for the right decisions, but we also look for the right kinds of missing pieces. If something that is usually there is missing, or something that is usually missing is there, then we will be suspicious.*
>
> — James Westby, round two <1228587563.4490.29.camel@flash>

One participant suggested that adopters might look at initial discussions and how the early flaws got dealt with [Joey Hess, round one[82]], but another participant doubted that "many people dig into a concept history before trying it. It takes time, and one can get more impressions by getting hands dirty with the tool" [Damyan Ivanov, round two[83]].

[80] <20081130232243.GA13121@zulo.hadrons.org>
[81] <20081106024658.GA10322@kodama.kitenet.net>
[82] <20081106024658.GA10322@kodama.kitenet.net>
[83] <20081130213033.GA10462@pc1.creditreform.bg>

7.2.3. Stage three: individual decision

By the time individuals reach the third stage, they have formed favourable opinions about an innovation and "engage in activities that lead to a choice to adopt or reject an innovation" [Rogers, 2003, p. 177]. The innovation is not put to use until stage 4 (section 7.2.4), but individuals start to interact with the innovation, as opposed to the mental exercises and considerations which were subject of stage 2 (section 7.2.2).

The influences at this stage cover learning about how to use an innovation by inspecting examples, documentation, and trying it out. A potential adopter places the innovation into the context of his/her existing practice and weighs the costs of adoption *vs.* the consequences of adoption.

The outcome of this stage is either a decision to implement the innovation, or a form of rejection. I will delve into these possible outcomes as part of my later discussion on the cost-benefit influence in section 7.2.3.5.

7.2.3.1. Influence 9: Examples vs.documentation

Gist: Examples and documentation are two sides of a coin, with merits to both.

Summary used during study: Examples are more practical and easier to grasp than documentation, and many of us (not just beginners) work off templates. However, examples do not replace documentation.

Often, a good way to get a feel for the consequences of an adoption is to inspect examples of how the innovation is used. Such examples may be found in other people's work (such as other packages), but they can also be provided in the form of tutorials. The knowledge available in these kinds of sources is termed "how-to knowledge" [Rogers, 2003, p. 172f] and is more accessible than full-blown documentation, the subject of section 7.2.3.2.

> *Examples are often better than full documentation, which can be really overwhelming. When I want to check out a new language or tool, I go to the "tutorial" and "examples" or "snippets" page, because there I can get a first idea of how an innovation is used.*
>
> — Pierre Habouzit, round two <20081122152202.GA30285@artemis.corp>

Examples may also facilitate trying out an innovation (*cf.* section 7.2.3.3), and convey an impression of its potentials:

> *A short "make it work in 5 minutes" example is a dream. The example doesn't really have to go into all details of the possible uses of the tool, but make it clearer that the tool will be easy to adopt, and allow for progressive learning.*
> — Christian Perrier, round one <20081107193431.GQ4145@mykerinos.kheops.frmug.org>

The availability of examples *per se* also plays a role. First, one participant deemed it useful to go back to examples after not using a tool for a while to help bootstrap him back into understanding what he is doing [Scott Kitterman, round two[84]].

Working off templates Examples can also be templates that DCs base their work on, rather than starting from scratch every time they put a tool to use. This aspect also enhances the effect of momentum and peercolation (*cf.* section 7.2.1.4), because a larger number of adopters also increases the amount of examples and recounted experiences, which a potential adopter can use to experiment with and use an innovation [Stefano Zacchiroli, round one[85]]. Similarly, there tend to be more examples for general-purpose tools [Colin Watson, round one[86]], and tools providing templates as a basis for people to work from are very influential:

> *I believe that many packaging works are released under the GNU Public Licence (GPL) because the dh_make `debian/copyright` template says so. If it would promote the machine-parseable format, most new packages would use it.*
> — Charles Plessy, round two <20081204143515.GA8588@kunpuu.plessy.org>

Examples do not, however, replace documentation:

> *As long as there's also reference documentation; too many tools come with precious little other than examples, and many developers don't find that good enough.*
> — Colin Watson, round two <20081201223929.GG4735@riva.ucam.org>

7.2.3.2. Influence 10: Quality documentation

> **Gist:** Properly maintained documentation is required for widespread adoption.
>
> **Summary used during study:** Documentation is a necessity, especially when a tool/technique diverges far from existing processes. Mailing list archives and source code are not sufficient for wide-spread adoption. Instead, documentation needs to be kept up-to-date and trimmed to not become overly verbose. Early adopters want background information and care about motivation, while later adopters seek tutorials.

[84] <200812070057.25291.scott@kitterman.com>
[85] <20081109101430.GA11190@usha.takhisis.invalid>
[86] <20081109010317.GK4735@riva.ucam.org>

While knowledge during the first stage was mainly about awareness (see section 7.2.1), and the previous influence on examples covered "how-to knowledge" (see "examples", section 7.2.3.1), quality documentation adds "principles knowledge" [*cf.* Rogers, 2003, p. 172f], but also includes the other types of knowledge — good documentation covers the full spectrum:

> *Some users want tutorial documentation; some users want reference documentation. The best tools come with both.*
>
> — Colin Watson, round two <20081201223929.GG4735@riva.ucam.org>

Potentially a function of sedimentation of ideas with time (*cf.* section 7.2.1.1), a plausible relation between users and the preferred type of documentation are adopter categories (see section 3.1.2):

> *For techniques, clarity and communication of the mindset and methodology are more important for me than step-by-step recipes, although I think this varies by person. (Probably the earlier of an adopter one is, the more one wants the mindset and background.)*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

The same partitipant later stated that Debian, as a result of its volunteer nature, has a higher percentage of early adopters than other projects he knows [Russ Allbery, round one[87]]. By extension, documentation conveying the "mindset and background" may be more important than examples for DCs.

Centrality and persistence Documentation in FLOSS is notoriously lacking, which may be because DCs prefer programming machines to writing human-accessible texts [Andreas Tille, round two[88]]. As a consequence, developers of tools and techniques often assume the position that "code is its own best documentation", but source code may be hard to understand and may not provide the information needed by a user. This could lead to a reduction in the adoption rate [*ibid.*].

In addition, the plethora of communication media make it easy to publish shorter, less-permanent announcements, examples, and tips. Publishing by those media may not require as much time as writing proper documentation, but the time expended is gone nonetheless. While such documentation would be found in a central location, messages spread across discussion media are not as readily found.

[87] <87wsep9a04.fsf@windlord.stanford.edu>
[88] <alpine.DEB.2.00.0812050821200.9613>

> *Mailing list discussion is only useful for a brief period, many new tools suffer from not putting the results of discussions into the documentation, causing other potential adopters to re-invent the wheel or abandon the tool without further consideration.*
>
> — Neil Williams, round one <1225790047.31771.202.camel@holly.codehelp>

Verbosity and completeness The title of this influence is "quality" documentation, because documentation needs to cater to different readers by offering tutorial and concept knowledge (*cf.* above), approach problems from multiple angles (*cf.* "sedimentation", section 7.2.1.1), and be maintained in a way that it does not grow too verbose and accumulates outdated information [Colin Watson, round two[89]].

Verbosity in this context also refers to the level at which concepts are presented and can grow to overwhelm if it restates obvious or prerequisite knowledge [Enrico Zini, round two[90]]. Documentation should concentrate on the corresponding tool or technique:

> *The importance of documentation is directly proportional to the amount of divergence from similar tools or existing workflow patterns.*
>
> — Neil Williams, round one <1225790047.31771.202.camel@holly.codehelp>

Conversely, documentation needs to be thorough and complete. The following statement is related to transparency, which is a later influence presented in section 7.2.4.2:

> *Thorough and complete documentation includes the details of how to reproduce or fix what the tool does manually, if necessary.*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

Finally, documentation is also a sign of maturity and stability of a project, and thus can have a strong impact on sustainability considerations (see section 7.2.2.4, and also the discussion of maturity in section 7.2.5.3):

> *Documenting ideas can be seen as a sign that they are serious and get stable.*
>
> — Gunnar Wolf, round three <20090420040612.GA6715@cajita.gateway.2wire.net>

7.2.3.3. Influence 11: Trialability

> **Gist:** Trying out a tool should be as easy as possible, since that is among the best ways to form an impression.

[89] <20081201223929.GG4735@riva.ucam.org>
[90] <20081205140625.GA22710@enricozini.org>

Summary used during study: We evaluate new tools/techniques in the context of our own use-cases to determine their worth. A tutorial, especially one which identifies problems, and the ability to try out the tool with minimal time-investment can help with that process.

A popular way to get a feel for a new tool or technique is through trying it out. This influence is congruent with the trialability attribute of an innovation defined by Rogers [2003, p. 258]. He states: "The personal trying out of an innovation is one way for an individual to give meaning to an innovation and to find out how it works under one's own conditions. A personal trial can dispel uncertainty about a new idea."

> *The first time I try out a new system, am I able to do anything (even something silly) with it in the first 10 minutes of using it? If not, I'm likely to leave it there and try it again "eventually", but I'll tend to pick it up with a rather pessimistic approach, given the previous failure.*
>
> — Enrico Zini, round one <20081104173445.GA508@enricozini.org>

Trialability is thus a measure of the ease of experimenting with an innovation. Aspects of previous influences, and in particular chunking (see section 7.2.1.2), continue to play a role in this. For instance, an innovation that comes in smaller chunks of which each can be tried independently, has a higher degree of trialability than a monolithic tool. On the other hand, tools that require infrastructure to be set up around them have lower trialability.

> *`lintian` has a great trialability because you don't actually have to do anything to test-drive the tool, you just run it. `svn-buildpackage` requires a bit more involvement.*
>
> — Gunnar Wolf, round three <20090813224804.GD12437@cajita.gateway.2wire.net>

Nevertheless, trying out an innovation may not be the first activity in which a potential adopter engages at this stage.

> *I think a lot of people behave this way, but we should not forget there's still people who enjoy getting well informed before even starting to try something new.*
>
> — Guillem Jover, round two <20081130232243.GA13121@zulo.hadrons.org>

7.2.3.4. Influence 12: Compatibility

Gist: The less time is required to learn and adapt to a new tool, the easier its adoption will be.

> **Summary used during study:** Learning a new tool requires investing time. Tools that can be used without having to learn them, or which automate what you are already doing are readily adopted, especially when they come with migration instructions. Tools that build on known concepts or are "finger-compatible" are easier to learn. Only drop-in replacements can completely superseed a tool though.

Compatibility is another innovation attribute in Rogers' diffusion theory, but unlike trialability (see section 7.2.3.3), there is no one-to-one overlap. The influence discussed in this section only concentrates on technical compatibility. Need and value compatibility are part of earlier influences, such as marketing, peercolation, or sustainability (sections 7.2.1.3, 7.2.1.4, and 7.2.2.4 respectively). Complexity, another of Rogers' innovation attributes, also closed relates to this influence, since complexity is about "the degree to which an innovation is perceived as relatively difficult to understand use" [*ibid.*, p. 257]

Rogers' definition of compatibility still applies in the context of this section, even though I only concentrate on technical aspects: "the more compatible an innovation is, the less of a change of behavior it represents" [p. 245]. Compatibility is hence a measure of how well an innovation fits with existing practice and how little change (and thus cost) is required to put it to use. Given the limited amount of time among volunteers in the Debian Project, or — alternatively — the large number of tasks to be done, it is not surprising that compatibility is an important influence in the project, and considered part of the merit of a tool:

> *Developers will build up their own arsenal of tools and their accompanying workflows, and changing that will cost productivity, so it is important that the impact of the switch be limited, or at least proportional to any productivity increase that will result.*
>
> — James Westby, round three <1240786091.3291.66.camel@flash>

> *To me the fact that the tool fits with what I do is part of its technical* merits *and benefits to some extent.*
>
> — Pierre Habouzit (his emphasis), round one <20081101165828.GA26229@artemis.corp>

The easiest tools to adopt are those that automate existing practice [Russ Allbery, round one[91]]. As workflows differ between contributors [*cf.* Neil Williams, round one[92]], this can be difficult to achieve, but building upon standards (see section 7.2.8.1) is likely to keep the delta low and facilitate adoptions [Enrico Zini, round three[93]].

[91] <87ej1m7yye.fsf@windlord.stanford.edu>
[92] <1225790047.31771.202.camel@holly.codehelp>
[93] <20090421104533.GA5652@enricozini.org>

Low compatibility can result in negative bias toward an innovation, and rejection instead of adoption, because the innovation cannot be properly evaluated in the context of existing practice, and its implementation expected to be more involved.

> *If the tool is so distinctive that it distorts normal workflow patterns or requires adjustments to long-established patterns, the perceived "quality" of the tool will be diminished.*
>
> — Neil Williams, round one <1225790047.31771.202.camel@holly.codehelp>

This concern applies on the technical level, as well as on the level of concepts. For instance, one participant criticised Git for its use of vocabulary inherited from the development environment, rather than using compatible language shared with existing VCSs [Neil Williams, round two[94]].

Intuition, learning curve, and familiarity A fully compatible tool is one an adopter can pick up intuitively and put to use without thinking [James Westby, round two[95]], and which does not get in the way [Pierre Habouzit, round one[96]]. "Finger-compatibility with previous forms of similar tools is important" and has helped *e.g.* Subversion's adoption rate [Colin Watson, round two[97]].

Technical compatibility only applies to potential adopters already versed in a domain, not to absolute newcomers. However, multiple areas of overlap with other tools and previous experience can exist even for newcomers, and re-use of concepts, vocabulary and usage patterns, if possible, helps to flatten the learning curve:

> *"Positive knowledge transfer" is another, more nuanced way to describe this point - the learning curve for a user unfamiliar with VCSs at all would be very different because there's a lot more conceptual knowledge that needs to be acquired first, not just familiarization with the interface to those concepts.*
>
> — Steve Langasek, round one <20081114092546.GA26586@dario.dodds.net>

Chasms that cannot be crossed through this form of positive knowledge transfer can require specific documentation explaining the migration. This was the case during the adoption of the Python helpers and the associated wiki page, which detailed the steps and helped prevent forgetting of critical ones [Loïc Minier, round one[98]].

[94] <20081121184632.106f2666.codehelp@debian.org>
[95] <1228587563.4490.29.camel>
[96] <20081101165828.GA26229@artemis.corp>
[97] <20081201223929.GG4735@riva.ucam.org>
[98] <20081111133148.GC18548@bee.dooz.org>

The next example of where compatibility has an influence in adoptions references Unix principles, which are at the core of the influences at stage 4 (see section 7.2.4):

> *For example, the tools in the `moreutils` package are trivial to pick up, because they reuse so many ideas of the Unix world we're used to. A useful tool like `mmv`, on the other hand, does not get picked up as readily because it introduces a new syntax.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

7.2.3.5. Influence 13: Cost-benefit

> **Gist:** Everyone weighs adoption cost against tool/technique benefit, albeit usually on the basis of perceptions, or even subconsciously.
>
> **Summary used during study:** Suboptimal tools can be painful to use, or become a nuisance in the future, but they might just get the job done *now*. A simple tool might also just be enough for a specific use-case. A high barrier of entry — having to spend a lot of time before being able to reap benefits — can deter potential adopters. While change for the sake of change can rarely be justified, DCs appear keener to experiment than members of other projects. This either offsets the cost-benefit analysis, or decreases its importance.

DCs will, without doubt, consider tools or techniques that automate manual labour [Niko Tyni, round one[99]], and reduce maintenance costs [Loïc Minier, round one[100]]. However, they are also aware of the costs of adopting an innovation.

> *While many people may acknowledge the benefits of a tool, they will be reluctant to spend too much time on it before reaping the benefit, as they will want to be "getting things done".*
>
> — James Westby, round three <1240786091.3291.66.camel@flash>

A new tool or technique may be perceived

The decision for or against a tool can involve an explicit consideration of whether the benefits of a tool are worth the costs needed to invest in order to implement a tool. This influence has been referenced in the discussion of several earlier influences, but I postponed its discussion to the end of stage 3, the individual decision stage, because the bulk of the cost is initiated with a decision, not earlier. Of course, to arrive at the point where an individual is capable of making a decision, s/he will have already expended time, and this cost may well factor into or taint a cost-benefit analysis.

[99] <20081107183929.GA5087@rebekka>
[100] <20081109193850.GD3077@bee.dooz.org>

The costs of an adoption can be multifarious and usually exceed the mere costs of migrating data, as suggested by the following statement describing the transition from Subversion to Git:

> *Migration costs include more than just running `git-svn` with the right options and convert the svn "tags" into Git tags. People need to change they workflow a bit, their fingers need to learn the new key sequences. Also, in the svn → Git case, the way they think about version control needs to be altered. People need to be convinced that these changes are for good and are worth the effort.*
>
> — Damyan Ivanov, round one <20081215094128.GU8107@pc1.creditreform.bg>

An important consideration in the context of the Debian Project is the distinction between individualistic and collectivist cultures [*cf.* Rogers, 2003]. In the former, individual goals take precedence over group goals, and *vice versa* in the latter. It is difficult to say which one of those best applies to the Debian Project, but regardless, the distinction effectively turns the two-tier cost-benefit analysis into a four-tier consideration: individual *vs.* collectivist cost bearers on the one side, and the individuals *vs.* the entire project as beneficiaries on the other.

> *I can understand why the machine parseable format will be useful for some people, but not why it's likely to benefit me personally. Until there are tools that make it not significantly harder to use the new format than the old one, it's hard for me consider investing the time in it.*
>
> — Scott Kitterman, round two <200812070057.25291.scott@kitterman.com>

Cost-benefit calculations are estimates at best, influenced by subjectivity, peers, and inability to determine all costs and benefits up front. Instead, they are based on perceptions of the two factors, and may not be very considered [James Westby, round one[101]]. It is furthermore hard to quantify the impact of cost already incurred before reaching this point.

A good resource in making an informed cost-benefit estimate is tutorial-like documentation, which gives a glimpse of the tool or technology in use [Stefano Zacchiroli, round one[102]]. This is likely not enough for a considered calculation, but better than mere speculation. As problems can be expected, and if only because people may put tools to use in unexpected ways, documentation that details problems and how to work around them enable more accuracy in the estimation of the costs [Joey Hess, round two[103]].

[101] <1226589634.10403.29.camel@flash>
[102] <20081109185753.GA26709@usha.takhisis.invalid>
[103] <20081120001750.GA19544@kodama.kitenet.net>

Beyond cost-benefit Decisions are not always preceded by (explicit) cost-benefit considerations. It happens that cost-benefit estimates are ignored, or not made in the first place. A common cultural trait in the Debian Project (and FLOSS in general) is "yak shaving" — spending considerable amounts of time on tangential stuff instead of the actual task at hand (see section 2.2.4.3). Perceived benefits may include intangible benefits, including elegance (*cf.* section 7.2.2.2), gained experience, and fun.

> *Evaluating cost-benefit estimates doesn't fit with my view of programming enthusiasts. All of the ones I know, including myself, can spend hours writing a small program to automate a task which would have taken less time to be done directly. Still, automating the task was the "right thing" to do. We often care more about that than about the total time spent.*
>
> — Stefano Zacchiroli, round two <20081130181416.GA13001@usha.takhisis.invalid>

Just as the benefits may not be tangible or exaggerated, it seems as if the mere existence of costs can be a barrier to entry *per se*, causing DCs not to innovate:

> *It is just so much "easier" to let your hands repeat same tedious routines over and over. And some tools are better at allowing that: continue using same editor, same VCS or whatever - even if perhaps it would in principle be better to restructure the packaging workflow.*
>
> — Jonas Smedegaard, round one <20081110031747.GH30186@jones.dk>

A variant case is impatience before being able to reap benefits:

> *If I have to invest too much time in learning the tool or technique before I can really benefit from it, I'll probably be lazy and just look somewhere else, or ignore it.*
>
> — Christian Perrier, round one <20081107193431.GQ4145@mykerinos.kheops.frmug.org>

Finally, peer evaluations (*cf.* "peercolation", section 7.2.1.4) can offset the calculation. Costs can be lower if peers are available for assistance, and the perceived benefit could be heightened by messages from peers enforcing the worth of a tool [Enrico Zini, round one[104]]. Both factors could similarly have effects in the other direction, too.

Pragmatism With the exception of people who like to concentrate mostly on process improvement, at the end of the day there are tasks to be done in the context of one's involvement with the project. In such a case, it can be better to "accept a good-enough-but-not-perfect tool and get on with other things" [Colin Watson, round two[105]]. If the tool does its job, such pragmatism can be quite appropriate:

[104] <20081104173445.GA508@enricozini.org>
[105] <20081215131854.GX4735@riva.ucam.org>

> *People may use simple techniques that work well enough for them because more complex techniques aren't worth the trouble to learn (even if in the long run it would make their lives easier, it doesn't appear worth it to invest the time up front).*
>
> — Scott Kitterman, round two <200812070057.25291.scott@kitterman.com>

A concern with using whatever works is that it is grounded in short-term thinking [Guillem Jover, round two[106]], and the tool may become a burden for the maintainer, or others at a later point [Pierre Habouzit, round two[107]]. Costs and benefits can be assessed in the now and immediate future, or with a more long-term perspective.

Realigning outlying factions is subject of the uniformity influence discussed in section 7.2.9.1.

Active and passive rejection The outcome of stage 3, the individual decision stage, is a decision for or against an innovation. A decision in favour of the innovation leads into its implementation, which will be subject of the next stage (see section 7.2.4). A rejection, on the other hand, may be active or passive. An active rejection is an explicit decision against an innovation. A passive rejection is "never really considering the use of an innovation" [Rogers, 2003, p. 178].

Finally, I postulate the possibility of non-decision, which is effectively a lingering at this stage without deciding for or against an innovation.

> *Often we'll become marginally aware of something, and decide we don't have time to deal with it, and come back to it later. Then we can look at what bugs it's accumulated over time, and how well they were dealt with, whether people are still using it,* etc.*, and avoid having to be an early adopter.*
>
> — Joey Hess, round one <20081106024658.GA10322@kodama.kitenet.net>

This may simply be a function of lack of time, or rooted in the speculation that the costs might decrease with time:

> *During this period there will be feedback to the developers of the tool about what is needed that will help improve the tool, and reduce the total time cost to the project though, so it is a useful period.*
>
> — James Westby, round three <1240786091.3291.66.camel@flash>

[106] <20081130232243.GA13121@zulo.hadrons.org>
[107] <20081122152202.GA30285@artemis.corp>

7.2.4. Stage four: individual implementation

A positive decision in the third stage leads an individual to implement an innovation, which is the focus of the fourth stage, where s/he starts putting the innovation to use. Implementation does not suggest successful adoption, as the individual may give up during the implementation, or discontinue using the innovation soon thereafter.

While the adoption process thus far has been mostly a mental exercise[108], at this point, ideas and theories are carried into practice. This potentially unleashes a slew of problems and consequences that could not have been foreseen previously.

It is paramount to have support options to help with unexpected problems. Popular tools and techniques also benefit from widespread knowledge about the innovation. As such, the previously discussed documentation influence (see section 7.2.3.2) continues to play a role, but interactive support will become more important at this stage.

Re-invention Innovations are often re-invented in the process of their implementation, and modified by the adopter to better suit the local needs. While generally not regard favourably by researchers and vendors, due to distortion (diversification) of the adopters' group, and loss of identity of the innovation, re-invention (see section 3.1.2) is a common phenomenon in FLOSS; some might even argue that it is a pillar of FLOSS, fostered by the open environment.

An open environment gives adopters the freedom to re-invent. In fact, re-invention is very desirable among adopters, because it increases the possibility of improvement and better suitability to any adopters' use case [Rogers, 2003, p. 185]. As such, re-invention generally leads to increased adoption rates and sustainability of the innovation [*ibid.*, p. 183] (*cf.* section 7.2.2.4).

The enabling quality for re-inventions is flexibility, and flexibility is the common theme among the influences discussed at this stage (*cf.* the fourth reason for re-invention according to [Rogers, 2003, p. 186]).

Even though Rogers [2003, p. 186f] postulates the ability to distinguish oneself from the masses to be one of the reasons for re-invention, this is doubtful in the context of the Debian Project, where adopters are more likely to minimise divergence (see section 2.2.4.5).

[108] Rogers [2003, p. 179] claims that the first three stages are *strictly* mental, with which I disagree; trialability is an influence at stage 3 (stage 2 according to Rogers) and involves trying out a tool, which is hardly a theoretical exercise.

The implementation stage comes to an end when the innovation has become part of regular activities, and the original idea has lost its distinctiveness [*ibid.*, p. 180].

7.2.4.1. Influence 14: Modularity

> **Gist:** Different tasks and communities need different levels of abstraction.
>
> **Summary used during study:** Highly granular solutions (e.g. debhelper, autotools) require repeating the same sequence of instructions for standard tasks. Such code duplication is often regarded as bad, but it also makes it easier to understand what the steps are and what is happening overall. Monolithic solutions, on the other hand, are frequently inflexible and obscure. The ideal solution is somewhere in-between, requiring one to only track local modifications, and therefore minimising boilerplate, as well as divergence from the baseline.

The first influence at this stage is very tightly related to the next influence, transparency, subject of section 7.2.4.2. The distinction between the two is that modularity focuses on the right level of abstraction, whereas transparency concentrates on control and understanding.

The following discussion is almost exclusively about the two most widespread build helpers used in the Debian Project, debhelper and CDBS. The first is a uniform collection of numerous scripts in true Unix spirit (meaning each script has a well-defined task, and there is a consistent interface across them). CDBS, on the other hand, centralises most operation and exposes countless knobs to tweak aspects of the build process. CDBS uses debhelper internally. Thus, the discussion is really between using an abstraction layer like CDBS, or rather employing the constituents directly.

Pros and cons of abstraction There are good arguments for an abstraction layer like CDBS:

For a start, it reduces code duplication, which is considered bad in programming but found across the entire archive in the form of more or less identical sequences of calls to debhelper-components by each package using debhelper [Stefano Zacchiroli, round one[109]]. CDBS centralises these sequences and, as a consequence, they can be modified without having to update

[109] <20081109185753.GA26709@usha.takhisis.invalid>

every package explicitly using them [*cf.* Enrico Zini, round two[110]] and keep the archive closer to policy [Gunnar Wolf, round one[111]].

Furthermore, by exposing knobs to customise default behaviour, CDBS encourages maintainers only to specify the ways in which their packages deviate from the norm [Enrico Zini, round two[112]].

Other advantages of CDBS become clearer as the unit of modification shifts from the package to groups of packages. Panelists involved in teams responsible for large numbers of packages seemed in favour of higher levels of abstraction (without unanimously embracing CDBS), and one panellist particularly interested in customising Debian for a very specific use-case (embedded systems) appreciates its uniformity:

> *An important aspect of CDBS, for me, is that it always does every stage for every package. Even if the call does nothing, the routine is called. This makes it easy to adapt packages for different build objectives, making test builds with new binary packages, cross-building experiments and general "what-if" work.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

It may be a core problem of CDBS that it centralises more than repeated code. The following panellist, a strong opponent of CDBS, argued as follows:

> *By centralizing not only the individual operations that make up the build process, but also the* flow control *of the build process, within a semi-opaque (and definitely read-only) blob of make classes, CDBS makes it far more difficult to solve a significant class of bugs in individual packages* — i.e. *bugs that are ultimately bugs in CDBS itself, but that can't or won't be fixed quickly in the central tool. Using CDBS means you are ceding control of your package to the maintainer of that central tool, because routing around damage becomes substantially more difficult.*
>
> — Steve Langasek (his emphasis), round two <20081219022542.GC19907@dario.dodds.net>

The aforementioned supporter of CDBS thinks it is actually *more* customisable (and therefore not "read-only" as claimed above), whereas debhelper does not allow for enough influence under the hood:

[110] <20081205140625.GA22710@enricozini.org>
[111] <20081109033301.GB11775@cajita.gateway.2wire.net>
[112] <20081205140625.GA22710@enricozini.org>

> *CDBS exposes the internals so that corner-cases can be resolved without needing a new version of CDBS. Emdebian uses this support extensively and debhelper does not support it, even in debhelper7. In this respect, it is trivial to influence CDBS as-is and almost impossible to influence debhelper7 without a new upload of debhelper. Individual variables within CDBS can be tweaked to modify the build — doing the same with debhelper requires changing debhelper itself.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

The conflict between these two interests is unresolved and both ways of looking at the fundamental problem have their merits [Russ Allbery, round two[113]]. While debhelper and CDBS may be seen as two extremes of the spectrum, there is a middle ground between the two [Neil Williams, round one[114]], which requires more exploration.

Nevertheless, it is questionable whether a single solution suitable for everyone can be found, given the diversity in the Debian Project:

> *It's clear no one particular compromise of is acceptable to everybody, which is why we need (or at least end up with) different tools.*
>
> — Niko Tyni, round two <20081130213055.GA19290@rebekka>

The difficulty of refactoring While refactoring common functionality is generally desirable, it is not an easy feat, because abstraction interfaces have to be designed to be adequately expressive [Stefano Zacchiroli, round one[115]]. The following backs up the previous criticism of CDBS centralising flow control and order of the package creation process:

> *Sometimes it's better and clearer to explicitely have "repeated" code, even if code might seem to be mostly the same or closely related. But it might not be as uniform, and trying to refactor could result in a mess.*
>
> — Guillem Jover, round two <20081130232243.GA13121@zulo.hadrons.org>

A similar situation exists around the scripts that are run as part of a package's installation and deinstallation. Those are called maintainer scripts, and their purpose is the further integration of a software with the system beyond simply installing files into the various places of the operating system. Without doubt, there are many common patterns in those scripts, but no efforts have been successul to centralise those patterns beyond factoring some aspects into separate scripts in a way that does not come close to the regularity provided even by the highly modular debhelper suite. As a result, automated analysis of such scripts is impossible, and new Debian

[113] <87wsep9a04.fsf@windlord.stanford.edu>
[114] <20081218095834.187b4abe.codehelp@debian.org>
[115] <20081109185753.GA26709@usha.takhisis.invalid>

Policy provisions or bug fixes might require changes across the whole archive [Russ Allbery, round two[116]].

A related discussion took place after a participant drew parallels between debhelper and autoconf, which is a macro-based system allowing software publishers to tailor code for specific target environments:

> *autoconf expands macros inside shell scripts, while debhelper provides external scripts for common functionality, but the result is the same in both cases: a piece of shellscript where the common code is encapsulated by a macro or a script invocation, and in both cases you can add your own scripting inbetween macros or debhelper calls.*
>
> — Enrico Zini, round one <20081105103911.GA14309@enricozini.org>

While one participant responded noting that autoconf's goal is mostly flexibility, and it has to cater for different use cases (distribution and users) which makes it hard to define defaults [Stefano Zacchiroli, post-study follow-up to round two statement[117]], debhelper comes from the other end:

> *Autoconf is meant to be used by unrelated upstream authors, which wants maximum flexibility "only". debhelper is meant to be used by DDs, which do want maximum flexibility, but are also meant to work* together *to create a distribution of packages which should be built in similar ways. While debhelper is enough to guarantee flexibility, it is not enough to guarantee uniformity: that's where we need something more (debhelper7, CDBS, …).*
>
> — Stefano Zacchiroli (his emphasis), round two
> <20081130181416.GA13001@usha.takhisis.invalid>

The strive for the right level of abstraction involves a compromise between individual flexibility, and regularity across the project: project-wide operations should be possible, but without limiting the flexibilities and freedoms of the individual contributors [Jonas Smedegaard, round three[118]].

It seems that starting with flexibility and iteratively working towards regularity is the right way to do. At least, one participant identified this as the common trend between the aforementioned similar approaches, autoconf and debhelper, despite their alleged differences:

[116] <87wsep9a04.fsf@windlord.stanford.edu>
[117] <20090918081038.GA12434@usha.takhisis.invalid>
[118] <20090420013821.GI23632@jones.dk>

> *It's interesting too that the trend in autoconf has been to extend the expressiveness of the abstraction to cope with some more of the sorts of things people tend to do in embedded shell scripts, and the trend in debhelper is to extend it to provide boilerplate that expresses common idioms. It's clearly still seen as a virtue that you can do this sort of thing, but the authors appear to be trying to iteratively improve the balance with portability. I hope I'm not projecting my own opinions here, but I think this is the sort of approach many DDs approve of.*
>
> — Colin Watson, round two <20081201223929.GG4735@riva.ucam.org>

7.2.4.2. Influence 15: Transparency

> **Gist:** More abstraction leads to loss of control and makes a process more difficult to understand.
>
> **Summary used during study:** We are reluctant to pass up control. With functionality spread across abstraction layers, a tool is harder to understand and handle in the face of problems. Transparent design, a readable programming language, and resilience to bugs in underlying components all help to prevent loss of control. Tools automating a process have much higher transparency requirements than tools automating a single task, as there is more variance in the use cases.

Whereas the previous discussion on modularity focused on the level of abstraction, this section deals with control and comprehensibility.

Transparency in an innovation has been succinctly coined by a participant at the start of the Delphi study in the form of the following question:

> *Can you grasp what is going on and is it easy to influence it?*
>
> — Damyan Ivanov, round one <20081107203114.GV8178@pc1.creditreform.bg>

By the above metric, a typical Unix tool is transparent, because it focuses on precisely one atomic task, and strives to achieve it in the best possible way. I use the word "atomic" in this context not in the way it applies to computer instructions, but from the perspective of the users: even the simplest tools and techniques may involve multiple steps to achieve a task, but the task is so well-defined and sufficiently well-implemented that it is commonly not necessary to understand what is happening underneath, or to take influence. For instance, copying a file to another directory is such an atomic task, whereas building a Debian package is a lot more involved. The underlying question, which is core to this discussion has been raised during the study:

[Enrico Zini, round two[119]] When can a sequence of steps be turned into a simple one-shot action?

Under-the-hood knowledge Atomicity of a tool seems to depend on the type of task it performs, and how it relates to existing practice. In the following statement, helper tools are those that encompass multiple steps, possibly automating existing practice, whereas the other class of tools are those that perform tasks that might be perceived as more atomic in the eyes of their users:

> *The details of how `lintian` analyses a package don't matter in the same way that the details of the steps performed by git-buildpackage or debhelper matters. The details of how GnuPG signs a message don't matter, as long as it's secure and uses best crypto practices. Maybe the distinction here is tools that are "helpers", and tools that do significant work. But the lines seem to move over time.*
>
> *One set, the helper tools, try to automate just what was previously done by hand. The other set solve problems that there was no solution to before. Helper tools are easier to write, but have a harder path to adoption because everyone was solving the problem somewhat differently. There is a pull toward transparency and flexability when developing them. Tools that solve an entirely new problem are surely harder to develop, but are easier to decide to use, and can get away with being less flexible in some ways.*
>
> — Joey Hess, round two <20081120001750.GA19544@kodama.kitenet.net>

Transparency therefore directly relates to the level of abstraction, and the concept of "black boxes": a black box is a method that cannot (or will not) be dissected and further analysed, but its behaviour has to bei (or will be) taken for granted. The further away a tool or technique moves from atomic tasks, the greater the transparency requirement will be.

> *If a tool is transparent about what it's doing, even if this involves multiple steps, and those tasks can be duplicated manually, that helps a lot in picking it up. For example, one can perform all of the operations that git-buildpackage performs, back them out individually, and understand each component, which makes it easier to become comfortable with the individual steps and then let git-buildpackage automate them.*
>
> — Russ Allbery, round one <87ej1m7yye.fsf@windlord.stanford.edu>

Beyond understanding a tool or technique, knowledge of the constituents facilitates handling in the face of problems. The following statement suggests that DCs are (or should be) foresighted enough to the point to anticipate such problems and know how to deal with them:

[119] <20081205140625.GA22710@enricozini.org>

> *It is important for developers to know the details of what's going on to an extent greater than what's required to learn certain high-automation tools, because at some point they are likely to need to step outside the confines of those tools and it's important that they not then be completely out of their depth. We are still engineers, not merely assembly-line workers or consumers, and should have appropriate respect for the quality of our discipline.*
>
> — Colin Watson, round two <20081201223929.GG4735@riva.ucam.org>

Just as it is deemed necessary to be able to disseminate a tool's work and understand its steps, familiarity with the concepts of a tool, up until and including the programming language with which it has been implemented, facilitate the use of the tool in the wake of problems or new challengens [Neil Williams, round one[120]]. To this extent, interpreted languages are more accessible than compiled code [Niko Tyni, round two[121]].

Quality and motivation Tools capable of automating repetitive and error-prone tasks are powerful companions in the hands of those who know how to use them, but dangerous in the hands of those who use and do not understand them. A participant succinctly related the difference between those two groups of people to their motivation:

> *Automation must be motivated by the desire to do more, not to do less.*
>
> — Charles Plessy, round two <20081204143515.GA8588@kunpuu.plessy.org>

While many uses of sub-optimal tools can be found in the Debian Project for lack of better approaches (*cf.* "cost-benefit", section 7.2.3.5), it appears unacceptable and dangerous for DCs to use tools or techniques they do not fully understand [Pierre Habouzit, round one[122]].

> *I have seen people leap at tools that appeared to shortcut the need to learn things to do Debian packaging. While there are certainly aspects of packaging that are amenable to increased automation, there is an irreducable minimum of knowledge needed to produce reliable, policy compliant packages. People who don't care enough to learn such things are looking for shortcuts and sometimes getting solutions that at least initially appear to work.*
>
> — Scott Kitterman, round one <200811081804.30371.scott@kitterman.com>

Indeed, one of the panellists, who is also a contributor to CDBS, posits that CDBS should be considered an advanced tool:

[120] <1225790047.31771.202.camel@holly.codehelp>
[121] <20081130213055.GA19290@rebekka>
[122] <20081031224322.GE21799@artemis.corp>

> *CDBS is in my opinion* never *a good tool for a newbee, because it hides details you must to understand and respect. Only when you understand the underlying mechanisms does it make sense to use CDBS.*
>
> — Jonas Smedegaard (his emphasis), round two <20081201013212.GA9286@jones.dk>

Conversely, full understanding is too high a requirement, and it seems sensible to agree on a base-line:

> *Sure, it would be better if everyone understood every single bit of all actions which are performed while building their packages, but good abstractions have always been used to separate concerns among different actors, I fail to understand why that shouldn't be applicable to Debian.*
>
> *Do I need to understand every single detail of debhelper to use it? No, as long as it has a documented Application Programming Interface (API), which corresponds to what the various tools do.*
>
> — Stefano Zacchiroli, round two <20081130181416.GA13001@usha.takhisis.invalid>

debhelper is a widely accepted base-line and assumed to be common knowledge among DCs [Russ Allbery, round two[123]]. The reasons for that include accessible source code alongside comprehensive documentation, and strong adherence to the Unix principles [Russ Allbery, round three[124]] — it is transparent. It also enforces just the right amount of policy [Colin Watson, round one[125]].

Abstraction vs. control There is an underlying dialectic between abstraction and control [Enrico Zini, round one[126]]. Abstraction comes with a loss of control [Niko Tyni, round one[127]], and the benefit of factorisation decreases when the interfaces are not sufficiently expressive [Stefano Zacchiroli, round one[128]].

Whether the result of the aforementioned anticipation of problems, or the effect of a cultural trait of contributors to the project (*cf.* "elegance", section 7.2.2.2), DCs seek the ability to control every aspect of their work, even if the control may not be exercised.

[123] <87wsep9a04.fsf@windlord.stanford.edu>
[124] <87fxb7ua6k.fsf@windlord.stanford.edu>
[125] <20081109010317.GK4735@riva.ucam.org>
[126] <20081105103911.GA14309@enricozini.org>
[127] <20081107183929.GA5087@rebekka>
[128] <20081109185753.GA26709@usha.takhisis.invalid>

> *It is my impression that many DDs want to be able to control every aspect of their packages. Hiding repetitive, boring details is good, but people want to be able to change the default behaviour if needed. Good documentation with examples of typical situations makes a big differrence. Well-documented "hidden" parts can satisfy the need to know what exactly is happening.*
>
> — Damyan Ivanov, round three <20090419214402.GF18759@pc1.creditreform.bg>

This desire for control seems to be rooted in Debian's rigorous quality-oriented approach.

> *Debian has a mindset of attention to detail that I think is somewhat unusual in its tools. We expect people to think about a lot of things that can't be well-handled automatically in software in preference to doing simple stuff in software that sort of half-works.*
>
> *One argument* against *doing this sort of factoring is that it essentially adds another level of indirection, which can interfere with human understanding when one isn't familiar with the abstraction layer.*
>
> — Russ Allbery (his emphasis), round two <87wsep9a04.fsf@windlord.stanford.edu>

As such, abstraction layers work contrary to the desire disseminate a tool or technique in the case of problems, or to work around bugs.

> *A bug in a template-based build system can be fixed directly by editing the incorrect portion of debian/rules, but working around a CDBS bug requires a far greater understanding of make behavior.*
>
> — Steve Langasek, round two <20081130093607.GA21630@dario.dodds.net>

On the other hand, abstractions are useful in keeping the project closer to standards, because decisions are factored into centralised components [Gunnar Wolf, round one[129]]. This seems desirable, as already discussed in the modularity influence (see section 7.2.4.1), but such abstractions also make it harder to understand a tool or technique and use it in an acceptable way.

7.2.4.3. Influence 16: Genericity

> **Gist:** A tool or technique that can be put to use elsewhere will offset the cost-benefit calculation.
>
> **Summary used during study:** Being able to streamline work by reusing the same (or similar) tool/technique in different scenarios is an important feature. This sort of flexibility of a tool/technique can be significant enough to avoid an existing solution for a single task if it cannot be used elsewhere. On the other hand, too

[129] <20081109033301.GB11775@cajita.gateway.2wire.net>

> much flexibility can result in result in reduced usability. The usual Unix-principles — KISS, modularity, adaptability, and enforcing just the right amount of policy — are key to reusability.

Under the heading of genericity falls the preference for tools to be usable in different contexts. First, people put the same tool to use in different ways, and any "tool that mandates a single way of work and thought will not be accepted in Debian" on a broad basis [Enrico Zini, round three[130]]. Second, individuals "tend not to like to apply an overly wide spread of tools, and instead prefer to apply similar techniques to everything they work on" [Colin Watson, round one[131]].

If a tool or technique is generic enough to be reused in different contexts, then it will be able to harness network effects (see section 7.2.5.1) to spread more efficiently: people who learn about tools and techniques through team exposure will be able to put the same approaches to use elsewhere [Russ Allbery, round one[132]], and an increasing number of people who can put the same innovation to use increases its exposure by creating a buzz ("marketing", section 7.2.1.3), providing multiple perspectives ("sedimentation", section 7.2.1.1), and more examples (section 7.2.3.1) [Colin Watson, round one[133]].

Moreover, once adopted, generic solutions tend to be reused, increasing the benefit and hence influencing the cost-benefit analysis (see section 7.2.3.5):

> *People usually have their favourite packaging helpers, their favourite patch systems,* etc.*, and when creating a new package will often reach for the last vaguely similar one they did. Indeed, there is a level of inertia here in that changes also seem to need to demonstrate their general-purposeness.*
>
> — Colin Watson, round one <20081109010317.GK4735@riva.ucam.org>

The flipside of this inertia is that once a DC has settled for an approach and employed it in various packages, adopting an innovation incurs the additional cost of migration across all packages [Stefano Zacchiroli, round one[134]], impeding on future innovations. This can go as far as having negative effects:

[130] <20090421104533.GA5652@enricozini.org>
[131] <20081109010317.GK4735@riva.ucam.org>
[132] <87ej1m7yye.fsf@windlord.stanford.edu>
[133] <20081109010317.GK4735@riva.ucam.org>
[134] <20081109101430.GA11190@usha.takhisis.invalid>

> *I think the reusability of tools is a major factor in their adoption in Debian, often very much at the expense of elegance. svn-buildpackage is, in my experience, very inelegant and lacks in high-quality documentation, but because it can be applied (perhaps, "shoehorned") to any package maintained in Subversion, it seems to be in fairly widespread use regardless of one's workflow preferences. On the other end, I think* lack *of genericity may be a barrier to adoption for* `dpkg 3.0 (quilt)` *source package format, because it imposes constraints on quilt usage that don't exist with traditional dpkg source packages and that are incompatible with a number of packages currently in the archive.*
>
> — Steve Langasek (his emphasis), round three <20090427073700.GA5755@dario.dodds.net>

There is also a danger in too much flexibility:

> *After a certian point, the tool becomes so flexible that there stops being a real benefit to using it,* i.e., *if it takes a small shell script to run git-buildpackage with the right options, there's little benefit over the small shell script that just does what you want "by hand".*
>
> — Joey Hess, round two <20081120001750.GA19544@kodama.kitenet.net>

One reason for keeping the toolset small is that learning to use a tool, and staying on top of it, requires time that could be spent elsewhere:

> *One of the reasons why I resist being pushed to use bzr is that no other projects I work on use it. The flip side of it being divergent from Debian is that it is (from my experience) unique to Ubuntu and as a volunteer developer I'd rather spend my time doing productive work and not learning new tools.*
>
> — Scott Kitterman, round one <200811091300.53015.scott@kitterman.com>

An approach in true Unix philosophy — separate, simple tools doing atomic tasks with well-defined interfaces — is the key to reusing tools and techniques [Loïc Minier, round one[135]]. It is also the basis for re-invention [*cf.* Rogers, 2003, p. 186].

7.2.5. Stage five: organisational adaptation

In an organisational setting like the Debian Project, group-scale adoption starts after a considerable number of individuals have adopted an innovation and acted as vanguard. By that time, knowledge of the innovation has spread through the organisation, and in the absence of authoritarian or just central decision-making, no resource allocation or decision had to be made for the innovation to diffuse into practice.

[135] <20081111133148.GC18548@bee.dooz.org>

In the context of the information systems (IS) implementation model by Kwon and Zmud [1987], the individual adoptions happened within the organisational adoption stage — resource allocation in the case of the Debian Project could mean "individuals allocated themselves voluntarily". Moreover, the individual adoptions have chiefly taken place in the context of the organisation, and organisational constraints and requirements have influenced individual implementation, as well as re-invention.

The current organisational adaptation stage is thus born out of numerous individual implementations of an innovation. Therefore, all the influences discussed in section 7.2.4 continue to apply, but the focus at this stage shifts away from the individual and towards the group. However, instead of looking at the group as a singularity, it is more a collective entity, and the diversity of its members is of foremost concern.

At the adaptation stage, organisation and innovation begin to converge. On the one hand, members of the organisation implement, extend, re-invent, or combine innovations to work in the organisational context. On the other hand, the processes in the organisation change to accomodate the innovation. Neither of those two have to be explicit or noticeable, and it is generally only at a later stage (see section 7.2.8) that an innovation is incorporated.

The organisational adaptation stage is very similar in nature to the individual implementation stage, except that the adaptation (re-invention) is in the foreground due to the inertia of an organisation. However, since implementation in a group context suggests acceptance, adaptation is only part of implementation, and this stage is indeed followed by the acceptance stage in the stage model by Kwon and Zmud [1987].

The segue from adaptation to acceptance may be accompanied by critical mass (see section 3.1.2). However, critical mass needs not be reached for acceptance to begin.

7.2.5.1. Influence 17: Network effects

Gist: Working in teams restricts choices when others would be affected.

Summary used during study: Collaboration in teams restrains people in their choice of tool/technique due to the effects on others and the barrier of entry. Tools that are optional do not raise the barrier of entry or force people out; if the tools are visible nonetheless, and fill a niche, the adoption rate can benefit from the exposure. Teams can also develop a group tolerance to software faults over time: individual annoyances decrease in relevance as the group learns to live with or work around the faults. All these network effects seem stronger in teams working on diverse software, as opposed to teams centred around uniform software, e.g. written

> in the same programming language. All the same considerations apply towards downstream, with respect to other people reusing the work.

Network effects have become increasingly noticeable in the Debian Project in recent years [Colin Watson, round three[136]], as the project has shifted more towards team collaboration, to alleviate bottlenecks due to volunteer maintainers and increasing package count. Team maintenance increases the sustainability of the project as a whole [Jonas Smedegaard, round three[137]], but also influences the choice of tools and techniques by each individual, due to network externalities and excess inertia (see section 3.1.4).

Such network effects have a number of implications: first and foremost, they restrict choice, as it is crucial to make changes harmoniously with the collaborators [Charles Plessy, round one[138]]. As such, they slow down adoptions, but also help ensure that the quality of tools remains high. Also, collaboration can expose people to new ideas and give them a chance to try them. Lastly, teams can produce monocultures through highly specific tools, and the lack of exposure to other solutions.

Restricting choice The need to cooperate can restrict individuals in their choice of the tool, chiefly if such a choice has an effect on others [James Westby, round one[139]]: one is free to chose any text editor, but when it comes to build dependencies, such as a patch management system, the chosen tool has to be compatible with what others use.

This effect slows down the adoption of new tools within teams. Even if a new approach constitutes an improvement, if it impacts on the collaboration, then it can only be used to its full potential once everyone involved has adopted it. In smaller teams, and especially those driven by a few core contributors, mandating a tool may be possible. However, in doing so, one risks alienating other collaborators, who may be unable or unwilling to follow suit.

[136] <20090425005022.GG25892@riva.ucam.org>
[137] <20090420013821.GI23632@jones.dk>
[138] <20081108080522.GA28360@kunpuu.plessy.org>
[139] <1226268729.21767.458.camel@flash>

> *When you have to make decisions for a group, learning curves are important, and should be considered modulo what you know that the other members of the team already know. This can result in the choice of solutions which are not "the best" but which are more accessible to contributors, and/or the creation of additional documentation (*e.g. *tutorials specific to the group of people), in the hope of lowering the learning curve.*
>
> — Stefano Zacchiroli, round one <20081109101430.GA11190@usha.takhisis.invalid>

Compatibility with existing practice (*cf.* section 7.2.3.4) is another way in the adoption rate of tools in teams can be increased:

> *If a new tool "scales down" somehow, allowing people to contribute without using it, then it never completely cuts people away. For example, you can always feed patches to a VCS, so even if people don't know the VCS you use, they can still send you patches.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

In a collaborative environment, popularity of a tool plays a role separate to considerations of sustainability or return-on-investment (see sections 7.2.2.4 and 7.2.7.1). In fact, the ability to collaborate seems to be the main network externality in learning a new tool in the Debian Project:

> *Far from being a simple case of mindlessly following the herd, there are a number of reasons why the popularity of a tool within Debian matters, such as the likelihood you'll be able to collaborate with other developers on a package without one of you having to first learn a new tool.*
>
> — Steve Langasek, round one <20081109132536.GA15447@dario.dodds.net>

Conservative adoption and quality Aside from slowed progress and suboptimal solutions, the excess inertia of teams has positive effects on the quality of tools. First, the delay between exposure and adoption (the persuasion of each individual, see section 7.2.2) helps to weed out bad tools. Consequentially, when a tool gets adopted by a team, that is a good indication of its quality — unless of course a given tool is the only one to address a need. The individual influences described in stages 1–4 (sections 7.2.1 through 7.2.2) continue to be relevant at this stage, but they are multiplied by team sizes.

Furthermore, in a FLOSS context, tools and techniques are prone to evolve and be improved by their users.

> *People working on the same packages bring their own experiences, which can result in better solutions. This is what I am experiencing in the FreeSmartphone.Org team.*
>
> — Luca Capello, round three <87hc0ks9sd.fsf@gismo.pca.it>

At the same time, collaboration considerations can help authors target their tools for team adoption:

> *It encourages authors to think about a variety of people in both their design and their documentation (for example, large software development teams need to comprise programmers, documenters, and translators, who tend to have widely varying skill levels with various tools). It reinforces peer-to-peer rather than top-down development (instead of getting some authority figure to mandate some practice, you're better off convincing individual developers who do sufficient work in a variety of places to be influential, and thus benefit from their wide experience).*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

The slowness of adoption by a team can be explained in various ways. One participant described teams as conservative in that they avoid creating too high of barriers of entry [Russ Allbery, round one[140]], while another participant preferred to simply reduce it to inertia:

> *I think the reason for team conservativism is more that changing a tool or technique in a team would force the change on all members: the larger the team, the more the inertia.*
>
> — Stefano Zacchiroli, round two <20081130181416.GA13001@usha.takhisis.invalid>

Collaboration and exposure Just as tools that are visible and whose use has an impact on others can restrict potential adopters in their choice, those tools may well spread faster as a result of the exposure in and across teams [Colin Watson, round two[141]].

The inertia of teams can also push adoption rates if the tools are adequate, address needs, and the team helps members with the adoption, *e.g.* through additional documentation [Stefano Zacchiroli, round one[142]], or by supporting its contributors throughout the implementation (*cf.* change agents in section 3.1.3).

Teams with well-defined rules and processes may be easier to join, as contributors are able to figure out what to learn to help out [Luca Capello, round three[143]], and potential adopters may be

[140] <87ej1m7yye.fsf@windlord.stanford.edu>
[141] <20081201223929.GG4735@riva.ucam.org>
[142] <20081109101430.GA11190@usha.takhisis.invalid>
[143] <87hc0ks9sd.fsf@gismo.pca.it>

able to try out new tools and techniques, without having to understand them all the way to put them into practice from scratch (*cf.* "trialability", section 7.2.3.3).

Team monoculture Inertia in the choice for tools in a team can also produce monocultures with homophilous members (*cf.* appendix B.1.2), which seems more pronounced in teams formed around technical commonalities (the Perl team was frequently cited as example of such a team), than in teams formed around concepts (*e.g.* the VoIP team, or debian-edu) [Jonas Smedegaard, round two[144]]. This can inhibit the adoption of tools and also prevent the spread of tools beyond team boundaries:

> *It can also lead to tools becoming widely used in a team, while still unheard of or little used outside of it. The secure-testing team has some tools and file formats that implement workflows that are too special-purpose for outsiders. The pkg-perl team has some tools and web pages that could be useful for other teams, but have only achieved wide use inside that team. The Debian installer team has tools such as `output-l10n-changes` that aren't really team-specific but are only used in it.*
>
> — Joey Hess, round three <20090601170437.GA26955@gnu.kitenet.net>

This statement suggests that cross-team communication decreases with team specialisation, a claim backed up by Katz and Allen [1982], who have shown a relation between long-term team stability (tenure) and decreasing performance:

> Project teams become increasingly cohesive over time and begin to separate themselves from external sources of technical information and influence by communicating less frequently with professional colleagues outside their teams. [...] If project communication with internal and external colleagues diminishes significantly as mean group tenure increases, and if such communications are essential to technical performance, then one should also expect a strong inverse relationship between group tenure and project performance.

One symptom of this tenure and team cohesion may be the "Not-Invented Here" (NIH) syndrome, which has been identified in teams, as well as the Debian Project as a whole:

[144] <20081201013212.GA9286@jones.dk>

> *The Debian Project uses a variety of tools or formats that are either specific or not popular elsewhere, and a lot of resistance is met when trying to move towards more compatibility with the outside world.*
>
> — Charles Plessy, round one <20081108080522.GA28360@kunpuu.plessy.org>

By extension, teams and the entire project may be oblivious to the work of others. In addition to suboptimal communication and possibly decreasing performance, collaboration across team border is often not considered. Teams need to be aware of those making use of one's work and not impose tools on them [Neil Williams, round two[145]], and also pay attention to downstream users who want to get involved [Scott Kitterman, round two[146]]

7.2.5.2. Influence 18: Scaled use

> **Gist:** Tools or techniques that can be gradually put to use can make adoption automatic.
>
> **Summary used during study:** Tools or techniques that offer various degrees of use are good; it should be painless to employ a tool's basic features, and then allow for progressively advanced use. Tools/techniques that implement a standard are easier to adopt if they are flexible enough to be used in non-standard ways, and support the convergence with the standard at a later point.

The idea of scaled use is that one can put a tool or technique to use with ease for basic tasks [Pierre Habouzit, round one[147]], and to continue using the same approach as the complexity in usage scenarios increases [Christian Perrier, round one[148]]. Similarly, tools that can "adapt to your current process but leave open the option of evolving that process to something more 'standard' are easier to adopt than tools that enforce their method of doing things" [Russ Allbery, round one[149]]. The devscripts collection is "particularly good at putting together tools like that automate existing practice, and then slowly adding more optional features so that people can evolve their work process into a more and more automated process" [*ibid.*].

A variant case was selected in the third round by a participant together with trialability (see section 7.2.3.3):

[145] <20081121184632.106f2666.codehelp@debian.org>
[146] <200812070057.25291.scott@kitterman.com>
[147] <20081031224322.GE21799@artemis.corp>
[148] <20081107193431.GQ4145@mykerinos.kheops.frmug.org>
[149] <87ej1m7yye.fsf@windlord.stanford.edu>

> *In Debian, maintainers have the freedom to try out new techniques on one package (or perhaps, set of related packages) at a time. In VCSs, this has been particularly true; I started using Subversion for packages through the installer team well before considering it for my other packages (many of which were still in CVS), and to try out bzr I picked a couple of packages that weren't in any VCS at all at that point. I've also taken advantage of bzr's interoperability with Subversion to introduce it into my workflow in some contexts (*e.g.*, the Samba packaging team) where it would not be feasible to expect the whole team to switch.*
>
> *This applies to project-wide patterns for the same reason it applies to my own experience. Changes that individual maintainers can adopt on a per-package basis gain traction; those that require top-down enforcement generally do not (unless there's consensus that packages which* don't *follow the new behavior are innately buggy).*
>
> *Adoptions of debhelper, CDBS, dpatch/quilt/dbs, and the various *-buildpackage tools have been heavily influenced by this.*
>
> — Steve Langasek (his emphasis), round three <20090427073700.GA5755@dario.dodds.net>

7.2.5.3. Influence 19: Maturity

> **Gist:** Tools or techniques must provide a reliable base before people will depend on them.
>
> **Summary used during study:** A tool/technique must be usable to sustain followers. It should not change continuously and require users to re-learn or change their scripts. It should also be free from serious bugs that would make its use too much of a pain, especially if alternative tools/techniques do not exhibit such bugs.

Organisational adoption is the sum of individual adoptions by members of the organisation. For an organisation to adopt an innovation, the innovation needs to become stable and mature. Mature tools are building blocks in the organisational adoption process, and tools that are not yet stable need to solidify to be accepted, which brings about gradual (non-revolutionary) change [Jonas Smedegaard, round three[150]].

Maturity is more relevant for tools and techniques that have a broader reach within the project, than for trivial ones. For changes like the Debian source package format "you need to consider every single aspect" [Luca Capello, round two[151]].

The most important message of this influence is that project-wide adoptions take time, and there is fairly little that can be done about it. As part of organisational adaptation, when organisation

[150] <20090420013821.GI23632@jones.dk>
[151] <87k5aksbpl.fsf@gismo.pca.it>

and innovation converge, tools that are not yet stable will have to mature to gain majority adoption:

> *Implementation maturity is important*, i.e. *stable interfaces and relative freedom from serious bugs over time. Developers invest effort into learning new tools because it pays off in improved productivity; if the tools instead cost the developer time tracking changes, they're liable to be shed in favor of more reliable tools.*
>
> — Steve Langasek, round one <20081109132536.GA15447@dario.dodds.net>

But this is not to suggest developers to assume a waterfall-model and postpone the expectation for adoptions until the software is "finished". It may be better to start small, iteratively improve design and interfaces [*cf.* Buxton and Sniderman, 1980] and give the community a chance to come on board:

> *Release early, as long as the software gets one simple, yet meaningful job done: that will give people a reason to start using it, and hopefully contribute to its growth.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

It seems that the focus should be on automating a task, rather than to concentrate on design, because it is likely that good ideas get reimplemented, if their first incarnations were suboptimal [Joey Hess, round two[152]]. While the Unix principles (*e.g.* "modularity", section 7.2.4.1) appear a good baseline, it is not always apparent how to split up a task without gaining experience and letting others contribute.

> *Many large-scale changes in Debian have to be implemented in this way — create small packages full of dirty hacks that allow others to test the results and the process with their own packages. Gradually, over a period that can often be several years, the hacks mature and the packages migrate into the core tools, leading to a permanent change.*
>
> — Neil William, round two <20081121184632.106f2666.codehelp>

Tolerance Tools that seek to replace existing approaches will be scrutinised and compared to the current ones, and any problems will become barriers to adoption, as people can just stick with the *status quo* [Neil Williams, round one[153]].

On the other hand, new approaches that fill niches, or solve problems that were hitherto not addressed or unknown instill lower expectations and requirements [Guillem Jover, round one[154]]. The tolerance to new problems decreases with the age of the tool, but project members also seem

[152] <20081120001750.GA19544@kodama.kitenet.net>
[153] <1225790047.31771.202.camel@holly.codehelp>
[154] <1225790047.31771.202.camel@holly.codehelp>

to build up a group tolerance to existing problems over time, and they "become well known, get documented, new tools and techniques spring up to paper over them, *etc.*" [Joey Hess, round two[155]].

After several unsuccessful attempts have been made to address a problem, the expectations seem to grow, possibly due to the experience gained from the previously inadequate solutions:

> *However, when it is a long known problem we can often reject anything that isn't seen as a complete solution. We all understand the problem and can spot where a solution falls short. In these cases we can often be opposed to incremental improvements, especially when those improvements come without a clear plan to solve the rest of the problems.*
>
> — James Westby, round two <1228587563.4490.29.camel@flash>

Availability The availability of a tool in, or support for a technique by the stable Debian distribution (see appendix A.1.4) is important for people who may be working in constrained environments [Russ Allbery, round one[156]], and those supporting packages in multiple distributions:

> *We maintain software in multiple distributions (stable, testing, unstable) and when the new tool requires a build-dependency of some sort, it will hinder its adoption if that build-dependency is not satisfiable in all supported distributions. I have heard this argument multiple times within teams where some members did increase the debhelper compat level needlessly. I also heard people wary of switching to "3.0 (quilt)" source formats because the previous stable release (oldstable) doesn't support it.*
>
> — Raphaël Hertzog, round three <20090413102934.GC23906@rivendell>

Stability In the context of the proposed change to the Debian copyright file format, one participant wondered why the idea has not been adopted yet, even though (he claims) most people see the advantages therein:

[155] <20081120001750.GA19544@kodama.kitenet.net>
[156] <87ej1m7yye.fsf@windlord.stanford.edu>

> *I think the answer is the perceived instability of the proposal: the Wiki page changes continuously, and it is not mentioned in "authoritative" packaging documents (*e.g. *the Developer's Reference).*
>
> — Stefano Zacchiroli, round one <20081109101430.GA11190@usha.takhisis.invalid>

As noted at the start of his section, stability is fundamental for organisational adoption. The same participant speculated that its stability contributes to the authority of the Debian Policy document [Stefano Zacchiroli, round two[157]].

> *Detailed documentation requires/promises a stable ABI - if promises are kept and tools adopted (thus puts pressure on dependent tools to keep their Application Binary Interface (ABI)), defacto standards have formed, and can be written down as official policy or "best practice".*
>
> — Jonas Smedegaard, round three <20090420013821.GI23632@jones.dk>

7.2.6. Stage six: organisational acceptance

The stage of organisational acceptance follows organisational adaptation, and may coincide with the occurrence of critical mass. I will present only one influence in the context of this stage, which deals with consensus as the barrier to incorporation of an innovation, and standardisation of its use.

Consensus and critical mass are related in that critical mass can be seen as a form of implicit consensus in a voluntary setting. Conversely, consensus can be driven to enable incorporation without critical mass.

By the time consensus has been reached, whether implicit or explicit, the innovation has been accepted and the next stage is reached.

7.2.6.1. Influence 20: Consensus

> **Gist:** Forming consensus is a necessary process, which benefits greatly from the right amount of driving.
>
> **Summary used during study:** Pioneering work is necessary, but concrete solutions need to follow from that. It is important to build consensus with experienced people, and proponents from all involved sides. Too much discussion, however, can cause loss of focus and hinder change.

[157] <20081130181416.GA13001@usha.takhisis.invalid>

The Debian Project is a project run by volunteers, and the voluntary involvement is an important motivation for many of its members (see section 2.2.3). In addition, the flat, meritocratic hierarchy of the project has a number of important effects on the cooperation between members.

First, DCs cannot be forced to use any particular tool or technique, but when a tool or technique does become mandated through standardisation (including *de facto* standardisation, see section 7.2.8.1), then those who use incompatible, non-standard ways, will effectively be forced to shoulder the cost of a migration anyway, or else their contributions may no longer be accepted [*cf.* Pierre Habouzit, round one[158]]. This can be a source of resistance (see section 7.2.2.3), but also a strong motivation for those people to engage in the preceding discussion in an attempt to block the change [*cf.* Charles Plessy, round three[159]].

Second, there seems to be an expectation for a minimum level of discussion and consensus in the project [Christian Perrier, round three[160]], discussion is a necessary component of change, and the amount and depth of discussion has been considered "possibly what most defines Debian" [Gunnar Wolf, round three[161]]. If a decision is made without giving everyone the chance to participate in the lead-up discussion, the decision is considered "cabalistic" and its authority is questioned (*cf.* section 7.2.8.1).

Third, everyone working on Debian has the ability to voice an opinion (unless a discussion happens non-publicly), and there are various reasons why people will. First and foremost, it is trivial to voice an opinion, and given the diversity in the project, a broad spread of opinions can be expected. While non-controversial, project-wide changes readily get implemented without much discussion [Guillem Jover, round three[162]], the amount of discussion increases for topics that are more general with respect to the project (*cf.* the bikeshed principle, section 7.2.1.4), or which have broader implications [*cf.* Luca Capello, round two[163]]. Second, by voicing opinions, members can influence a tool or technique (*cf.* "sustainability", section 7.2.2.4), which can convey the feeling to be part of the innovation.

Need for consensus The need for consensus seems to be proportional to the amount of (anticipated) controversy over an innovation, which is somehow paradoxical when put in the context of the amount of discussion: the more controversial a decision, the more people

[158] <20081031224322.GE21799@artemis.corp>
[159] <20090712142503.GI29671@kunpuu.plessy.org>
[160] <20090416202847.GD9510@mykerinos.kheops.frmug.org>
[161] <20090420040612.GA6715@cajita.gateway.2wire.net>
[162] <20090422045458.GA14006@gluon.hadrons.org>
[163] <87k5aksbpl.fsf@gismo.pca.it>

will engage in the discussion, and the search for consensus will be harder. Justified changes and clear improvements, on the other hand, tend to propagate steadily [Guillem Jover, round three[164]].

> *The lack of clear top-down direction in many places in the Debian Project (while often a plus) means that the search for consensus is often interminable, and too easily reopened by a small number of dissidents.*
>
> *The key is to try to ensure that consensus is achieved on matters where it is important, but not to fetishise it when it isn't; a maintainer making a decision by fiat on the spelling of a field name is not going to disenfranchise anyone in any meaningful way.*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

Consensus-finding happens on multiple levels in the project, starting from small teams to the entire project at large, and therefore spanning different group sizes. The larger a group, the more difficult the process becomes, because of diversity in opinions, but also because of diversity in the problems that are being addressed:

> *I think that reaching some kind of overall consensus about any big packaging effort is nowadays difficult: both because so many maintainers means that someone will always raise her/his hand to complain (and she/he will find supporters) and also because the software we distribute is so different from one to another that it is impossible to conceive something more general than the current situation.*
>
> *Teams are now a key-value in Debian: they (can) promote changes because the userbase is smaller (less conflicts)*
>
> — Luca Capello, round three <87hc0ks9sd.fsf@gismo.pca.it>

In teams, it is also more likely to find leaders that are in a position and willing to make decisions to move forward.

> *In smaller teams I see decisions getting made via discussion, identification of consensus be peer leaders, and then the team forming around that consensus and moving forward. An example of this is the recent decision in the Python Modules Team to standardize on `pysupport` and drop `pycentral` in the team.*

[164] <20090422045458.GA14006@gluon.hadrons.org>

> *On a larger scale, across the project as a whole, this seems very difficult to do. It seems to me like most issues devolve into mailing list flamewars and an unwillingness of people on the losing side of consensus to recognize it and accept the collective decision over their own preference. This is coupled with a lack of mechanisms that are widely viewed as legitimate to go to an official group that resolves disputes (In theory I guess the technical committee has this role for some things, but it doesn't appear to me to be much used).*
>
> — Scott Kitterman, round three <200905232049.11003.scott@kitterman.com>

This strikes a core conflict in the Debian Project: on the one side is the volunteer collective with a strong belief in the freedom of choice[165] (*cf.* "resistance", section 7.2.2.3), and on the other side one finds the need for decisions to enable progress. While choice may sometimes be imaginary because of very strict rules in the project [Pierre Habouzit, round one[166]], the project as a whole is lenient in deprecating approaches [Joey Hess, round two[167]]. Lack of decisions yields loss of focus. This was ostensively the case in the discussions about the homepage control field, where many solutions were proposed but no settlement occurred until much later, causing different approaches to propagate [*ibid.*].

Similarly, the definition of guidelines for packaging patch maintenance could potentially have benefitted from an earlier initiative:

> *Things would probably be generally better if we had had standardized patch handling from the start. Luckily this is finally getting better with the new dpkg source format.*
>
> — Niko Tyni, round three <20090420113239.GA25346@kuusama.it.helsinki.fi>

It helps when people take responsibility, finalise proposals [Guillem Jover, round three[168]], and enable progress:

> *I would prefer more decisions pushed from the top, simply because these decisions oblige people to move on. These kind of decisions are not bad at all: if they are well coordinated and announced, they can save both time (usually spent in discussing/flaming) and acceptance (since the decision is "imposed").*
>
> — Luca Capello, round two <87k5aksbpl.fsf@gismo.pca.it>

Executive decisions include the maintenance of proposals, so that they can stabilise, rather than subjected to polishing rounds of diminishing benefit [Colin Watson, round two[169]]. Pioneering

[165] *cf.* a post to the `fedora-devel` mailing list on the problems of this belief: `https://www.redhat.com/archives/fedora-devel-list/2008-January/msg00861.html` [8 Sep 2009]
[166] <20081031224322.GE21799@artemis.corp>
[167] <20081120001750.GA19544@kodama.kitenet.net>
[168] <20090422045458.GA14006@gluon.hadrons.org>
[169] <20081201223929.GG4735@riva.ucam.org>

ideas is important, but at some point, ideas have to be finalised, and the success of a tool depends not in the least on the organisational talent of its key supporters [Andreas Tille, round two[170]].

> *The machine-readable copyright file proposal is an excellent recent example of over-reliance on consensus. The effect, at least when I last looked, was a generally fine proposal but one that showed very noticeable signs of design-by-committee, with quite a bit of overengineering in various areas; the constant back-and-forth about tiny details also made the lack of a maintainer very clear.*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

Building consensus Discussions are sought for constructive criticism [Gunnar Wolf, round three[171]]. In a project which prides itself on its openness like Debian (*cf.* appendix A.6), non-public discussions are somewhat frowned upon. Nevertheless, they prove important to prepare proposals with rigour proportional to the size of the affected group, to anticipate disagreement, and to prevent the initial rebuttal from stealing the thunder:

> *This initial rebuttal can be managed and prevented by careful preparation (of people not code). If a tool potentially introduces a step that is obviously going to be controversial and is likely to get the whole idea rejected out of hand, the solution for the problem needs to be discussed in smaller groups, out of reach of the main public lists. It is essential that the proponents gather a mix of people, some of whom are able to see both sides of the problem, so that a compromise becomes not just possible but desirable. The problem needs to be raised alongside the benefits of the proposal and care is needed to ensure that the proponents are willing to modify the proposal in whatever ways are necessary to achieve the objective and solve the known problem.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

Anticipating backlash is part of the larger, necessary endeavour of building consensus, and thus a positive effect [Colin Watson, round one[172]]. Another participant put it like this:

[170] <alpine.DEB.2.00.0812050821200.9613@wr-linux02>
[171] <20090420040612.GA6715@cajita.gateway.2wire.net>
[172] <20081109010317.GK4735@riva.ucam.org>

> *Trying to reach a consensus is an open process that requires the driver to clearly articulate the new tool and its goals. It also means incorporating ideas/improvements proposed by peers. Given the nature of the Debian community, those improvements will lead to all the important points that you have listed like "genericity", "compatibility", "modularity" and "chunking".*
>
> — Raphaël Hertzog, round three <20090413102934.GC23906@rivendell>

Consensus is a considerable driver in the Debian Project, and it seems to originate mainly from developers with distro-wide interests, while the developer base at large learns of such changes by way of standards documents [Guillem Jover, round three[173]], but fundamental consensus is necessary to maintain the authority of those documents (see section 7.2.8.1).

7.2.7. Stage seven: use-performance-satisfaction, individual confirmation

When an innovation is being accepted by an organisation, benefits of the innovation for each adopter appear at the group level, another instance of network externalities (see section 3.1.4). Such benefits include increased support options (people to ask, documentation, examples) due to increased popularity and a wider spread of knowledge, an "after-market" of related products that enhance the original innovation, and personal and group efficiency gain.

The seventh stage is a fusion of the organisational conglomerate "use-performance-satisfaction" from the IS implementation model [Kwon and Zmud, 1987], and the confirmation stage in Rogers' individual innovation-decision process [ch. 5], because they are essentially about the same: (positive) post-adoption consequences at the individual level.

Even though the literature is not helpful in defining use-performance-satisfaction beyond calling them "assessments", the choice of words themselves are sufficiently expressive for the discussion at hand. In addition, together with acceptance, they are considered to play a "critical role within the [IS implementation] model as they determine whether an innovation was appropriate or inappropriate" [Kwon and Zmud, 1987]. As such, they relate directly to the concept of confirmation at the end of an individual adoption process, when an individual "seeks reinforcement for the innovation-decision already made" [Rogers, 2003, p. 189].

These benefits need not happen only after the previous stage, organisational acceptance (section 7.2.6) has been completed by the organisation. Instead, individuals and sub-groups may experience benefits long before the organisation has accepted an innovation. And even though

[173] <20090422045458.GA14006@gluon.hadrons.org>

this stage is not necessarily reached by each individual [*ibid.*], it is an important prerequisite for later incorporation. An individual may be able to reduce any post-adoption dissonance about an innovation [*ibid.*], but dissonance prevailing across a social system can act as a barrier to incorporation by the organisation, lead to individual discontinuance, and potentially yield disenchantment across the members of the organisation, effectively reverting the organisational adoption of a previously accepted innovation.

The influence I present at this stage focuses on the individual returns on previous investment of time and energy. As suggested above, some of the benefits are more pronounced at the group level (such as increased support options), but the benefits are mainly at the individual level. There is no question that *e.g.* a gain in group efficiency is also (or even primarily) an organisational benefit. However, for the purpose of this discussion, I would like to postpone treatment of organisational benefit to the later discussion of uniformity as an organisational initiator in section 7.2.9.

7.2.7.1. Influence 21: Return-on-investment

Gist: With increasing adoption, the benefits of a tool or technique tend to grow.

Summary used during study: Adopting a tool/technique incurs costs. We are conscious about these costs and seek to profit from the adoption decision. Active maintenance, other users, better documentation and support options, an aftermarket, evolution of features, and adaptation to changes in the environment are some of the returns on investment we require. For the final adoption decision, we also seek the comfort of the "multiple eyes and shallow bugs" principle.

What sustainability (see section 7.2.2.4) was pre-adoption, return-on-investment is post-adoption: an adopter has expended costs in implementing an innovation, because the same argument and aspects that led him/her to consider the innovation sustainable have now turned into benefits and can be reaped. Similarly, the cost-benefit considerations discussed in section 7.2.3.5 play into the equation at this stage. Furthermore, any re-invention that may have occurred at stages 4 and 5 (individual implementation and organisational adaptation in sections 7.2.4 and 7.2.5) have added to the effect, the adopter can profit from the returns.

As stated in the stage's introduction above, returns on investment need not wait until the organisation has accepted an innovation (*cf.* organisational acceptance in section 7.2.6): individuals may already profit from streamlined processes and sensible automation, but once

consensus or critical mass have been reached, benefits emerge out of the popularity of a tool

Perhaps the most important such benefit is the increased availability of support:

> *If a system is not optimal, but I can go on IRC, ask a question and get a meaningful answer in a matter of seconds, then most of its shortcomings can be worked around with little effort.*
>
> — Enrico Zini, round one <20081104173445.GA508@enricozini.org>

In addition to such live support, popularity also brings better documentation [Niko Tyni, round one[174]], and the "aftermarket" grows [Raphaël Hertzog, round one[175]].

7.2.8. Stage eight: incorporation

Incorporation occurs when "the innovation becomes embedded within an organization's routine and when the innovation is being applied to its full potential within an organization" [Kwon and Zmud, 1987].

In the context of the Debian Project, explicit routinisation happens through the definition of a policy item or the specification of best practice in a seminal standards document. Contrariwise, *de facto* standards, albeit unformalised, are often enough to be considered routine without explicit definition. These are topic of the upcoming section 7.2.8.1.

In addition to the individual benefits presented in the previous section (stage 7, section 7.2.7), organisational benefits come from standardisation, which enables uniformity (see section 7.2.9.1). At the level of complexity of the Debian Project, it is necessary to ensure standards-compliance, and implement other quality assurance mechanisms. Those will be discussed in section 7.2.8.2.

At the end of this stage, the innovation has been routinised and has lost its character as an innovation, meaning that it is no longer perceived as novel.

7.2.8.1. Influence 22: Standards

Gist: Standards evolve from practice. Defining them is a balance between imposition and choice.

[174] <20081107183929.GA5087@rebekka>
[175] <20081107094111.GA25888@ouaza.com>

> **Summary used during study:** Standards should define end results, not processes or tools. They are a good way to drive evaluation, though not necessarily adoption. Standards evolve or result from long-term network effects, they are usually not simply defined. Yet, by defining standards, one can enable progress. It is necessary to communicate standards and changes to everyone. Using an already wide-spread tool can be an effective vehicle, but care must be taken not to introduce problems.

I pick up the discussion of this influence where the discussion of the consensus influence left off (see section 7.2.6.1); the previous stage in-between was special in that it is not a prerequisite for later stages, but more a result of previous stages (see section 7.2.7). As part of the discussion on consensus, the participants highlighted both, the need for consensus to drive progress, and the need to build consensus before standardisation can occur.

While section 7.2.6.1 focused on building consensus, I also wrote about the need to finalise a solution (*cf.* the homepage control field example). Arguably, this is part of incorporation and the distinction is indeed fuzzy. In the current section, I concentrate on the role and effects of standards.

In the Debian Project, a number of sources of standards exist, both in explicit form, as well as implicitly defined. The most important explicit sources are the Debian Policy, as well as the Developer's Reference[176]. The former is a binding document describing rules by which packages must abide to be included in the Debian System. The latter is a collection of best practices for developers, covering responsibilities and best practices.

In addition to these documents, several unwritten rules, or best practices also exist in the project, *e.g.* avoiding certain outdated tools (yada, dbs), or using a VCS to maintain package source code. Yet another source of best-practice suggestions are the base tools used for packaging, such as the tools included in the `dpkg-dev` package, as well as tools included in the `devscripts` collection.

Many of these best practices have not been formalised, either because consensus has not been reached, or no one has taken up the initiative, which often needs quality-assurance efforts [*cf.* Raphaël Hertzog, round one[177]]. Nonetheless, consensus — explicit or by critical mass — is necessary for all but the non-controversial changes.

[176] `http://www.debian.org/doc/developers-reference/`
[177] `<20081107114515.GC25888@ouaza.com>`

> *Debian has a strong culture around the idea "we're all volunteers and so no one can tell another volunteer how they should do something". As a result, policy can only describe current practice, not lead it. So the process for new tools and processes is inherently* ad hoc *and chaotic. This is both a bug and a feature.*
>
> — Scott Kitterman, round two <13222-SnapperMsgD8DB99B6C56F29A6@[75.196.134.29]>

Standardisation is thus an evolutionary process [Andreas Tille, round two[178]], which is not always explicit [Raphaël Hertzog, round one[179]]. This seems generally beneficial, because of the possibility of re-invention and adaptation (see sections 7.2.4 and 7.2.5):

> *The proliferation of *-buildpackage tools has led to some interest ideas being developed. I'm not sure if this would have happened if there had been a single vcs-buildpackage from the start.*
>
> — James Westby, round two <1228587563.4490.29.camel@flash>

Authority For standards to enable progress instead of limiting it, they need to follow an innovation, not lead it, and not be defined too strictly [Loïc Minier, round three[180]]. In fact, the Debian Policy, as the most authoritative standards document, is not an authoritarian document:

> *Dictating through the policy is a very good way to make people mad at it. So I would go lightly with that.*
>
> — Pierre Habouzit, round two <20081122152202.GA30285@artemis.corp>

Much of the authority of standards stems from the process in which they have been defined:

> *As a newcomer to the policy team, I feel rather strongly the responsibility to help refine standards that meet the project's collective notion of elegance. If experienced developers feel that many additions to the policy manual run contrary to their instincts, whether those are rational or not, we'll lose the respect for packaging standards that has long been (or at least been seen as!) a hallmark of quality Debian development.*

[178] <alpine.DEB.2.00.0812050821200.9613@wr-linux02>
[179] <20081107094111.GA25888@ouaza.com>
[180] <20090505203417.GA25853@bee.dooz.org>

> *To a large extent, this kind of thing tends to rest on evolutionary development, at least in terms of style. Perception of elegance and familiarity have something in common. Finding the minimum interoperability requirements that need to be expressed in policy is a useful exercise which often improves the simplicity and robustness of the end result.*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

An important message in this statement is the perception of familiarity, which offers another perspectives on the origin of authority of standards: an innovation has to have been around for long enough for everyone to embrace it; only then can it approach standardisation. Nevertheless, the process is anything but automatic.

Interfaces Standards define boundaries between sub-products and specify the interfaces necessary to assemble the parts [Loïc Minier, round three[181]]. The trick is to standardise the right things [Scott Kitterman, round two[182]], and defined interfaces allow for the use of different tools to accomplish the same task:

> *Having all the packages of a team in the same Subversion repository doesn't mean I must use the `svn` tool myself. Yay for `git-svn`.*
>
> — Niko Tyni, round two <20081130213055.GA19290@rebekka>

On the other hand, this is not possible for all kinds of tools, and network effects (see section 7.2.5.1) may require groups to settle on specific tools:

> *Mandating things in build-depends like quilt or CDBS is certainly dictating tools, and team-wide standardization of them is a good thing.*
>
> — Niko Tyni, round two <20081130213055.GA19290@rebekka>

Another participant separated the standardisation into rules and implementations [Jonas Smedegaard, round three[183]], which I reproduce verbatim:

- Defining rules, and evolving those rules in a slow pace.
- Avoiding a single implementation of rules.

Keeping the distinction between these two points allows for standardisation and maintains choice.

[181] <20090505203417.GA25853@bee.dooz.org>
[182] <200812070057.25291.scott@kitterman.com>
[183] <20090707172229.GE7723@jones.dk>

Free-riding standardisation As mentioned in the introduction of this influence, another source of *de facto* (implicit) standardisation is found in the base tools (*e.g.* `dpkg-dev`) and the `devscripts` collection. As soon as tools or techniques are added to those, their use tends to spread thanks to the already-wide-spread adoption of the base tools [Colin Watson, round one[184]].

For instance, the convention of splitting changes into groups according to who made them in the changelogs seems to have originated in the Debian installer team. These conventions were later used as the basis for implementation of the functionality into the `debchange` tool, which is a part of `devscripts`, and hence widely used. This is a good example of a standard that is *de facto*, but which has never been (properly) formalised:

> *The multi-author changelog notation is indeed a good example of standardization through a network effect. However, it has the drawback of not being machine-parseable, AFAIK: there's still no formal specification for the change details and not much chance to come up with one. Maybe policy-driven standardization would have given a better end result in this case?*
>
> — Niko Tyni, round two <20081130213055.GA19290@rebekka>

Learning from standards While standards are binding to a certain degree, and provide a strong argument in attempts to convince outlying factions, they also define the methods used by new contributors. Similar to the influence by mentors (see "peercolation", section 7.2.1.4), documents like the Debian Policy mandate practice even for those who were not part of the consensus-finding process that led to the formalisation. I described the role of such documents in terms of their function to "lessen the number of decisions a package maintainer has to make" [Krafft, 2005, p. 203].

> *Standards documents obviously have a substantial influence, particularly over the behaviour of new developers (long-established developers might try to change policy if they think it's introduced something wrong…). Once a technique gets written down somewhere that looks "official", a large number of developers will use it by way of a default behaviour. I remember that Raphaël Hertzog was particularly keen to get the Package Tracking System (PTS) described in the Developer's Reference, for example, and I seem to remember that its use increased significantly once that was done.*
>
> — Colin Watson, round one <20081109010317.GK4735@riva.ucam.org>

While there is a direct effect of formalised standards on newcomers, regular contributors may not re-read the documents and instead rely on quality assurance techniques to keep them

[184] <20081109010317.GK4735@riva.ucam.org>

on track [Stefano Zacchiroli, round two[185]]; this will be subject of the next influence in section 7.2.8.2.

The necessarily lengthy, evolutionary process leading up to standardisation, could also make such documents unsuitable as a reference for recent development, and teams might harness their smaller size and agility to expedite standardisation (*cf.* section 7.2.6.1):

> *Learning to package by studying the Debian Policy does not teach any recent best practice. In the Debian Med team, we have our "group policy" that adds the best practices that are not yet written in the Debian Policy (*i.e. *to forward patches and document it has been done in their headers, or to use a machine-readable copyright format).*
>
> — Charles Plessy, round one <20081108080522.GA28360@kunpuu.plessy.org>

7.2.8.2. Influence 23: Quality assurance

Gist: Automated quality assurance strengthens consensus and standards.

Summary used during study: Automated quality assurance (*e.g.* lintian) can drive adoption, but only if there exists consensus about the rules; they must not be abused to recommend or enforce policies for which there is no consensus.

The standards discussed in the previous section (section 7.2.8.1) are often binding, and they have substantial influence especially on new contributors, whereas regulars may not re-read standards documents and thus do not necessarily stay up-to-date actively [Stefano Zacchiroli, round two[186]]. To ensure the adherence to standards, which are after all fundamental to systems of a complexity such as Debian, automated tools have been created to fill the niche.

> *Packaging can be suboptimal in many ways. Having an automated checker helps prevent cutting corners.*
>
> — Damyan Ivanov, round three <20090419214402.GF18759@pc1.creditreform.bg>

The prominent automated checker is lintian[187], which is maintained in parallel to the policy and runs automated tests over a package to help spot policy violations. The use of lintian has, in turn, become strongly recommended by the Developer's Reference, and helps newcomers as well as regulars avoid common errors. Moreover, the "lintian lab"[188] collects the results from

[185] <20081130181416.GA13001@usha.takhisis.invalid>
[186] <20081130181416.GA13001@usha.takhisis.invalid>
[187] lintian got its name from lint, an old C code analysis tool.
[188] http://lintian.debian.org

the automated checks and provides maintainers with useful overviews of the status of their packages.

> *lintian is, I believe, kind of the greatest common denominator that all maintainers use, no matter how much they are involved in the community. When it issues a new warning, it will potentially reach a way wider audience than other mechanisms. In that sense, it is a pity that we do not exploit it more, even to prepare for future changes, but of course we have the principle that "policy follows adoption".*
>
> — Stefano Zacchiroli, round three <20090427090909.GA10691@usha.takhisis.invalid>

Aside from lintian, several other tools and components of the Debian infrastructure fall into the domain of quality assurance, which are relevant to adoption behaviour in the Debian Project; the PTS is another example. By checking for standards-compliance, these tools enhance consensus and strengthen the foundation on which to grow the Debian System [Luca Capello, round three[189]]. Such tools can also spread knowledge about changes [Scott Kitterman, round three[190]], and have enabled progressive learning through trial-and-error [Raphaël Hertzog, round three[191]]. A participant and member of the Debian Policy maintenance team had the following to say:

> *lintian is an excellent tool with huge effect on the project. When we've added something to Lintian, even at the off-by-default levels (info or pedantic tags), I immediately see wide-ranging impact on packages as shown in subsequent debian/changelog entries. For example, just recently, Lintian added a pedantic tag for using `#!/bin/sh -e` instead of `set -e` in maintainer scripts, and I was stunned at the number of packages that quickly changed.*
>
> *Now that ftp-master[192] is relying more heavily on lintian to check new packages, I expect this will increase.*
>
> *This is true of our other package audit tools as well. Anything that shows up on the PTS seems to have a significant effect.*
>
> — Russ Allbery, round three <87d4b9o4m7.fsf@windlord.stanford.edu>

Layered severities Somewhat similar to how new approaches have diffused by being tagged on to `devscripts` (see "standards", section 7.2.8.1), lintian has grown the ability to notify maintainers of more than just policy-noncompliance (*e.g.* it has been instrumental in the diffusion of `po-debconf` [Colin Watson, round one[193]]), but it uses different severity levels to

[189] <87hc0ks9sd.fsf@gismo.pca.it>
[190] <200905232049.11003.scott@kitterman.com>
[191] <20090413102934.GC23906@rivendell>
[192] The team exercising quality control over the contents of the Debian archive
[193] <20081109010317.GK4735@riva.ucam.org>

indicate the level of consensus behind each suggestion and thus provide a way to introduce changes slowly:

> *I regard tools such as lintian as a very very good way to drive change at the scale of Debian; they allow for gradual highlights of suggested, recommended, or required changes (policy-mandated for instance) and distribute the load effectively on all maintainers. Perhaps we will reach a point where lintian will tell you that your package isn't in a VCS and should better be?*
>
> — Loïc Minier, round three <20090505203417.GA25853@bee.dooz.org>

Not a consensus replacement

> *lintian is a conservative force for change: unlike many of the other things detailed in your list, it doesn't help with early adoption. It's much more of a trailing edge technique, so the situations in which it can be used are more limited.*
>
> — Russ Allbery, round three <87d4b9o4m7.fsf@windlord.stanford.edu>

As with standards, consensus remains in the foreground, and abusing a tool like lintian can reduce its legitimacy [Scott Kitterman, round three[194]]. For one participant, this point has apparently already been crossed:

> *I believe lintian is going too far nowadays. It is pushing for things I don't agree with and for which I don't think there is that a large consensus. I used to strive to comply with lintian, but I have been really annoyed by one or two warnings it issued recently. I don't have specific examples though.*
>
> *It's quite a shame, because lintian was an excellent way to deprecate techniques, and to track progress automatically through the lintian lab, but that is not so true nowadays anymore due to the excessive number of deprecation warnings it spits.*
>
> — Pierre Habouzit, round two <20081122152202.GA30285@artemis.corp>

Dangers One of the dangers of tools like lintian is that they facilitate action without embracing the reasons [Charles Plessy, round three[195]]. It is questionable though, whether this is problematic. During the discussion of transparency (section 7.2.4.2), participants voiced the concern about DCs using tools without understanding them, but there was also the notion of a cut-off level beyond which knowledge was not necessary. It is questionable how much automated checkers raise that level and further actions without comprehension.

[194] <200905232049.11003.scott@kitterman.com>
[195] <20090420141054.GA2267@kunpuu.plessy.org>

> *I am not aware of problems, which is why I chose to write "without really embracing" and not "without knowing". For instance, I do not know why it is good to switch from `dh_clean -k` to `dh_prep` apart that it is what the author of debhelper wants us to, and I got informed of this by lintian.*
>
> *Asking to fix lintian warnings is not the same as writing "manpages are something important and I would like you to make sure that there are no hyphen bugs in them before we upload your work". This is what I mean by "without really embracing".*
>
> *lintian is quantitative. We are taught at school and at work to perform well by such criteria. It often does more good than harm, but in Debian we have more freedom, so I always feel sorry when we reproduce that kind of behaviour here: doing things because it is in the manual or because somebody said to do so, instead of doing things because we want to see their effect happening.*
>
> — Charles Plessy, round three <20090712144552.GJ29671@kunpuu.plessy.org>

7.2.9. Stage nine: organisational initiation

In the IS implementation model by Kwon and Zmud [1987], initiation is the first stage of an organisational adoption process. As I have argued in section 7.1.2, the stage assumes a unique position in the context of the Debian Project, because organisational adoption is not initiated by the project, but the adoption cycle has to start with individual adoptions, due to the voluntary involvement of the project members.

Nevertheless, following individuals adopting an innovation, in the presence of organisational merit and the subsequent standardisation of the innovation, uniformity across the project rises to be not only an organisational benefit, but also a driving influence in the organisational adoption process, initiating further individual adoptions, which might yield improvements, that can later be incorporated. I present this influence in section 7.2.9.1.

Incorporation by standardisation (section 7.2.8.1) needs not be a prerequisite for the current stage; critical mass and consensus (see section 7.2.5) can already begin the organisational initiation by providing incentives for uniformity.

7.2.9.1. Influence 24: Uniformity

Gist: Uniformity paves the way to synergy, but that may not be reason to change for all.

Summary used during study: Consistency across the project as a whole can motivate change. A good reason for adoption of a tool/technique can be the desire

> to realign outlying factions, i.e. teams who are doing things differently. Driving that kind of change is tricky – events, such as negative incidents, can be good vehicles for change. Support by the project infrastructure is vital, if applicable.

A standard — whether explicitly formulated in a definitive document, or an evolved practice that exists as an unwritten rule — gives reason for people to strive to conform with it, for at least two reasons:

First, uniformity lessens complexity across the project, which can enable progress as approaches are streamlined.

> *While diversity is good to let many technique compete, there's a time when they have matured where that diversity hurts more than anything else and the project will try to standardize on the best option.*
>
> — Raphaël Hertzog, round one <20081107094111.GA25888@ouaza.com>

Second, an individual profits from uniform approaches, because it reduces the danger of making mistakes when working to make things fit in with other entities complying to the same standards, but this is difficult to quantify. One of the panellists thinks uniformity is more of a consequence, but carefully formulated the grounds for uniformity as an influence like this:

> *In my humble opinion, uniformity is in fact more a consequence than a factor, but it's also a goal some people try to pursue (and it is a good goal when it comes to computer stuff anyways).*
>
> — Pierre Habouzit, round three <20090424103539.GA26076@artemis.corp>

Collaboration The importance of uniformity increases with the amount of collaboration, because choice can split users, rather than unite them [Andreas Tille, round one[196]]. Group-specific documentation can result in divergence, and uniform documentation across the project helps to keep efforts in line with each other [Gunnar Wolf, round two[197]]. The Debian Project already limits choice at the project level through the Debian Policy [Pierre Habouzit, round one[198]], and it is easier for people to cooperate when standards are employed [Guillem Jover, round one[199]].

Uniformity furthers efficiency, because people will be able to focus their attention and efforts more on getting work done [*cf.* Luca Capello, round two[200]], instead of spending time learning

[196] <alpine.DEB.2.00.0811031039280.16818>
[197] <20081201014042.GB6395@cajita.gateway.2wire.net>
[198] <20081031224322.GE21799@artemis.corp>
[199] <20081109225614.GA8728@pulsar.hadrons.org>
[200] <87k5aksbpl.fsf@gismo.pca.it>

particular approaches [Luca Capello, round two[201]]. Uniformity, if evolutionary, does not need to inhibit diversity [*ibid.*].

It is easier for a single person to adopt a standard, then it is for a group to deal with non-standard ways [Enrico Zini, round three[202]]. Developing uniformly to what others do also means that others will be able to help out, or take over maintenance of packages [Gunnar Wolf, round two[203]].

Level of uniformity As stated in the discussion on standards in section 7.2.8.1, it is important to seek uniformity on the right level, and that is the level of interfaces, not individual tools.

> *Debian as a whole should be considered a public interface: we provide a defined infrastructure and some rules about how to interact with the various pieces of the infrastructure. We also cater to much diversity as possible, because we are clever and we like to work on stuff, even if we do not like how our fellows go about it. But in the end it does not matter whether you prefer Git or Subversion, CDBS or debhelper, because what we want is a Debian package which fits nicely in with the rest.*
>
> — Luca Capello, round two <87oczd9ic9.fsf@gismo.pca.it>

Ideally, uniformity exists at the level of the lowest common denominator (the largest common subset), but this can be difficult in the presence of complex packages:

> *Paradoxically, packages using simple tools can be the easiest ones to improve by submitting patches. The more active developers tend to have more complex packages with complex build patterns and many layers of customisation. Processes that need to make changes across large parts of Debian often come up against significant barriers, not with packages from less active developers but from complex core packages like gcc and glibc.*
>
> — Neil Williams, round two <20081121184632.106f2666.codehelp@debian.org>

At the same time, package maintainers can get carried away and make packages unnecessarily complex, for any number of reasons, such as experimentation, rigour, or fun [Neil Williams, round two[204]]. It is neigh impossible to prevent or discourage this behaviour, except when the standardised solutions are flexible enough, and yet, Debian has a bad track record of deprecating approaches [Stefano Zacchiroli, round two[205]], and moving away from them requires compatible

[201] <87oczd9ic9.fsf@gismo.pca.it>
[202] <20090421104533.GA5652@enricozini.org>
[203] <20081201014042.GB6395@cajita.gateway.2wire.net>
[204] <20081121184632.106f2666.codehelp@debian.org>
[205] <20081130181416.GA13001@usha.takhisis.invalid>

replacements, possibly along with removal of the old tool(s) by authority decision [Neil Williams, round one[206]].

Uniformity as an initiator to organisational adoption needs to walk the fine line between flexibility and abstraction (see "transparency", section 7.2.4.2).

> *debhelper is meant to be used by DDs who want maximum flexibility, but are also meant to work* together *to create a distribution of packages which should be built in similar ways. While debhelper is enough to guarantee flexibility, it is not enough to guarantee uniformity: that's where we need something else (debhelper7, CDBS, ...).*
>
> — Stefano Zacchiroli (his emphasis), round two
> <20081130181416.GA13001@usha.takhisis.invalid>

It seems that the right compromise involves iterative trials, but this is purely speculative on my side: CDBS went too far into abstraction, and even debhelper7 initially leaned too much in that direction (see "modularity", section 7.2.4.1). Both tools have since been amended towards the middle.

Triggers While uniformity may be considered an implicit initiator, giving individuals a reason to adopt the tool or technique commonly used by others (thereby reducing their own dissonance), there are occasions in which the strive to uniformity can be explicitly triggered:

> *After the "OpenSSL accident", a long discussion took place on the debian-devel mailing list about divergence from upstream, and although no formal rule emerged from it, we changed our practices in Debian Med, taking better care to document and forward the patches we apply in our packages.*
>
> — Charles Plessy, round one <20081108080522.GA28360@kunpuu.plessy.org>

Even though the OpenSSL debacle cannot be reduced to lack of uniformity,[207] one argument was that the problem might have been prevented, had there been a uniform, canonical resource to track divergence between Debian and upstream software, which would have increased the chances of someone else spotting the error. Out of the discussion emerged improved guidelines for patch management (the `README.source` specification), as well as a project-wide patch tracker.[208] The use of patch managers for packages has continued to rise, and more packages are advertising their use of VCS.[209]

[206] <1225790047.31771.202.camel@holly.codehelp>

[207] Multiple instances of human error came together so that a bug introduced by a Debian-only patch was not noticed by several pairs of eyes until it was (way) too late.

[208] `http://patch-tracker.debian.org`

[209] The OpenSSL incident happened approximately half way between the Debian releases "etch" and "lenny". In "etch" 14.59% of source packages were using one of the two main patch managers (dpatch or quilt), and in "lenny",

But apart from such extraordinary triggers, organisational uniformity often depends on infrastructural support, as suggested in the following statement:

> *There are cases where it's clear that the adoption curve goes nowhere fast unless the technique in question shows up somewhere nice and central: the next-generation dpkg source format is clearly unlikely to be adopted until dpkg-dev in unstable supports it, not to mention the archive maintenance scripts.*
>
> — Colin Watson, round one <20081109010317.GK4735@riva.ucam.org>

I will summarise my contribution, discuss implications for research and practice, and consider shortcomings and problems of my approach in the next chapter (chapter 8).

the percentage grew to 25.66% (it was 5.60% in "sarge", the release prior to "etch"). The number of packages advertising their use of VCS grew from 3.81% in "etch" to 32.23% in "lenny". These statistics only take into account the number of packages, not only the number of packages with non-trivial divergence from upstream, which would be lower, meaning those percentages would be larger.

Chapter 8

Conclusion

Following the self-contained discussion of results in chapter 7, this chapter discusses my contributions (section 8.1), as well as the implications for practice of the results (section 8.2). I then review my research approach in section 8.3 and reflect upon design decisions alongside participant feedback in section 8.4. Towards the end, the reader can find a number of future research directions that this work created.

To facilitate reading and using these sections in the context of the 24 influences presented in chapter 7, I include the influence name next to the section reference. For instance, if I am talking about marketing, a section reference could appear like this: ("marketing", section 7.2.1.3).

8.1. Research contribution

My research makes a number of contributions, to the Debian Project and FLOSS in general, as well as to academic researchers studying those domains, or wishing to employ the Delphi method in similar contexts. I summarise these contributions in the following.

In section 2.4, I identified a divide between the slow adoption and availability of efficient tools and techniques that could help the project scale better. The underlying challenge lies in the voluntary nature of Debian Contributors (DCs), and the lack of authoritarian decision-making structures to push changes. Therefore, change depends on members of the community deciding for and implementing it. My goal was to determine the influences that shape these adoptions (see section 1.2).

DCs are increasingly investigating and using version control systems (VCSs) for packaging, to facilitate collaboration, to maintain changes, and to track the history of packages for later debugging and accounting purposes. This trend is not fresh, maybe a dozen different approaches to using VCSs already exist, but definitive solutions have yet to be found. If the tool/technique-space is not to become further fragmented, the project will have to focus on a common way forward.

However, Debian is a project of volunteers without an authority in a position to mandate methods. Rather, as exemplified by the numerous statements in chapter 7, solutions have to diffuse, stand up to scrutiny, converge with the tasks, and reach a level of maturity, before they can be documented as a standard due to wide-spread use and the right balance of consensus and decision. Interdependencies between teams and tools necessarily slow down this process, and require the most careful definition of standards to not alienate contributors; interfaces, not processes, are key to successful standardisation, and automated methods that aid contributors maintain compliance are necessary.

Classical diffusion and adoption studies have thus far either focused on authoritarian diffusions, or individuals, whose adoptions did not have much of an effect on other adopters. Therefore, none of the existing diffusion frameworks seemed to fit, as I have argued in section 3.2. I proceeded to collect the data such that a structure could emerge, and have designed a presentation framework in form of a temporal, cyclical stage model to which I will return shortly.

The data are the constituents of my primary research objectives, all of which I have achieved:

1. The 24 influences presented in section 7.2 are salient influences to innovation adoption behaviour of DCs. The set may well not be comprehensive, but due to the process by which they were identified, it is constituted of the most salient influences. In section 8.5, I will present future directions in which these results can be further enhanced.
2. I have given each of these 24 influences a name to describe their angle and content. These names were not immediately grounded in the panellists' arguments and I should have used the opportunity to derive a terminology for them in cooperation with the panellists, but the influences are clearly defined and labelled, and I explicity described the gist of each. In section 8.3.1.4 I will return to the grounding issue.
3. From the discussion of these influences, I crystallised implications for practice, which I have compiled in section 8.2.

The major contribution of my work is hence the set of 24 influences, as well as the implications for practice in section 8.2. The set of influences can educate leaders and contributors alike, focusing their attention to a number of core considerations, and providing a lens through which to assess and engineer diffusions. The implications, on the other hand, provide concrete guidelines for those designing and diffusing tools and techniques in the Debian Project, and paying attention to them should allow diffusers to speed up the rate of adoption. In my theory, this will enhance competition, drive progress, and increase the scalability of the Debian Project.

8.1.1. Influence salience

In the third Delphi discussion round (see section 6.2.6), I invited the panellists to select the three influences they deemed most salient with respect to the Debian Project at large, and to write about them. Although the primary goal of this round was not to establish a ranking of influences, a short discussion of the frequency of choices seems in order.

Since I let the participants choose combinations of influences, rather than forcing them to limit their choice to three single items, any analysis of frequency of choices needs to take that into account. I chose a very simple scoring method: each participant had three choices, and each choice was worth one point. If a participant combined n influences into one choice, each influence was scored with $1/n$. This is a naïve approach, but any more involved analysis would be senseless given the non-quantitative nature of the data. Table 8.1 on the next page reports the final scores and ranks the influences in decreasing order.

Even though I do not want to blow the significance of these data out of proportion, for their census was not without faults (see section 8.3.2.4), and never aimed for representitativeness, it is nevertheless interesting to note that the top-most influences reflect the nature of the Debian Project as a complex, consensus-oriented, interdependent, and tightly-connected association of peers. I was furthermore surprised by the relatively low ranking of influences such as sustainability, modularity, trialability, and cost-benefit.

If there is one conclusion to be reached from the scores in table 8.1 on the facing page, then adoption behaviour in the Debian Project is a lot more a social phenomenon than it is an individual or technical aspect.

Influence	Score	Rank
Quality assurance	9.08	1
Consensus	6.17	2
Peercolation	6	3
Network effects	5	4
Genericity	3.58	5
Sedimentation	3.5	6
Marketing	2.83	7
Compatibility	2.5	8
Standards	2.5	8
Resistance	2.33	10
Transparency	1.58	11
Return-on-investment	1.5	12
Uniformity	1.5	12
Scaled use	1.33	14
Cost-benefit	1.33	14
Maturity	1.17	16
Quality documentation	1.17	16
Trialability	1	18
Modularity	0.75	19
Elegance	0.5	20
Examples *vs.* documentation	0.5	20
Sustainability	0.5	20
First-impression	0.5	20
Chunking	0.17	24

Table 8.1.: *A naïve scoring of influences using frequency of nomination in the third Delphi discussion round.*

8.1.2. A stage model of adoption in the Debian Project

Once I had identified the 24 influences, I was able to build a presentation framework. The result is the adoption cycle depicted on page 141, which is a cyclical combination of two linear stage models, namely the individual innovation-decision process proposed by Rogers [2003, ch. 5], and the organisational information systems (IS) implementation model by Kwon and Zmud [1987]. Essentially, the combination happened through three steps:

1. I replaced organisational resource allocation (the organisational adoption stage) with individual stages because individuals are the "resources" of FLOSS organisations, but they are not allocated and rather contribute to organisational adoption by way of their own adoptions;

2. I fused the organisational use-performance-satisfaction stage, which is a stage about individuals in an organisation, with the individual confirmation stage;
3. I linked the ends to form a cycle, but let the cycle start not with organisational initiation (which does not exist up front in a voluntary setting), but with pioneering activity and individual adoptions. Organisational initiation thus does not begin the initial iteration of stages, but precedes subsequent revolutions of the cycle.

Even though a spiral might have been a better metaphor, and the position (and role) of the use-performance-satisfaction/confirmation stage is not clear, the cycle as depicted presents what I believe to be a useful stage-model for a complex process, without being overly complex itself. The 24 influences fit the stages without forcing.

8.1.3. Specificity of results and stage model

None of the 24 influences appear to be specific to Debian. In fact, I expect all of them to play a role in the adoption decisions of participants of other FLOSS projects, some more and some less; some influences may even carry to other volunteer projects, even though the set I found through my research has a strong orientation towards technology and software packaging.

The degree to which an influence is relevant in the context of any project depends on the circumstances. Specifically, size and complexity of, and diversity within the project and its produce, its governance structure, and the nature and experience of its contributors among others seem to determine the degree to which each influence affects adopters in different contexts.

Nevertheless, a number of influences appear to be largely ubiquitous. I shall discuss these first before looking at the effects of each of the aforementioned criteria.

8.1.3.1. Project-independent influences

A number of influences seem to have an effect on contributors to all kinds of FLOSS projects, suggesting that these relate to specific features of FLOSS communities. For instance, the influence of an innovation's compatibility or trialability, or considerations related to cost-benefit evaluations, are likely relevant in any voluntary context where individuals strive to produce, rather than focus purely on the processes.

Unsurprisingly, these influences all have correspondent innovation attributes in Rogers' individual diffusion framework [2003]: compatibility and trialability reuse the names of Rogers' attributes, and three of the influences I have identified relate to the relative advantage of an innovation: sustainability, cost-benefit, and return-on-investment. One could argue that return-on-investment also relates to observability.

The reason for this overlap is to be found in the generality of Rogers' framework, which has been successfully applied in thousands of different contexts — one of the shortcomings of this work (see section 3.2.2.1). The same applies to the Technology Acceptance Model (TAM), which only frames two highly generic evaluative criteria that reflect the cost-benefit influence on the one hand, and a conglomeration of chunking and compatibility on the other. Its successors provide more granular frameworks, but still only focus on the process leading up to adoption decisions (see section 3.2.2.2).

Rogers' fifth attribute, complexity, touches many of the influences I identified, *e.g.* sedimentation and chunking, modularity and transparency, and documentation and examples. Complexity, however, has many facets that make it depend on contextual circumstances, and these influences need further investigation when it comes to their applicability in other contexts.

The model proposed by Wejnert [2002] could also accomodate the independent influences mentioned at the start of this section. Her variable "public *vs.* private consequences" relates to network effects, familiarity is strongly related to compatibility, status characteristics and position in social networks at the core of "peercolation", and she even includes uniformity in her list.

8.1.3.2. Influences not captured by existing frameworks

In section 3.2.1, I speculated that existing diffusion frameworks could be inadequate to capture the adoption behaviour of FLOSS participants, and DCs in particular. Instead, I let the model presented in section 7.1 emerge from the data. Given the influences, I can now look back at the existing diffusion frameworks to identify specific inadequacies or shortcomings.

The Technology Acceptance Model The easiest, most obvious, comparison is with the TAM, which only captures (individual) perceived usefulness and perceived ease-of-use, and only looks at the adoption process up until the decision. Therefore, the TAM does not capture any of the

influences past the adoption decision, which includes most organisational aspects of diffusion. The TAM2, TAM3, and Unified Theory of Acceptance and Use of Technology (UTAUT) extensions (see section 3.2.2.2) do not overcome these influences, although they include constructs spanning social influence and cognitive instrumental processes.

Rogers' theory of diffusion of innovations Rogers' diffusion framework is about individual adoption decisions without network effects. His attributes of innovations (the first element of the framework, see appendix B.1.1) represent a number of the influences I found during my study, albeit in rather generic ways (see section 8.1.3.1). While these attributes are mostly concerned with individual persuasion, other elements in his framework are concerned with matters of communication, and traits of the social system in which adoptions occur. Nevertheless, these generally suffer from the same problems related to generality.

For instance, the nature of communication (see appendix B.1.2) is split into medium (personal *vs.* mass-communication) and homophily of peers, which are but two aspects of influences such as sedimentation, marketing, peercolation, and chunking. I see no way to express the idea of chunking in Rogers' framework, nor the necessity of sedimentation as a process. Peercolation could refer to a network of homophilous peers, but it is unclear whether homophily should refer to cooperation, personal acquaintance, or sympathies.

The fourth element of Rogers' framework describes the social system in which an adoption process takes place (see appendix B.1.4), and encompasses large parts of the previously discussed communication element. This element is concerned with the spectrum of how decisions are made, from voluntary to authoritarian, and it even references consensus as a way of collective decision-making, but seemingly without considering network effects (which is not always just a consequence of an adoption, but often a limiting factor). The effects of resistance cannot be captured either.

Another shortcoming I mentioned already in the introduction of chapter 7 is concerned with differences in the desirability of re-inventions. Rogers views the degree to which an innovation is re-invented by members of the social system as having a net positive effect on adoption rate and sustainability of an innovation, but this fails to take into account the desire not to diverge too much from how a tool or technique is designed to be used commonly found among DCs (see section 2.2.4.5) — which is a requirement for flexibility, rather than a refusal to re-invent.

Re-inventability is a feature of an innovation, which is captured by the influences to be found at the individual implementation stage of my model (modularity, transparency, and genericity).

For Rogers, re-inventions are, however, simply a possible (even undesirable) outcome of the innovation-decision process, not an attribute of an innovation, *i.e.* his model does not account for the flexibility of an innovation being taken into account by potential adopters when making a decision.

Despite these aforementioned nadequacies, Rogers [2003] presents a useful framework that shaped the results presented in section 7.1 in two ways: (1) I used the time stages identified in the third element of diffusions (see appendix B.1.3) to display the influences to individual adoptions, and (2) in section 7.1.1, I found Rogers' model of organisational adoption to be on par with the one by Kwon and Zmud [1987] I eventually used.

Wejnert's integrated model of innovation diffusion The model by Wejnert [2002] does not capture influences related to communication, flexibility, matters relating to consensus and resistance, or personal preference, such as elegance. At first glance, it seems that many influences neatly slot into place in her framework, but when reading the descriptions for each of the variables, these associations turn out to be premature, as the variables are defined rather broadly, and with a certain degree of vagueness.

To give an example: while the name "political conditions" suggests at first to cover matters relating to consensus-building, resistance, and standardisation, the variable seems more concerned with governance, censorship, and authoriarian structures. A link is not entirely non-existent, but it would require a lot of interpretation to put forth a coherent case.

Wejnert's framework appears highly specific to political studies, and I could not find any instances of use other than by Wejnert [2005] herself, studying the diffusion of demcracy. While the Debian Project exposes political traits, the project is primarily about voluntary cooperation and technical output, which I think accounts for the aforementioned vagueness.

8.1.3.3. Effects of project variables on influence relevance

The 24 influences have been extoled in the context of the Debian Project. While they are likely to apply to other projects, and especially FLOSS projects, to varying degrees, I expect certain project-specific features to have an effect on the relevance of each influence. In this section, I investigate the effects of the most obvious of these features.

Size, complexity, and diversity Network effects stem from dependencies between processes, and between actors. Such effects will likely be much stronger in projects producing a single software, because contributors are working on the same code. On the contrary, when a group works on sets of applications and libraries in a loosely-coupled way, as might be the case in projects managing collections of software, including entire operating systems, overall dependence decreases due to increased granularity and freedom of choice. Standards and automated qualiy assurance to specify and verify the end product are important to manage the diversity.

Contributors to the former kinds of projects are likely going to be limited in their choices, as they will have to fit in with existing practice (or attempt to persuade people to switch, which is seldom a good way to join a community). Unless existing processes are sufficiently standardised and modular to allow for choice at different levels, individual adoption processes are going to focus more on the adoption of a particular tool, rather than on choice. Chunking and scaled use are of benefit in those cases, as is documentation.

This comparison between software projects and distributions is between apples and oranges in some ways. For simplicity, I concentrate on extremes and consciously ignore that even projects producing single applications, however small, may well encompass multiple endeavours: code, artwork, documentation, *etc.*. Network effects do not necessarily persist across sub-endeavours: code maintained in a version control system does not keep documenters from collaborating using *e.g.* a Wiki, much in the same way that unrelated packaging efforts in the Debian Project only need to observe the standardised interfaces but are mostly free in their choice of tools and techniques.

A distribution like Debian consists of hundreds of sub-projects, and most of those have established procedures to be adopted by newcomers. Only new or young sub-projects may leave choice, just as it would be the case with new or young software projects. With reference to the adoption cycle depicted in figure 7.1 on page 141, individual choice decreases with increasing iterations of the cycle; it may simply be that smaller projects are quicker to complete the first revolution, while even small sub-projects in the context of a complex organisation such as Debian will be slower.

Single software projects are often smaller, less complex, and streamlined than the Debian distribution, and adopters therein are going to be faced less with issues around standards, uniformity, and maturity, mostly due to the lower effect of changes: changing the version control system used to maintain the code of a small application incurs a low volume of modifications, making it less necessary to standardise on uniform approaches, or requiring the use of mature tools to

prevent such required changes. The Debian Project, on the other hand, must build on mature tools, as well as well-defined, conservative standards, and contributors are more likely to prefer tools and techniques that blend well with existing approaches (uniformity), because the project's inertia is too large to move against it.

At the same time, however, changing processes in small projects may require special care in terms of documentation, sedimentation, and chunking to help contributors follow along. If the new is not compatible with the old, individuals may otherwise not be able to make the switch and be henceforth excluded from contributing to the project, which could prove detrimental to the project itself.

Communication culture The relevance of the influences related to communication — marketing and peercolation — increases with increasing diversification of communication media, as well as increasing volume of communication and available information. Where messages generally reach people, marketing will not be necessary, and all communication seen as a form of peercolation. As soon as the volume grows, causing people to selectively subscribe to channels or filter communication, active marketing gains importance, and messages from peers increase in relevance.

Nature of produce Another distinction exists in terms of the proximity between the product and the tools and techniques used to develop it. For instance, software packaging techniques are closer to the output (software packages), than *e.g.* a version control system to a web application. It might be that such influences as transparency, modularity, and also the balance between documentation and examples are more relevant in the former case, while contributors working with tools less related to the final output are more inclined to treat their tools as monolithic, black box entities. Compare a taylor and his/her sewing machine to an author writing a text, for whom a text editor or word processor is just one way of attaining the goal.

A related case can be made for the Unix philosophy. Since the Debian Project, as producer of Debian GNU/Linux, uses Unix tools to create a Unix-like operating system, influences relating to the Unix philosophy are paramount: maturity, transparency, modularity, and genericity. A lot of these relate to a perception of elegance, which, albeit subjective, is shared among those working close to the Unix core. In projects further away from the operating system level, other perceptions of elegance will prevail.

Governance The kind of governance in a project determines the salience of many influences. Purely authoritarian enterprises are best advised to pay attention to influences such as chunking, resistance, sustainability, documentation, compatibility, cost-benefit, and consensus to avoid alienating people. The early adoption stages (knowledge and persuasion) are less important, because a decision has already been made. Instead, the focus should be on individual implementation, as well as organisational adaptation.

With less authoritarian decsion-making, dealing with resistance, and aiming for the right level of consensus rise to the top. Whether critical mass can be achieved determines the success of a diffusion, and individual decision-making becomes the pinnacle; the focus should therefore be on the early stages, knowledge and individual persuasion.

8.1.4. Secondary research objectives

I have also achieved both of my secondary objectives. First, all the data from the study are available from my research webpage[1] and may be used by future researchers under the licence shown on page xvi. I believe that there are several other research topics in which these in-depth data can be used, or research endeavours that build upon them.

Second, my research has demonstrated the suitability of the Delphi method in a FLOSS context, because it is flexible enough to be fit to different social settings, because the aspect of anonymity enables participants to concentrate on merit, rather than be distracted by preconceptions about others, and because the controlled feedback component of the method fits the typical communication mode of a FLOSS project quite spectacularly. Discussions in FLOSS projects often degenerate into widely-branched threads including tangential discussions and personal attacks, and the Delphi method is an effective means to maintain focus.

The Delphi method is, however, not a cookbook approach, but a resourceful collection of tools which can be combined to fit the approach as close as possible to the research object. For lack of previous applications of the Delphi method in the FLOSS domain, I expended considerable time on investigating the different possibilities, their strengths, and their disadvantages. The design I devised proved adequate to research within the Debian Project, as exemplified by the discussion and participant feedback in section 8.4. Debian may be a peculiar project for its rigorous adherence to freedom, quality, and other ideologies, but it is not radically different from FLOSS projects in general. I believe to have identified the important knobs and argued their settings for my goal

[1] `http://phd.martin-krafft.net/data/`

in section 5.4, and I expect this discussion to aid future users of the technique in setting the knobs in a way appropriate to their FLOSS-related tasks.

The Delphi technique is furthermore a time-intensive process, at least for the facilitator, who has to carefully plan the study, and deal with potentially large amounts of data throughout. I expect that (a) my elaborate discussion of the Delphi method in this thesis (chapter 5), (b) the argumentation for the Delphi's suitability to FLOSS research (section 5.2), (c) the design considerations presented in section 5.4, (d) the detailed account of the research approach in chapter 6, (e) the discussion of problems (section 8.3), and (f) the review of the design (section 8.4) will make it easier for future researchers to design their own approaches, estimate the time and resource requirements, and find guidance for their conduct. Table 8.2 on page 285 summarises potential pitfalls in the use of the Delphi method, augmenting Linstone [1975].

Finally, by engaging myself and the panellists into this exercise, I have brought the Debian Project and the Delphi approach a bit closer together. The project is starting to employ the Debian Enhancement Proposals (DEP) process,[2] which is closely related, and insights gleaned through my research and documented in-depth in this thesis will hopefully help improve this process further.

8.2. Implications for practice

Out of the discussion of influences in section 7.2 emerged a number of implications for practice. I would like to present these in itemised form, and I shall formulate them imperatively, as advice to those interested in increasing the adoption rate of a given tool or technique, or to optimise its diffusion from the start.

1. The Debian Project is a slow-moving body, and even though the percentage of early adopters in the project may be higher than usual, ideas still need time to spread and settle ("sedimentation", section 7.2.1.1). Furthermore, for project-wide adoption and incorporation into standard processes, resistance needs to be overcome ("resistance", section 7.2.2.3), solutions need to gain maturity ("maturity", section 7.2.5.3), and standardisation is an evolutionary process ("standards", section 7.2.8.1), which similarly takes time. Driving change in Debian requires patience and persistence.

[2] http://dep.debian.net

2. Investigate how to split an innovation into smaller pieces. These are easier to understand ("sedimentation", section 7.2.1.1), try out ("trialability", section 7.2.3.3), and put to use (sections 7.2.4.1 and 7.2.4.2).

3. If possible and worthwhile, convert a revolution into a series of evolutionary steps ("chunking", section 7.2.1.2). This can keep the learning curve flat ("compatibility", section 7.2.3.4), prevent resistance ("resistance", section 7.2.2.3), and paves the road for scaled use ("scaled use", section 7.2.5.2).

4. Marketing — reaching out to people with truthful and down-to-earth statements — is a necessary part of a diffusion. Properly designed marketing messages can create exposure and excite curiosity, stimulating future investigation (section 7.2.2). Such messages also indicate that a project is active and suggest that an adoption is a future-proof investment ("sustainability", section 7.2.2.4). Multiple messages from different angles and of different styles are required to create a sense of familiarity among recipients, and to reach as many people as possible ("sedimentation", section 7.2.1.1).

5. Obtaining feedback from respected, experienced peers early on in the process helps to ensure that an effort is not completely misguided, and such respected users are key to spreading credible knowledge throughout the social system. If such peers are convinced and carry on a message, the rate of adoption will greatly profit ("peercolation", section 7.2.1.4).

6. Anticipate negative backlash when taking an idea public, and carefully prepare for it through prior discussions with peers from different fields of interest. Problems need to be made explicit alongside the benefits, and it might help to include arguments for some major decisions that were made. Try to propose an idea for further development, not impose a static view ("consensus", section 7.2.6.1).

7. While a positive first impression is desirable, avoiding a negative first impression is more important, because it is hard to overcome, and may ripple through the social system to inhibit other potential adopters ("first impression", section 7.2.2.1).

8. If possible, reuse familiar concepts, which may well instill a sense of elegance on the side of the adopters ("elegance", section 7.2.2.2). If the innovation is so groundbreaking that it parts with existing practice, metaphors and documentation mapping between familiar concepts and the novel ideas can make an innovation more accessible ("compatibility", section 7.2.3.4).

9. Consider actively supporting adopters, *e.g.* with specific documentation, tutorials, and demonstrations. Helping with the implementation, or doing the boring groundwork can be key to overcoming resistance ("resistance", section 7.2.2.3) and lowering costs ("cost-benefit", section 7.2.3.5).

10. A clear direction and an open, community-based development approach convey the impression that a project is here to stay, and that decisions have been made based on consensus, rather than in what may appear to be arbitrary ways ("sustainability", section 7.2.2.4).

11. Examples do not replace documentation, but are an important part of complete documentation, because they offer a quick glimpse of a tool, allowing an adopter to gain an idea of how an innovation is used, before having to invest time to try it, or enable a returning user to quickly regain an understanding of how it works ("examples", section 7.2.3.1).

12. Templates often bootstrap individual usage and are thus influential, both in terms of furthering adoption (*e.g.* dh_make), as well as establishing basic usage patterns (see "examples", section 7.2.3.1).

13. Quality documentation includes tutorials, as well as background information on the concepts and motivations behind approaches ("quality documentation", section 7.2.3.2). Knowledge of principles helps earlier adopters understand and support the diffusion ("sedimentation", section 7.2.1.1), while how-to knowledge lowers the barrier to entry for later adopters ("compatibility", section 7.2.3.4).

14. Mailing list archives do not replace documentation, but mailing lists are fertile ground for important ideas, explanations, and answers to common questions. These need to be pulled together in a central location without flooding this location with too much verbosity ("quality documentation", section 7.2.3.2).

15. As many potential adopters prefer to try out an innovation before putting it to use, it is essential to make an innovation trialable. This might be achieved through tutorials, step-by-step instructions on how to set up the necessary infrastructure, or providing access to test instances ("trialability", section 7.2.3.3).

16. Proposals and other entities on which to seek consensus need maintainers willing to make decisions to end otherwise interminable discussion and let an approach stabilise ("consensus", section 7.2.6.1). This also applies to software, which has matured ("maturity", section 7.2.5.3).

17. Build consensus in teams and smaller groups, and formalise approaches. This is easier than it would be in larger groups, helps anticipate problems, and builds momentum ("consensus", section 7.2.6.1).
18. Standardisation enables progress, but it is important to standardise at the right level. By defining interfaces, one lets people choose the tools between them, while still fostering collaboration ("standards", section 7.2.8.1).
19. If possible, reuse previously adopted tools or techniques, but beware of abusing such means, *e.g.* through imposing controversial solutions for which no consensus has been built, or through the use of such a vehicle to spread an unrelated concept ("standards", section 7.2.8.1).

8.3. Review of the research approach

My research approach was well-suited to the task, because it was exploratory, and the Delphi method fit the "research object" very closely — all participants of the Delphi study were comfortable with the task, and no significant problems were reported, which might have tainted the results.

However, the study itself was not without problems, even if the panelists were not directly subjected to them. I spent a considerably amount of time and effort on the identification and analysis of the problems I encountered, but not always was I able to deal with these problems to my own satisfaction. As a result, the study as conducted has a few shortcomings. I consider it important to make these shortcomings explicit, which should improve the value of the results, and also facilitate the endeavours of future researchers who work off my findings [*cf.* Schultze, 2000].

Possibly the biggest problem throughout the study was the volume of data with which I had to deal, which was responsible for delays (*cf.* section 8.3.2.4) and issues of bias (see section 8.3.1) during the study. The reasons for the high volume lie in the nature of the study, as well as in some of the design choices I made. I discuss such problems with the design and conduct of the study in section 8.3.2.

The Delphi method itself also has limitations, which I have addressed in section 5.4. Hung et al. [2008] provide a comprehensive overview of strengths and limitations.

8.3.1. Bias

Bias can hardly be avoided, especially in a qualitative study, which involves written and spoken words, opinions, and the possibility of multiple interpretations of statements, or misunderstandings. It is therefore important to be explicit about potential sources of bias.

8.3.1.1. Sampling bias

Even though I sought to eliminate selection bias through a combination of snowball and stratified purposeful sampling strategies, the process of sampling panellists was sufficiently elaborate to open several avenues for bias. Furthermore, I made several decisions that clearly influenced the study. In hindsight, those were methodological mistakes, even though I had good reasons at the time, and I do not believe that the outcome of these decisions had a diminishing effect on the validity or value of the result.

Unable to undo these mistakes, I am explicitly documenting the sources of bias of which I am aware in the following paragraphs to enable better assessment of the validity of the data.

Hand-picked nominees While the sheer quantity of the initial call for nominations I sent (162) makes it unlikely that I asked (too many of) the wrong people, I augmented the rules for nominee selection by including 5 lesser nominated people in the final set of recipients of the invitation to apply to the study. In each of the five cases, I had good reasons for doing so, but none of these reasons stand up to scientific scrutiny; instead, they were based on intuition (*cf.* section 8.4.1).

For instance, one participant had only been nominated once, and I still sent him an application (and he ended up on the panel), because he came to Debian from Ubuntu and was described a critic of the Ubuntu workflow, which was diffused inside the project in an authoritarian manner. Another nominee, who also became a panellist, was a proponent of the workflow, so selecting both seemed to increase the diversity of the panel.

In the cases of the other four manually selected onto the list of invitees, I had similar reasons, but only the candidate I explicitly mentioned above was selected onto the panel. Nevertheless, the inclusion of all five of them in the selection process was an instance of researcher influence.

Panelist selection The panellist selection process involved the selection of 16 candidates according to a knowledge resource nomination worksheet (KRNW) in the form of a four-dimensional hypercube model (which has 16 corners). With this approach, I selected for maximum diversity along the four strata identified in section 6.1.5. In addition, I selected three candidates onto the panel who were located closer to the centre of the hypercube than any of its corners. There are two source of bias in this approach:

Firstly, I picked the four strata along which I selected for diversity. A different set of strata would have yielded a differently diverse selection. However, I believe I sufficiently argued the appropriateness of the four strata in section 5.4.3. I'll return to this in section 8.3.2.1.

The second source of bias is more pronounced: the process required me to select 19 out of 43 people. Even though the hypercube selection method provided reasonable guidelines for the selection, I made a few decisions which potentially introduced bias:

- A number of corners could not be populated based on the KRNW alone, and I was forced to hand-pick candidates who fit the combination of strata values, even though their KRNW entries didn't suggest so, most likely due to inadequate coding.
- In a number of cases, I preferred a candidate who loosely fit into a corner over someone who was a closer match, because the latter candidate had reported not enough available time, or considered him/herself a team player but had not been collaborating much in the past year.
- In two cases, a candidate was the best fit for both of two corners. I had to chose one corner, and populate the other corner with another candidate. I tried to reason my choice based on the candidate's profile, rather than on which slot I would rather fill with someone else, but I cannot rule out that looking at the next step influenced my decision for one of two corners.

Accepting two further candidates As described in section 6.1.6, I politely declined the 18 applicants who did not get chosen for the compensated panel positions, but invited them to let me know if they were interested in participating regardless of compensation. Since I was expecting participant drop-out, as well as a slew of other problems, I deemed it a good idea to have a number of additional candidates available, who would follow along until needed.

Unfortunately, I failed to clearly communicate this to the two non-compensated participants, who expended considerable effort to participate in the first round, and sent in their responses

like everyone else. This put me in an awkward position, and while luck had it that their first round responses did not introduce any bias, I cannot rule out possible influences on the validity of the study during the second Delphi round.

By nature of the third round, and specifically the fact that it was the final round and the participants did not receive feedback to their responses, the bias due to the inclusion of the two non-compensated panellists in the third round was easy to contain.

The two non-compensated panellists were both team players using uniform tools. One claimed to be working on a diverse set of tasks, and he was also not specifically interest in workflow improvement. The other reportedly has a rather uniform set of tasks and a moderate interest in workflow improvement.

I am unable to quantify the extent of bias due to their participation in the second round. However, if one is ready to accept the assumption that Debian is constantly moving towards uniformity in the tools (*cf.* the last two influences, standards and uniformity, in sections 7.2.8.1 and 7.2.9.1), and that team maintenance will become more and more of a requirement with increasing complexity, then the impact is future-bound and thus inherently positive.

8.3.1.2. Data processing bias

As the moderator of the Delphi study, I was in the position to introduce significant amounts of bias into the research (see section 5.4.10). I can identify three ways in which I have influenced the study during the Delphi study, both related to data processing: data reduction, and exploration of participants' arguments.

Picking statements from initial responses In the responses from round 1 of the Delphi discussion, I tagged keywords onto paragraphs and statements by the respondents which identified influences in adoption behaviour (see section 6.2.3). This allowed me to reduce 3,500 lines of text to 1,300 lines, and group statements into 15 groups,[3] which made the data set more manageable albeit still huge.

Nevertheless, this selection of relevant paragraphs and sentences from the respondents' replies obviously bore the danger that I might have left out some, or included others that were not actually relevant.

[3] see section 8.3.1.4 for the discussion on how this grouping also introduced bias.

Since I was aware of problems with this form of reduction at the time, I specifically asked the panellists in the second round to watch out for omissions or misrepresentations in the data I presented, highlighting the fact that I reduced their size to a third. I received not a single reply from a respondent who did not see his/her point of view adequatly respresented, which lead me to assume that the data were shortened without significant loss of content.

On the other hand, the amount of data presented to panellists at the start of round 2 may have also drowned them to the point were they were unable to judge whether their previous statements were properly included, or not.

A second or third person to supervise the task of picking relevant statements from the responses would have been beneficial.

Condensing and rewording statements A larger amount of bias due to data reduction was potentially introduced in the data processing between the second and the third round. From the responses to the second round discussion (8,700 lines), I picked 281 "non-obvious and insightful statements by the panellists", abbreviating/summarising the verbose ones, removing redundancy, and merging closely related statements. At the end of the process, the thousands of lines of text had shrunk to 204 statements. Several sources of bias are inherent in this approach:

1. Again, picking 281 statements from 8,700 lines of text introduced a significant amount of selection bias.
2. By abbreviating and summarising paragraphs into single statements, I potentially slanted the messages of these, and may not have properly represented the participants' views in the shortened versions.
3. The act of merging closely-related statements depends on the definition and perception of what "closely-related" means, and also involves a small amount of reformulation of sentences.

Even though I consulted with various colleagues during the collation and grouping to minimise the bias, I did not have a second pair of eyes available for the reduction process.

Direct discussions with participants Another source of bias may have been my follow-up questions to participants, in case I needed further specification of their statements, or something was unclear. I saw three problematic aspects in follow-up discussions:

1. Since I followed-up only in cases when a given statement was unclear *to me*, I may have missed subtle unclarities in some places, while digging to deep for details in others.
2. In following-up, I did not adequately guard against asking suggestive questions and may have influenced the respondents' answer(s).
3. Finally, as an active member of the Debian Project myself, and often an early adopter, asking for clarification of a given statement may have sent the wrong message to the peer, who might have thought I was arguing and hence felt forced into a defending/opposing position. Similarly, those participants aware of my take on certain tools or techniques outside of the study might have catered their responses to me as the (only non-anonymous) reader.

Unfortunately, the first two issues could only have been properly addressed through a supervisory board, which was infeasible due to the volume of data, and would have prolonged the study substantially.

The last issue is one without a solution, other than anonymously conducting the study, or conducting such a study in a social setting without any self-involvement. However, this would have voided all the positive impacts of my own involvement, such as the ability to go in-depth, a base-level of dedication among the participants that translated into enough patience to put up with the unexpectedly large amount of text to process in the second Delphi round, and the long time it took me to prepare the third discussion round (*cf.* section 8.4.1).

It could be interesting to investigate the effects of using a combination of a wide-band Delphi approach (see section 5.3) and an anonymising medium or tool on the study and its results, although I expect that removing the controlled feedback component (the facilitator) could yield diminishing returns, as discussion threads explode and grow out of sequence (*cf.* section 5.2.1).

8.3.1.3. Pro-innovation bias

Even though I prompted participants to consider influences to the decisions to adopt *or reject* innovations, the primary focus throughout was on successful adoptions. As such, my study is subject to a certain degree of pro-innovation bias [Rogers, 2003, p. 106f]. In particular, within

the stage model I used to structure the results (see section 7.1), I investigated only delays to adopt, but not instances of adopters losing interest, deciding not to adopt, failing to implement, or discontinuing the use of a tool or technique post-adoption.

Even though I can enumerate instances of such rejections, before as well as following adoption decisions, in the Debian Project (*e.g.* dbs, yada, and GNU arch), investigation into these directions would have blown up the scope of this work. I have thus removed rejections from my research question.

8.3.1.4. Researcher-imposed evaluative criteria

Kearns [1992] has criticised innovation diffusion studies for two shortcomings:

1. researchers commonly decided on a framework *a priori* and hence "force fit" data into criteria, which were not grounded in the "underlying assumptions, values, and belief systems of decision makers" (*cf.* the introduction to chapter 7).
2. the evaluative criteria used by most researchers is unable to reflect the actual cognitive properties and processes of decision makers *per se*, and "the deeper cognitive structure of how the decision maker perceives one or more innovations is bound within the artificial confines of language." He suggests the we need "a deeper understanding of how these comparative judgements are made in order to get beyond the boundaries of language and labels and into the deeper structure of decision making."

While I avoided the first of these shortcomings by letting the structure presented in section 7.1 emerge from the data, I chose labels for each of the 24 influences (and in round two, I attached keywords to groups of statements), which were not grounded in the "underlying assumptions, values, and belief systems of decision makers" nor allowed access to the "deeper cognitive structure of how the decision maker perceives one or more innovations." Kearns puts it succinctly:

> We have been exhorted to refine the operational definitions and measures of these researcher-imposed evaluative criteria [Perry and Kraemer, 1978, Tornatzky and Klein, 1982] when we do not even know if these criteria would be mentioned by decision makers if given the chance to do so in an environment uncontaminated by the researcher's own theoretical framework.

In the light of this captivating quote, the results of the study stand, but the labels of each of the 24 influences is questionable, and it might be worth discovering how these labels influenced the participants' responses in the third discussion round.

Kearns himself conducted a study with which he sought to overcome these limitations. Unfortunately, I only came across his work after the third round of the Delphi discussion had commenced. The approach builds on the "innovation grid (INGRID) protocol", which is based on the psychology of personal constructs [Kelly, 1955]. Rogers [2003, p. 226] describes the approach as follows:

> The name of each of the eight innovations of study was written on a 3-by-5 inch card, together with a one-sentence description of each innovation. Each of the 127 respondents was handed three of the innovation cards and asked, "Can you think of any way in which two of these innovations are alike, and different from the third?"

This would have been a fantastic approach to explore influences (or attributes). First, the question is somewhat closed and automatically frames the context; this facilitates participation and allows respondents to focus better on the task, as opposed to the open-ended question I asked in the first Delphi round (see section 6.2.1). Second, the approach enables the researcher to use terminology grounded in the participant's perspectives in later phases of surveying or research.

The approach also provides room for variation. For instance, it might suffice to ask for similarity between two innovations, but that would have to be investigated. Unfortunately, Kearns does not clearly argue for why he chose his exact approach, rather than a simpler one.

The problem was also identified by a few participants during the study, *e.g.*:

> *Maybe a second-and-a-half round for proposing the keywords could have avoided misunderstanding and perceived redundancy. One of the things that took time was,* e.g.*, to try to understand the difference between peercolation and network effects.*
>
> — Charles Plessy, feedback round <20090813043815.GD13031@kunpuu.plessy.org>

I had never considered letting the participants provide the influence names. I investigated numerous approaches of categorisation (see appendix C.1), but to let the participants label the short paragraphs presented in the third round never crossed my mind.

8.3.2. Design issues

The study I conducted yielded valuable results, but a number of problems became apparent as time went on. To aid future researchers, I would like to address these problems in turn.

8.3.2.1. Panelist sampling

I treated bias issues during the sampling phase already in section 8.3.1.1.

The most important part of the panelist sampling process was the stratified purposeful selection of candidates along four strata. While the four strata appeared sensible (see section 5.4.3), and I could not identify additional or different ones during the study, I have not analysed the degree of diversity they are able to map. It could be that a different set, or additional strata, could have increased the diversity.

Instead of asking for free-form answers and later coding the applicants' positions in the 4D-hypercube, it might have been more sensible to let the panellists do the positioning themselves, *e.g.* on a scale from 1 to 10 (which is what I used, effectively).

Furthermore, given the codings from each applicant, I could have used a more rigorous method to identify the 16 candidates with greatest diversity between them, *e.g.* selecting for the maximum average distance between vectors. Due to the qualitative and inherently interpretive nature of the data, such an approach would probably be overkill though.

I asked a number of questions about the panellists' involvement in the Debian Project, such as what kind of tasks occupy which percentage of their time. While the data collected in this way could have been used for selection, I did not pay considerable attention to them.

Finally, it might have been useful to have had concise, detailed, and accurate information about how the study will progress available at the time of application solicitation. However, none of the participants seemed to want this information; instead, a few commented on the usefulness of the level of detail I did provide.

8.3.2.2. The first round

The first round laid the foundation for the vast amount of data the study generated. Its design was therefore responsible for a number of other problems throughout. I could identify two problematic aspects of how I conducted the initial Delphi round, both of which were responsible for the immense volume of data.

First, I followed the advice by Schmidt [1997] and asked the participants to provide "at least six" influences along with explanations, to prevent replies that were too short to be useful. Had I anticipated the amount of data the participants generated, and how the body of data grew in subsequent rounds, a lower expectation (*e.g.* 3) or a maximum limit may have helped contain the volume.

The second aspect lies in the question itself, which sought (a) soft, subjective data, (b) from 21 different individuals, who were (c) asked to speculate across a very diverse and large group of people. As one participant put it:

> *The topic with which the study started was very broad, which is necessarily going to generate a lot of information. A narrower initial topic with a more structured reply format would have reduced the amount of data, but would have produced less interesting general discussion.*
>
> — Russ Allbery, feedback round <87y6r4vyrc.fsf@windlord.stanford.edu>

If I were to repeat the Delphi study, I would build my approach on the experiment conducted by Kearns [1992] and ask the participants to identify ways in which two innovations compare, rather than to identify influences "out of the blue", as I did in the first round of my research. As mentioned before, this idea did not occur to me before coming across Kearns' work in the middle of the third round of the Delphi study.

8.3.2.3. Round two

In a typical Delphi study, the first round is exploratory (open-ended questions), while the aim of subsequent rounds is to home in on the data, reduce their volume, and build consensus [Hung et al., 2008, Keeney et al., 2006]. This may work well in a previously-explored domain, where data can be imported from the literature, but in completely unexplored territory, more than one round was needed to give the participants a chance to reflect upon what the others panellists had to say, and to refine their answers, or provide additional information.

One of the strong arguments for using the Delphi method for this study was the controlled feedback component (see section 5.1) and that called for collating and anonymising the data before providing it to the panel. Initially, my plan had been to summarise and anonymise, obtain an okay from each participant that his/her arguments were correctly represented in the results, and finally collate and return the result. However, the data that came from the first round replies was so multifarious that any attempts to summarise would have chastised the data, and there were countless, in part subtle overlaps between different statements by the same individual as well as across the entire set, that I saw a need to explore these overlaps in an attempt to canvas clearer distinctions between the influences.

I chose to anonymise and collate, but without the summarising. This proved somewhat fatal to the volume of data sent out for the second round. My initial time estimates for participant involvement per round was 1-2 hours, but several participants reported taking up to 10 hours to wade through the details, in a way that approached haphazard for some:

> *Unfortunately, this message is too long to fit in the RAM of my brain, so I can only process it in a streaming way, commenting as I read it. I would like to be able to give comments that interrelate various bits of it, but that would require either rereading it all so many times, or printing it out and scattering it over a large surface; I'm sorry that I do not have time to do any of that, because it could have been very interesting.*
>
> *Reading the mail feels a bit like reading Joyce or Proust: a stream of consciousness about various aspects of software adoption. It does have the quality of being a very, very insightful stream of consciousness: if it were for me, it should be published as it is and be used as an initiation ritual for developers to meditate on, before writing new software. I'm quite sure it would have helped me.*
>
> — Enrico Zini, round two <20081205140625.GA22710@enricozini.org>

One participant later suggested that I should have sent a "partial data set to each participant while still ensuring that each point of view is reviewed by at least 4–5 persons" [Raphaël Hertzog, feedback round[4]], but that would have undermined the benefits of the panel diversity. Possibly a *larger* panel, in combination with more restrictive data collection in the first round could make this a viable approach, but restricting the data was not considered a good objective, either [Neil Williams, feedback round[5]].

The bottom line is that the success of round two was mostly a result of the panellists' dedication to the study, and their high level of interest in the subject matter and the data:

[4] <20090622173622.GB22895@rivendell>
[5] <20090622133137.9a0c6469.codehelp@debian.org>

> *Even if it took more than an hour, it was a pleasure to do it because it gave insight. I think that spending time is not an issue as long as it is useful, fun or otherwise rewarding.*
>
> — Enrico Zini, the feedback round <20090812152342.GA23848@enricozini.org>

In some ways, I knew I could count on this level of dedication from the panellists (*cf.* section 8.4.1), many of whom I knew personally, but that was a shaky expectation, and the study could have failed at this point.

Yet, all participants completed the task of the second round and augmented the data set with refutations, qualifications, and additional information. Even though I had asked people to ensure that I had not dropped any of their arguments, and no-one voiced any concerns in that direction, it is well possible that I left out relevant points which their authors did not miss in the large volume of the data. An intermediate round for this verification may have helped [*cf.* Charles Plessy, feedback round[6]], but would have called for an even larger time investment. I therefore deemed it appropriate to request the panellists to do it all in a single pass: check for omissions, voice disagreement, and simply skip arguments in case of general agreement.

The result was positively received. For instance, one panellist thought the collection was already nearing completion:

> *It's surprising how little there seems to be left to say. I generally agree with most of the collection, and even the places where I disagree, I can see it's usually about personal preferences and not anything that can be judged right or wrong.*
>
> — Niko Tyni, round two <20081130213055.GA19290@rebekka>

At least one participant missed the availability of all data in a different format, such as a cross-linked and searchable reference index, available on-line or as a separate attachment to the second round e-mail [Neil Williams, feedback round[7]].

It is furthermore unknown in what way the grouping of statements into 15 loosely-related categories, which I used to present the data in a more manageable way, have affected the participants' reactions and responses in the second round of the discussion (*cf.* section 8.3.1.4). I encouraged the participants to suggest alternate categorisations, but that was arguably asking too much, given the amount of data; only one person suggested a different approach to categorisation, but I could not implement his suggestion successfully (see appendix C.1.1).

[6] <20090813043815.GD13031@kunpuu.plessy.org>
[7] <20090622133137.9a0c6469.codehelp@debian.org>

Nonetheless, a few participants expressed their disagreement with some of the classifications, but their concerns were not compatible with each other. This was the main reason why I embarked into the third round with a clean slate, rid myself of these categorisations, and formed new groups (the 24 influences) only after processing all the data from round two.

One possible avenue of research I left unexplored was to correlate participants' answers with their positions on the four strata used during panellist selection. Potentially, this would have allowed for greater analysis depth, but the selection data would need to be more rigorously collected for this to make sense. I did not conceive of a way in which such a correlation could have been incorporated into the study, but the idea was there.

8.3.2.4. The final round, #3

With the third round, I sought to reduce the data, similar to what happens already in the second round of a typical Delphi study [Hung et al., 2008, *cf.*], but I did not find a way to achieve that (see appendix C.1). Due to lack of a suitable approach, as well as growing concerns over the completely underestimated time requirements I expected from my panellists, I saw no other way than to reduce the data myself. This brought about a number of issues:

First, the process of preparing the third round took severely longer than anticipated, introducing a delay into the study that made it difficult for some panellists to get back into the required mode of thinking for the third round, or were unable to dedicate as much time to the study as if the schedule had been kept. Given that I could not keep the initial pre-Christian-Christmas schedule, it was beneficial that I considered the releases of Debian "lenny" and Ubuntu 9.04, which took a lot of time from most participants, and only issued round three after both had been published. I return to this topic in section 8.4.3.1.

Bias Second, the third round may be the most significant source of bias of the entire study (see section 8.3.1.2): my autonomous identification of only 281 statements in 8,700 lines of text, and the subsequent merging of statements to yield a final set of 204 most probably purged some valuable insights from the data, despite my best efforts.

In addition, I had to transcribe sets of statements into coherent paragraphs, which may have caused further loss of data, and may have reproduced arguments in inappropriate contexts. One participant spotted a misrepresentation in my transcription of the elegance in-

fluence, which I reported at the beginning of the corresponding discussion ("elegance", section 7.2.2.2).

Grounding The third problem was already subject of section 8.3.1.4: to make better sense of the collection of 24 paragraphs I transcribed, I affixed short labels, which were not necessarily grounded in the data. Furthermore, the distinctions between several of these labels wasn't clear to the participants. For instance, some had a hard time separating "peercolation" from "network effects" (the former is more about trust and information spread, while the latter is more about interdependencies), or "cost-benefit" and "return-on-investment" (the former focuses more on cost, while the latter is mainly about returns).

Potentially, an intermediate round could have helped find proper labels for the transcribed paragraphs [*cf.* Charles Plessy, feedback round[8]], but I considered it doubtful that a single such round would have yielded useful results, and did not want to impose additional burden on the panellists, possibly risking their loss of interest in the study with further exploding time expectancies. In hindsight, however, such a labelling exercise would have been a better goal of the third round; I will return to that aspect later in this section.

Presentation of data The amount of data was still too high, which was the fourth problem of this round. As one participant put it, "prioritizing a large list of items is difficult without some structure" [Russ Allbery, feedback round[9]], and I had no suitable structure at the time, forcing me to present the 24 influences in random order instead. It could have been beneficial if I had waited a bit after reducing the data to let the cycle presented in section 7.1 emerge, which would have allowed me to present the influences in temporal order, but as I had already incurred a large delay, I went ahead with the study.

Inappropriate medium Furthermore, the e-mail medium proved inadequate for the task of identifying the three most salient influences, as participants sought a more interactive and iterative way to rank and compare influences with each other, because the approach I proposed did not yield satisfactory results:

[8] <20090813043815.GD13031@kunpuu.plessy.org>
[9] <87y6r4vyrc.fsf@windlord.stanford.edu>

> *I feel like if I'd answered it on a different day, I could have picked entirely different choices, and been equally as convinced they were the right three. (Actually, I did answer it twice on two different days, and changed three out of my six choices!)*
>
> — Joey Hess, round three <20090601170437.GA26955@gnu.kitenet.net>

Suggestions in the feedback round ranged from pairwise comparisons, Likert scales[10], scoring the influences [Loïc Minier, feedback round[11]], formalising relationships between influences [Raphaël Hertzog, feedback round[12]], to a web-based survey tool, described as follows:[13]

> *I used phpESP to create a survey out of the 24 points and included the supporting statements in the questions, with a "rank" or "scale" answer for each question (ranging from 0 to 10, from strongly negative to strongly positive impact of the influence).*
>
> *The survey format really helped by presenting the entire point and all supporting statements on a single screen along with the scale to rate the point for the tool under consideration — it turned out to be quite useful that the survey format did* not *allow textual comments in the form that I used, I had to make a decision on a fixed scale and leave all the "finer points" for later when I could consider just the most important factors.*
>
> — Neil Williams, round three <20090422140835.19af7584.codehelp@debian.org>

I had considered the use of a similar approach — which is not unlike a Likert scale — but could not work around shortcomings, such as the tendency for people to lean towards extremes, or the avoidance of the outer scale items [White and Mackay, 1973].

It turned out to be a good design choice to let people combine influences, rather than asking for three singular ones:

> *What I would have personally hated is the impossibility to express ties or, more precisely, to state that two influences were actually the same. Given that you gave that possibility, I felt no other particular over-constraint.*
>
> — Stefano Zacchiroli, feedback round <20090707155307.GA10704@usha.takhisis.invalid>

The wrong goal All in all, the third round as conducted was considered the least satisfactory round, and a regression in terms of its contribution compared to the previous rounds [Joey

[10] One participant used a Likert-style approach after admitting that he saw no way to identify the three most salient influences [Andreas Tille, round three <alpine.DEB.2.00.0904082240140.21833@wr-linux02>]).

[11] <20090622144632.GA10842@fox.dooz.org>

[12] <20090622173622.GB22895@rivendell>

[13] The source code he used, as well as the full description will be available under a Free licence with the rest of the data. The author kindly made his work available, even though he was not prepared to be asked for it.

Hess, round three[14]]. A different goal would have yielded more value in the data. Considering that I did not make use of the ranking which the panellists performed beyond the short presentation of the influences' frequencies in section 8.1, it would have been better to hand the influence transcriptions to the participants, along with the invitation to invent labels for them. I expect this would have taken less time, and might have made a fourth round possible, in which consensus over which single labels to use in each case was established. I later considered to follow-up along these lines, but discarded the idea because I had already tainted the domain with my own invented labels, and also could not ask for more time from the panellists.

I also did not make use of the participants' top-most salient influences to their own work, as opposed to the influences most affecting the project as a whole. I was not planning to study those individual influences, and my main motivation to explicitly ask for them was to enable the participants to focus on group adoption instead (see section 6.2.6). On the other hand, this incurred work for people who may have expected these data to be used as well. For instance, the following idea was tagged along to a response by a panellist:

> *I may be borrowing trouble here, but I think it'll be extremely interesting if you find that the influences deemed most relevant to the project as a whole by most panellists are not strongly shared by most panellists for themselves. (Would that suggest a non-representative panel, or merely that people naturally believe that "other people" are more susceptible to certain influences than they are themselves?)*
>
> — Colin Watson, round three <20090425005022.GG25892@riva.ucam.org>

The following alternate approach was proposed by the same panellist who suggested the use of searchable, cross-referenced indices (see above), and who used the phpESP-based approach to ranking the influences:

[14] <20090601170437.GA26955@gnu.kitenet.net>

> *What occurred to me whilst preparing the reply was the level of inter-operation between the issues, forming a network of inter-related factors that act with varying amounts of bias and leverage. It's almost biochemical in nature — there are striking similarities with how the brain is affected by chemical, physical and internal factors. It's like a many-pointed seesaw that has to be balanced across all sides and where pressing on any one point has complex effects on all the other points. Predicting the results of pushing any one factor is very, very difficult. In pharmacy, most mental health drugs affect many systems at the same time, with varying levels of potency. It can be like walking a tightrope, trying to balance the 4 effects of drug A against the 3 effects of drug B and the 5 effects of drug C across maybe 7 different sub-systems. I get the same feeling with these — the interplay is fascinating and challenging*
>
> — Neil Williams, round one <1226008184.15670.53.camel@holly.codehelp>

I had experimented with concept-mapping tools at various stages in the study, *e.g.* when preparing the second round (see section 6.2.3), but I did not come across a suitable tool to use in a group. I found Deepamehta,[15] which promises to be a collaborative mind-mapping tool, only towards the end of my study.

8.4. Review of the design

8.4.1. Intuition and self-involvement

Perhaps the most striking aspect of this research goes back to my strong involvement in the Debian Project, and my close contact with the community, including most of the panellists that participated in the Delphi study. In addition, my strong interest in the subject matter, which extends to separate endeavours, such as the vcs-pkg project[16], and which had me publish or present my thoughts in various contexts, gave this research a twist that had both, very positive, but also some negative effects. I have documented these effects more formally during my discussion of the philosophical considerations underlying this research (see section 4.1). The followig are *post-hoc* reflections.

On the positive side, I was in a highly advantageous position to conduct this research. My proficiency with Debian and its community allowed me to explore arguments in depth, and I had access to (and knowledge of) all resources that are in use by DCs in their daily work.

[15] http://www.deepamehta.de/
[16] http://vcs-pkg.org

Furthermore, I was able to make use of my knowledge of the community traits within the project, as well as the individual traits of volunteers, being a volunteer myself, and having worked intensively with many of them for several years. This enabled me to fit the study tightly around the participants' expectations, which led one participant to compare my study to a FLOSS-related survey, to which he had been invited, and which he perceived as highly disappointing and inappropriate: "Compare this approach with your research, it's like two alternate universes." [Enrico Zini, Internet relay chat (IRC) conversation].

Lastly, my close ties with the participants, who perceived me as "one of them", yielded a higher level of dedication to participate, which was

paramount to the study's success. I should not, however, discount the importance of each panellist's interest in the subject matter; several of them have stated at various points how enticing and insightful their participation has been.

However, my involvement with Debian also presented a few obstacles. First and foremost, my interest in the subject, and my previous ventures into the domain considerably undermined my objectivity, both when confering with panellists in follow-up rounds, as well as when interpreting their statements. Even though I made best efforts not to let my interests and opinions get in the way, I cannot qualify the degree to which I succeeded. Nevertheless, I maintain that I sought to explore influences to adoption behaviour, rather than to confirm my own preconceptions, and I am confident I managed to the best of my abilities.

Being an early adopter in the Debian Project who has often published thoughts and opinions in the pertinent forums, and who regularly engages in discussions with peers, my views on topics were no secret to the participants and may have influenced their phrasing, or their statements — after all, the participants were anonymous with respect to each other, but not anonymous towards me. For instance, as a strong supporter of the Git VCS, I felt the need at times to make my position explicit, along with the encouragement not to perceive a need to argue with or against my view.

In addition, my knowledge of the project and its processes may have caused me to forego exploration of topics or disagreements, which appeared too obvious for me to notice.

Finally, as a contributor to the Debian Project for more than 10 years, I possess a conspicuous level of intuition in Debian topics, and volunteer involvement. I have made a few decisions throughout this research, which I cannot scientifically back up, but which I can argue to have been correct. Such decisions shaped the panellist selection process, although I made sure not to succomb to "convenience sampling". Other instances of such decisions took place during the

data analysis and reduction phases, the way I chose to pseudo-personalise e-mails, and the choice of incentives.

Also, in this thesis, I have made claims without referencing sources, when those sources simply did not exist, but the claim was justified by my knowledge of the Debian Project.

As a cultural insider with notable experience in the project, I feel that I was able to discover depths that would have been unreachable in the context of a project I would approach from the outside [*cf.* Mintzberg, 1979]. This is primarily a benefit, and allowed me to produce better results. I chose to minimise the negative effects of my involvement through the meticulous, detailed description of my approach and all assumptions made throughout.

8.4.2. Compensation for participation

As described in section 5.4.9.2, I compensated the panellists for their participation with a choice of one of four gadgets to be given out after completion of the study. My motivation was to reduce the chance of participant dropout, and to give an incentive not to minimise the amount of time of involvement.

The offer was received well throughout, many expressed their surprise, and a few panellists responded in the first round without listing their preferences, which I requested to help me plan the distribution of the gadgets. Five participants (out of 21) explicitly stated that they would participate even without compensation, and one person declined the offer.

Following the study, I asked the participants to reflect on the effect of the compensation on their agreement to participate, as well as on their dedication and time allocation during the study. I received replies from two thirds of the panellists.

While some respondents stated that they experienced difficulty assessing their readiness to participate without compensation in the aftermath, all respondents claimed that they would have probably participated even without the offer of compensation, but many would have had to think twice about it.

The offer seemed to have three effects:

- it underlined the seriousness of the research;
- it conveyed the feeling that participation and time investment were valued;

- it helped make the decision.

One participant felt that the perceived (high) value of the compensation was appropriate (even though not in relation to the time expenditure), and that a lower value would have potentially been worse than no compensation whatsoever, because it would have conveyed a low estimation of value of participation. Not being compensated may have elicited different levels of commitment.

Throughout the study, the compensation helped and motivated some panellists to continue investing time and effort into the study, up to the point where they felt an obligation to participate through the end to earn the reward. Some participants, on the other hand, claimed that once they had committed to the study, the compensation became secondary to their involvement.

A few participants noted that the compensation was not the only reward they perceived. Other incentives included interest in the subject matter and the Delphi approach, the prospect of being part of an effort that could potentially be of benefit to the project, the chance to exchange opinions with other domain experts, and the satisfaction from investigating a challenging topic.

A few comments suggested that their participation was influenced by the study being conducted by an active contributor to the project, and that they might not have been participated had an external researcher been behind this, even when offered compensation. One participant even went as far as claiming that compensation offered by a stranger might have led him to be more sceptical.

In summary, the compensation was adequate. Even though it would be possible to conduct such a study without rewards, ensuring commitment throughout may be difficult or impossible. The offer to compensate sent a positive message that was well received by the participants, but it was not the only factor responsible for the success of the study. General interest in the subject, as well as the researcher's affiliation seemed to play a role as well.

8.4.3. Feedback from the panellists

According to Presser and Blair [1994], it is important and often foregone to seek feedback on a study from the participants *post-hoc*. The authors only mention feedback to survey questions to give the researcher a feel of how the questions were received, but since my research was not based on questions *per se*, and part of my contribution is the evaluation of the Delphi

method in a FLOSS context, I asked the panellists to comment on a number of aspects of the study.

The study was positively received throughout. The participants thought I conducted the study very well [Colin Watson, feedback round[17]], were impressed with the data and results [Niko Tyni, round three[18]], and appreciated my efforts [Russ Allbery, feedback round[19]]. They considered their participation "fairly comfortable" [Christian Perrier, round one[20]], and felt that I was doing them a service [Enrico Zini, IRC conversation]. Furthermore, it enabled some to clarify their own perceptions of their work [*e.g.* Jonas Smedegaard, feedback round[21]].

In addition to such positive remarks, the feedback round also brought out a number of concerns, but also confirmations of design choices I had made. These I present in the following sections.

8.4.3.1. Deadlines and delays

During the sampling process, the deadlines I set for responses were too far away. It was good that I sent reminders before the deadlines were reached, as people had shelved and forgotten my call for participation and were thankful for being prodded. Similarly, it was beneficial that I accounted for late responses and did not enforce hard deadlines.

The long delay of the third round posed difficulties for some to refocus [Scott Kitterman, feedback round[22]], and may not have been motivating: "I was still keen on completing the study, the long delay just meant that I was less keen, and it had lower priority" [James Westby, feedback round[23]]. Contrariwise, one participant appreciated the distance, which allowed him to approach the topic anew [Luca Capello, feedback round[24]].

Furthermore, the delay caused conflicts with real life scheduling for some, who would have been able to dedicate more time, had I kept the schedule [Steve Langasek, feedback round[25]].

On the other hand, some participants felt that the first two rounds were too quick and the short deadlines started to impinge on other commitments, including commitments possibly related

[17] <20090820112041.GC16966@riva.ucam.org>
[18] <20090420113239.GA25346@kuusama.it.helsinki.fi>
[19] <87ws99wql2.fsf@windlord.stanford.edu>
[20] <20081107193431.GQ4145@mykerinos.kheops.frmug.org>
[21] <20090619095702.GB20414@jones.dk>
[22] <200906190924.00705.scott@kitterman.com>
[23] <1245707278.6051.42.camel@flash>
[24] <87fxbzbh69.fsf@gismo.pca.it>
[25] <20090810164234.GH18344@dario.dodds.net>

to the pre-Christian-Christmas period; the long delay of round three was perceived in part as beneficial [Neil Williams, feedback round[26]]. Others welcomed the break, which allowed them to concentrate on the Debian "lenny" and Ubuntu 9.04 releases that happened between February and April 2009.

Some participants denied any negative impact of the delays, because of the asynchronous communication, and the nature of cooperation in the Debian Project, which tends to come in bursts anyway [Enrico Zini, feedback round[27]]. One concern was that *e.g.* debhelper7 had been improved meanwhile, rendering some of its criticism in the first two rounds obsolete, but otherwise, change in Debian is not so fast-paced as to affect the validity of the research [Stefano Zacchiroli, feedback round[28]].

In general, a different time of the year, and a more relaxed schedule, which would have included enough time for the evaluation and preparation of rounds, might have been better. Unfortunately, I had no way to foresee the time requirement (and thus delays, and the best choice of deadlines) *a priori*, due to lack of experience with and information on the approach.

8.4.3.2. Choice of medium

I chose to conduct the study using only e-mail as a communication medium, for reasons which I have articulated in section 5.4.6 — particularly its asynchronous nature, as well as the level of experience among the participants to deal with large volumes of text using their own, familiar tools. The participants unanimously agreed that this choice was adequate, and that any other medium would have lowered their motivation, or would have otherwise had a negative impact on their participation.

The possibility of using a web application for the task was discounted by a number of panellists, for example:

[26] <20090622133137.9a0c6469.codehelp@debian.org>
[27] <20090812152342.GA23848@enricozini.org>
[28] <20090707155307.GA10704@usha.takhisis.invalid>

> *I do not like web application, mostly because when you start something you are in some sense obliged to finish it before a certain period of time expires (usually minutes/hours). The other fact is that I am someone who tend to remember what I write in e-mails and during the study I even went back to check what I wrote in previous answers. Web applications usually are intended with a sort of direction (yes, there is a back button, but does it really works as expected with forms & Co.?), while e-mail give you as much freedom as you want.*
>
> — Luca Capello, feedback round <87fxbzbh69.fsf@gismo.pca.it>

Similarly, face-to-face interviews, or the use of a synchronous medium, such as IRC, would have not been well accepted:

> *Having to be present at a particular time to interact with the study would probably have meant that I couldn't participate. An asynchronous method let me respond when I had the time to do the topic justice and meant I didn't have to block out specific time to a schedule, which would have been more difficult. I much prefer being able to reply when I have time.*
>
> — Russ Allbery, feedback round <87y6r4vyrc.fsf@windlord.stanford.edu>

> *I think a synchronous communication medium would have required me to be less reflective in my answers, having a negative impact on the quality of my contributions.*
>
> — Steve Langasek, feedback round <20090810164234.GH18344@dario.dodds.net>

Aside from the ability to work off-line and within the domains of one's own tools, the choice of e-mail as a medium also allowed for deeper reflection in the answers.

> *E-mail was certainly the best choice for me from the perspective of providing comprehensive feedback. It provided opportunity for reflection, re-reading, taking notes, and then turning those into a more detailed response. However, it does mean that my response was not very structured and probably diverged into information that wasn't directly relevant to the goal of your study. A more structured mechanism may have kept my response more focused on the goals of the study and given you less text to sort through.*
>
> — Russ Allbery, feedback round <87y6r4vyrc.fsf@windlord.stanford.edu>

The volume of data could have posed significant problems for those with less experience with high volume communication, but it is questionable whether another medium would have been able to address that.

> *I suspect that someone less familiar with e-mail may have run into tool problems with the amount of text involved and the editing requirements, but I'm not sure there's another medium that would have avoided this problem. My impression is that people with little familiarity with e-mail often also don't have good tools for dealing with response to large amounts of text.*
>
> — Jonas Smedegaard, feedback round <20090619095702.GB20414@jones.dk>

As the study progressed, additional resources or media could have been worthwhile to allow the participants to look up or explore previous arguments:

> *I could have easily discounted the entire study and decided not to participate if it had been promoted as a "web survey", especially as the subject material clearly indicated that a proper discussion would involve a significant amount of time and preparation. My free time, like most volunteers, does not happen in predictable slots. Asynchronous was more important to me than whether it was HTTP or e-mail.*
>
> *However, once the study started, some form of web-based reference/index was clearly missing. A division was necessary between the means of referring back to previous data and answering the current data. Doing all of that within the e-mail client was a problem for me. I usually had to load the text of an e-mail in a different application (via a gateway like MhonArc for conversion to HTML). The length of the study also impacted upon this issue because even using a dedicated e-mail folder, locating which e-mail contained the relevant discussion was not simple, hence conversion to HTML which was easier to search with grep, especially when the HTML contained the usual links within threads.*
>
> — Neil Williams, feedback round <20090622133137.9a0c6469.codehelp@debian.org>

Finally, while [Brehm, 1994] highlights the importance of the initial contact, and Oppenheim [2001] suggests to keep initial contact short, the e-mails I sent to the panellists at the various times were quite long, because I wanted to make the process as transparent as possible, which I assumed would be preferred by the recipients, who were experienced in dealing with large volumes of data. Two participants explicitly expressed their gratitude over the amount of details I provided.

In summary, while e-mail was the best choice of medium, future studies should augment the main discourse over e-mail with a persistent data store, which could be web-based and thereby profit from features such as searching, cross-linking, and in the case of a Wiki, even collaboration, though some sort of moderation of edits seems to be adviseable.

8.4.3.3. Anonymity

The choice to conduct the study anonymously (see section 5.4.5) was received well throughout the panel, and considered a benefit by everyone. It made participation more appealing [Steve Langasek, feedback round[29]], motivating [Gunnar Wolf, feedback round[30]], gave people more freedom to express their ideas [*e.g.* James Westby, feedback round[31]], enabled to judge statements only based on their merit [Damyan Ivanov, feedback round[32]], helped in staying focused [Russ Allbery, feedback round[33]], and made people consider whether the method should not be used more widely in the Debian Project:

> *Anonymity (and the resulting ban on direct conference with others in Debian who may or may not have been participants) was unusual for discussions around topics that had been previously transparent and open, but it did make me wonder if some of the more transparent and free-for-all discussions that occur within Debian may actually achieve more if the contributors had some kind of acceptable referee and did not receive the replies directly from other contributors. The flame-wars and trolling, the bike-shedding and tangential discussions that can so dominate our public mailing lists can be a huge barrier to actual development. Many contributors and developers opt out of subscription to these mailing lists solely due to these problems. This is an especially difficult issue when certain people decide to use such lists solely as a way of point scoring or "I answered last therefore I win" contests.*
>
> *In these situations* not *receiving the replies to your own posts directly from those who have the strongest opinions on the topics discussed could act as a fire retardant, preventing flame wars. It's not an appealing prospect to be the person in the middle doing the fire-fighting but it is an interesting idea nonetheless. Debian clearly lacks the manpower to use such a method for anything more than the most poisonous and resistant flamers.*
>
> — Scott Kitterman (his emphasis), feedback round
> <200906190924.00705.scott@kitterman.com>

For some, it was initially a bit distractive to be talking to a panel made up of anonymous people they already knew:

[29] <20090810164234.GH18344@dario.dodds.net>
[30] <20090813042842.GB22900@cajita.gateway.2wire.net>
[31] <1245707278.6051.42.camel@flash>
[32] <20090625133022.GP28503@pc1.creditreform.bg>
[33] <87y6r4vyrc.fsf@windlord.stanford.edu>

> *At first I kept imagining who might be participating, and that distracted me some. But slowly I realized the power of* not *competing or assuming certain characteristics of others.*
>
> — Jonas Smedegaard (his emphasis), feedback round <20090619095702.GB20414@jones.dk>

I gave the participants the choice whether their answers would stay anonymous after the study had ended, but everyone gave permission to the publication of non-anonymous data.

> *I think it was good. I felt able to speak more freely, as I didn't have to worry that the other participants wouldn't value my contributions as I am less influential in Debian, and I had to worry less about who I was disagreeing with.*
>
> *While releasing the responses at the end will counteract that somewhat, I found that even the temporary anonymity helped. The fact that anonymity might only be temporary may mean that others would feel more constrained. However, you gave the choice, so if it was an issue for someone they could keep their responses private.*
>
> — James Westby, feedback round <1245707278.6051.42.camel@flash>

8.4.3.4. Question design and answer format

In section 5.4.8, I argued for the use of open-ended questions, mainly because of the need to generate a body of data, due to lack of existing literature on the topic (*cf.* section 8.3.2.3). In addition to open-ended, exploratory questioning, I called for free-form answers, rather than to present multiple-choice answers, Likert scales, or other constraining elements. As the amount of data grew, I started questioning whether my initial choice was appropriate. The subject was very broad, so a large volume was to be expected, but perhaps other answer formats could have helped with data management [*cf.* Loïc Minier, feedback round[34]].

The panellists, however, were not in favour of the idea of limiting the response format, seeing value in the ability to write free-form:

> *No, not really. Not without making the research potentially much dumber, I mean. I think the value was exactly in giving people the possibility of writing free-form answers, and in you sweating your way through all of them to find the gems.*
>
> — Enrico Zini, feedback round <20090812152342.GA23848@enricozini.org>

Most thought that any multiple-choice approach would have limited the value of the study, influenced the answers [Charles Plessy, feedback round[35]], and introduced bias [Scott Kitterman,

[34] <20090619075557.GA13724@fox.dooz.org>
[35] <20090813043815.GD13031@kunpuu.plessy.org>

feedback round[36]]. In addition, one participant noted that limiting answers can be demotivating and impersonal:

> *Asking for free-form answers stimulates participants to reflect on the questions. I find that whenever answering is limited, I tend to be less interested because it does not call for what I am actually thinking.*
>
> — Luca Capello, feedback round <87fxbzbh69.fsf@gismo.pca.it>

A few participants had some interesting suggestions, which may be worth pursuing. For instance, asking participants to limit the *length* of their responses [Scott Kitterman, feedback round[37]], *e.g.* to a number of paragraphs/words/characters can help to keep the volume low. However, this would impose the extra burden on participants to write or edit for conciseness, which might take up significant amounts of time, could be hard or impossible for those not gifted with writing skills, or simply yield less considered or incomplete answers. Compared to shortening by the facilitator, this approach would better ensure that a participant's views would be correctly represented in the shorter text.

Another participant highlighted the need for (and lack of) condensation of the data as the study progressed:

> *In the early stages, it would have severely hindered the study to have limited the possible responses. At the final review stage, it might have been beneficial, as long as the possible responses were on a suitably wide scale and not Yes/No answers and had room for freeform text as well.*
>
> — Neil Williams, feedback round <20090622133137.9a0c6469.codehelp@debian.org>

8.5. Recommendations for future research

Out of this research emerged a number of questions and possible avenues of future directions with immediate relevance to my research. I discuss them in the following paragraphs, and highlight how their investigation would further the understanding of adoption behaviour in the Debian Project, and FLOSS.

First and foremost, from an academic perspective, it could be insightful to compare the results from this research to those of similar research in other FLOSS projects. Even though Debian is unique in many ways, it probably shares more traits with other FLOSS projects than there are differences, at least when compared to traditional, for-profit software engineering, and other

[36] <200906190924.00705.scott@kitterman.com>
[37] <200906190924.00705.scott@kitterman.com>

fields in which diffusions of innovations have been studied. I hope that my work will enable future researchers to investigate technology adoption in other FLOSS projects, as well as other aspects of the spread, sedimentation, and implementation of ideas. I also hope that I have established the Delphi method as a suitable tool for such tasks, and facilitated its use through the meticulous documentation in chapter 5.

8.5.1. Comprehensiveness and salience of influences

The 24 influences presented in section 7.2 were iteratively developed, but there is no guarantee that those 24 influences are comprehensive and account for all factors affecting adoption behaviour. Indeed, it seems sensible to assume that there are other influences at work, which we did not discover. At the same time, it would be wrong to attribute equal weight to the 24 influences; instead, one ought to expect a subset of these influences to be *pro rata* more salient than the whole set. The determination of the most salient influences was in conflict with the diversity of the assembled panel. Even though a few influences stood out (see section 8.1), that result could not be seen as representative of the project.

A palpable continuation of my work could set out to obtain quantitative data on adoptions, of which next to none exist at time of writing. A page on Debian's wiki[38] refers to numerous sites collecting and presenting statistics, and Debian's "popularity contest[39]" gives a very rough idea of number of installations of packages.[40] Such data could be surveyed at the scale of the whole project, *e.g.* with a Likert-scale approach that allowed write-ins, and possible comments. To obtain representative data, a lottery could be used as an incentive, and participants could be asked to classify themselves on a number of scales, similar to the approach I described in section 6.1.5.

On the other hand, richer results could probably be obtained on smaller scales, because the entirety of the Debian Project encompasses such rich diversity that the identification of a single adoption behaviour could well prove to be an unsurmountable task. There exist countless subgroups of different sizes in the project whose members might behave more uniformly when it comes to change. It could be beneficial to study some of those groups in isolation, and later to compare finding and attempt to correlate them with facets of the group, such as size, coordination, goals, and type of work.

[38] `http://wiki.debian.org/Statistics` [6 Aug 2009]
[39] `http://popcon.debian.org`
[40] I have been collecting daily and weekly snapshots of these data for years, but the numbers are not authoritative, since participation is voluntary, and installations do not correspond directly to users.

A potentially interesting line of research could involve a questionaire approach to Debian sub-projects, and other FLOSS projects. The questionaire would explain the influences described in section 7.2 and inquire about the effects of each in the context of the target group.

Additional insights could be gained from analysing and attempting to explain past diffusions with the help of the adoption stage model and the set of influences described in.

Furthermore, a collaborative approach to collecting data about past and present instances of any of the 24 influences would visibly enrich the data and pave the way for further refinement.

Finally, the approach by Kearns [1992] across a number of longitudinal diffusion studies to determine how much variance in adoption rate can be accounted for by the 24 influences, and which subset of influences can account for most of the variance could provide valuable information about the salience of the individual influences. For instance, if one were to find that the 24 influences account for 90% of the variance in adoption rate, but 6 of these influences alone account for 85% of the variance, then that would tell us two things: those 6 influences are a lot more salient than the remaining 12, and that additional influences exist, which are responsible for the remaining 10% in variance.

8.5.2. Longitudinal diffusion studies

Longitudinal studies of individual diffusions could yield more reliable results. The so-called "recall problem" is a significant problem with diffusion research, which relies on the subjects' abilities to recall an event or a decision in the past. Rogers [2003, p. 127] suggests that "diffusion studies should rely on 'moving pictures' of behavior, rather than on 'snapshots', because of the need to trace the sequential flow of an innovation as it spreads through a social system [rather than] freezing a continuous process over time."

Rogers suggests four alternative approaches: (1) field experiments, (2) longitudinal panel studies, (3) use of archival records, and (4) case studies of the innovation process with data from multiple respondents.

A research endeavour, that focused on a single innovation from as early as possible and accompanied its diffusion could provide ground-breaking insights into the organisational adoption process in FLOSS. At the moment, some possible projects can be found in Debian in the domains

of distributed version control system (DVCS) and specific workflows for packaging[41], new source package formats (*e.g.* dpkgv3), and debhelper7.

8.5.3. Executive decision-making

Another fruitful, quantitative endeavour could seek to investigate the degree to which project members would like to follow technical decisions made in a executive fashion, *e.g.* by the technical committee. In addition, such a survey could query for the conditions under which such decisions would be respected, and where the limit lies before the authority would be undermined. Debian has traditionally been very libertarian, but executive decisions have happened before, albeit less so at the project-level than in sub-projects. How could executive decisions be used to enable progress without alienating project members?

8.5.4. Consequences of innovation

My research identified the salient influences to adoption behaviour. Even though I tried to assess the *status quo*, rather than focus exclusively on past diffusions, I did not specifically focus on studying the consequences of innovations.

While Rogers [2003, p. 440] highlights that past diffusion research often ends with the analysis of the decision to adopt, the implementation and other consequences of that decision are not studied. In my study, I included *e.g.* implementation and later incorporation (including standardisation), which are certainly consequences, but I did not look into or assess the consequences of major innovations or new concepts in the Debian Project, such as the adoption of patch management systems for packaging, or the effect of DVCSs on collaboration.

8.5.5. Investigation of single stages

Several stages in the combined innovation-decision stage model developed in section 7.1 are broad enough to warrant closer inspection. In section 7.2, I have laid out influences along the stages, and focusing on specific stages may produce more, granular knowledge about the influences and how they affect the behaviour of adopters who are passing a certain stage. The stages of particular interest may be:

[41] *cf.* `http://vcs-pkg.org`

Individual persuasion — How are opinions formed? What subjective factors play a role?

Individual implementation — How are innovations put to use, and what is the relationship between re-invention and minimising divergence from the original tool (*cf.* section 2.2.4.5).

Incorporation — How are standards identified, created, defined, documented, spread, and policed?

Organisational initiation — Can organisational initiation be identified? Can it be bootstrapped, or influenced?

Another unexplored research direction is critical mass in a FLOSS project, and the interplay between criticial mass and consensus (*cf.* "consensus", section 7.2.6.1). While it is often impossible to forecast critical mass [*cf.* Gladwell, 2002], the identification of factors that play into the creation of critical mass could have important impact on the process by which concensus can be built, as well as the importance of consensus *per se*. For instance, how does meritocracy relate to critical mass? Does the average reputation of an adopter play a role, and could 50 high-profile adopters generate the same momentum as 500 average adopters? How does popularity factor into adoption behaviour at the various stages?

8.5.6. Needs and technology

In the discussion of the knowledge stage (section 7.2.1), I touched the relation of need and availability of a technology ("need-pull" and "technology-push"), citing Zmud [1984] on the effect of need and availability coinciding. FLOSS development is strongly anchored in individual "itch-scratching" — individuals create tools and techniques to meet their own needs [*cf.* Raymond, 1999], and this suggests that technology follows needs. However, exposure to a new technology (*e.g.* social networking, or a patch management system) might also create a need. The causalities may well differ on an organisational level, if one can assume that something like an organisational need can exist.

While I postulate little benefit in knowledge of what tends to come first, a need or a technology, I think it would be worthwhile to study differences in adoption behaviour in both situations. If a technology appears to meet a need, I expect such influences as sedimentation (section 7.2.1.1) to be a lot less salient, than if a technology appeared for which the exposed individual did not perceive a need ("selective perception"). Conversely, if a need exists, then examples, templates and tutorial-like documentation ("quality documentation" and "examples *vs.* documentation" in sections 7.2.3.2 and 7.2.3.1) might be more important in helping address the needs more

efficiently, and the importance of marketing (section 7.2.1.3) or peer influence ("peercolation", section 7.2.1.4) might decrease.

8.5.7. Information flow and channel effectiveness

It could be very interesting to study the flow of information in the Debian Project, as well as the effectiveness of the communication channels. Rogers [2003, p. 303ff] presents a number of different models of communication flow, alongside studies that investigated how the percolation of information through social networks depends on the channel by which the information reached the social system. As part of the discussion on marketing in section 7.2.1.3, I presented various views on web-logs in Debian: they are better suited for announcements, but only read by a proportion of DCs ("Camel advertising in Camel cigarette packs"), and their effectiveness might fall prey to increasing popularity, similar to the fate of central, public mailing lists, which are flooded with traffic.

The following questions appear relevant and worthwhile:

- What are the traits of these channels pertaining to the content exchanged on them?
- How do topics move between channels (*cf.* "earliness of knowing" [Rogers, 2003, p. 94f])?
- Are there correlations between developer traits and the channels they use?
- Do certain channels appear to be more effective in spreading information that others?
- Are there channels more conducive to building consensus?
- What are the post-transmission effects of messages?
- How do the individual channels relate to the sedimentation of ideas (see section 7.2.1.1)?

The work of Kevin Crowston and his colleagues and students at Syracuse University, New York, USA, may be of relevance for research endeavours into this direction.

Another point of focus could be tools like `lintian` (automated checkers) and their role in spreading information. What percentage of knowledge about innovations is spread simply because they have become standard, are now checked by such tools, and thus brought to individuals' attentions? What are the effects of changes to standards documentation on regular project contributors? Knowledge of the effects and effectiveness of standards documents, and automated quality assurance tools could help improve their use in diffusing innovations.

Furthermore, the analysis of diffusion networks [*cf.* Rogers, 2003, p. 99] in a FLOSS project could produce useful insights. Luciana Fujii is working on mapping the interpersonal relationships within Debian. Related topics are opinion leaderships [*ibid.*], as well as cluster and clique analysis within such relationship networks.

8.5.8. Revolutions and evolutions

The discussion on chunking in section 7.2.1.2 was about revolutions and evolutions: rather than expecting people to revolutionise the way they work, smaller, evolutionary steps are more compatible (*cf.* section 7.2.3.4). However, it may not always be possible to convert a revolution into a series of evolutionary steps, and doing so will require a significant amount of time investment so early in the process that it is hard to judge the worth of doing so. Also, it is not always possible to identify revolutions up front.

Maybe an investigation of existing instances of when revolutions were (or were not) converted into evolutionary steps could shed more light onto the issue and provide reliable criteria to help identifying revolutions, and chunking them up without much time investment.

8.5.9. Corporate affiliation and credibility

While looking at the credibility of peers ("peercolation", section 7.2.1.4), I hinted at the possibility of corporate links affecting credibility, as well as the reputation of tools that have corporate origin. While researching this point, I could not establish any direct relation, and most people I talked to denied any such influence. And yet, corporate affiliation enters some discussions, especially those about Git and bzr. The following statement stems from a panellist, who is a Canonical employee:

> *Scott's rant[42] can't have been what I was thinking of since it was written after I wrote the comment above, and it was more of a GNOME thing anyway, but it was quite characteristic of a number of exchanges on Planet Debian at the time: somebody posts with (*e.g.*) a complaint about Git's user interface, somebody else responds with how they're only saying that because they work for Canonical or similar, and it all gets worse from there on in.*

[42] `http://www.netsplit.com/2009/02/17/git-sucks-2/` [21 Aug 2009]

> *I remember Ian Jackson making a point of not talking much about Bazaar (or even all that much about Ubuntu) until he left Canonical, because he felt he was likely to be accused of being a corporate shill otherwise.*
>
> — Colin Watson, post-study follow-up to round two statement
> <20090820132553.GD16966@riva.ucam.org>

Previous work into this direction include Robles et al. [2007], and Stürmer [2009].

8.5.10. Elegance

Elegance is undoubtedly a highly-subjective notion. Nevertheless, as suggested in the discussion of elegance as an influence section 7.2.2.2, as well as in the discussion of numerous other influences (*e.g.* "consensus" and "standards" in sections 7.2.6.1 and 7.2.8.1), a group notion of elegance seems to persist in the Debian Project. It could be highly beneficial to inventors and diffusers in the Debian Project to know what notions of elegance have become accepted at the project level. Furthermore, investigation into the evolution of these notions could reveal insightful patterns and augment the knowledge on the influences of the individual implementation stage (see stage 4, section 7.2.4), which deal with flexibility, as well as aforementioned influences consensus and standards. In this context, the claim by a panellist during the study on the relation between elegance and familiarity comes to mind [Colin Watson, round three[43]].

8.5.11. Level of abstraction

In the discussion on modularity and transparency in sections 7.2.4.1 and 7.2.4.2, I explored the spectrum between highly granular solutions (*e.g.* debhelper), and rather monolithic approaches (*e.g.* Common Debian Build System (CDBS)), and noted how both sides are tending away from the extreme positions towards the middle. This calls for further investigation for the right level of abstraction in a project with a goal similar to Debian's: granularity provides for flexibility, but also incubates problems when trying to implement change across large parts of the Debian archive, when changes have to be made in countless places, instead of a single, central location.

debhelper is already an abstraction of underlying, basic Unix commands, and it is accepted by the mostpart as a base-line beyond which understanding and control is deemed secondary. Which amount of further abstraction is possible and desirable? How do abstraction and control interact

[43] <20090425005022.GG25892@riva.ucam.org>

in other parts of the project and aspects of Debian packaging work? For instance, package scripts are still very granular, whereas a tool like dpkg-buildpackage automates an elaborate process and is still the standard method to build Debian packages. Other examples exist across the project and its sub-projects, and it might be possibile to find commonalities and carry experience from one domain to another.

8.5.12. On-line diffusion theory

The findings presented in chapter 7, as well as the implications listed in section 8.2 were collected in the Debian Project. In section 8.1.3, I analysed the specificity of the individual influences to Debian, and postulated ways in which each influence would be applicable other FLOSS projects. The extrapolation of my findings to other on-line, virtual communities could potentially constitute a contribution of on-line diffusion theory.

Pitfall	Remedy
Unclear goal	Be specific about the goal of the study from the start. Know what you are aiming for, and let the participants know.
Incomplete plan	Plan the study to the end, and consider informing participants up front about the input and output to expect to/from each round.
Non-strict timeline	Define a timeline and adhere to it. If longer intervals are required, ensure that the participants are kept up-to-date, and provide them with an easy way to bootstrap them back into the material.
Inappropriate medium	Carefully assess the needs of the communication medium. If using e-mail, augment the correspondence with permanent, indexed resources, such as a Wiki, or a web site cataloging what has been said/found in previous rounds.
Inappropriate question format	Carefully investigate the possibility of using closed questions, rather than prompting for free-form answers, but do not avoid free-form out of convenience if the goal calls for free-form responses — exploratpry studies usually do.
Huge data volume	Focus on single questions only, as otherwise too much data are generated. Seek methods to reduce data which do not rely on interpretation, outsource the process to multiple people working in parallel to eliminate reduction errors, and/or let a third party supervise the process.
Non-strict sampling	Choose an appropriate sampling technique and follow it *strictly*.
Participant drop-out	Attempt to be as accurate as possible about expected time requirements for participation. Be prepared to compensate participants, and ensure they know how important their contribution is to the cause.
Ungrounded concepts	When the goal is to explore, investigate ways in which the participants themselves can coin terms for concepts.
Loose follow-up discussions	Discussions between facilitator and participants which go beyond clarification introduce unquantifiable bias, make data analysis harder, and should be avoided.
Bias	Keep meticulous records and documentation of all decisions; since bias cannot be avoided, the researcher needs to be explicit about it.

Table 8.2.: *Pitfalls of the Delphi method identified during my research, along with potential remedies.*

Epilogue

I would like to thank the panelists of my study one more time for their participation, for their patience, for their dedication, and for the incredible depth of insight they have shared. You have made this study extremely enticing, challenging, fruitful, and fun for me.

Andreas, Charles, Christian, Colin, Damyan, Enrico, Guillem, Gunnar, James, Joey, Jonas, Loïc, Luca, Neil, Niko, Pierre, Raphaël, Russ, Scott, Stefano, and Steve — this would not have been possible without you.

Bibliography

Adams, D. A., Nelson, R. R., and Todd, P. A. (1992). Perceived usefulness, ease of use, and usage of information technology: a replication. *MIS Quarterly*, 16(2):227–47.

Adler, M. and Ziglio, E. (1996). *Gazing into the Oracle: the Delphi Method and its Application to Social Policy and Public Health*. Jessica Kingsley Publishers.

Ajzen, I. and Fishbein, M. (1980). *Understanding Attitudes and Predicting Social Behavior*. Prentice-Hall, Eaglewood Cliffs, NJ, USA.

Albaum, G. S., Evangelista, F., and Medina, N. (1998). Role of response behavior theory in survey research: A cross-national study. *Journal of Business Research*, (2):115–25.

Allen, T. J. (1970). Communication networks in R&D laboratories. *R&D Management*, 1(1):14–21.

Allen, T. J. and Henn, G. (2006). *The Organization and Architecture of Innovation: Managing the Flow of Technology*. Butterworth-Heinemann, Oxford, UK.

Amabile, T. M., Conti, R., Coon, H., Lazenby, J., and Herron, M. (1996). Assessing the work environment for creativity. *Academy of Management Journal*, 39(5):1154–84.

Amor-Iglesias, J.-J., González-Barahona, J. M., Robles-Martínez, G., and Herráiz-Tabernero, I. (2005a). Measuring libre software using Debian 3.1 (sarge) as a case study: Preliminary results. *UPGRADE: the European Journal for the Informatics Professional*, 6(3):13–6.

Amor-Iglesias, J.-J., Robles-Martínez, G., González-Barahona, J. M., and Herráiz-Tabernero, I. (2005b). From pigs to stripes: A travel through Debian. In *Proceedings of the 6th Debian Conference*, Helsinki, Finland.

Anderson, B. A., Puur, A., Silver, B. D., Soova, H., and Vöörmann, R. (1994). Use of a lottery as an incentive for survey participation: a pilot survey in Estonia. *International Journal of Public Opinion Research*, 6:64–71.

Armstrong, M. (1989). *The Delphi Technique*. Princeton Economic Institute, Princeton, NJ, USA.

Bagozzi, R. P. (2007). The legacy of the technology acceptance model and a proposal for a paradigm shift. *Journal of the Association for Information Systems*, 8(4):244–54.

Bagozzi, R. P., Davis, F. D., and Warshaw, P. R. (1992). Development and test of a theory of technological learning and usage. *Human Relations*, 45(7):659–86.

Bagozzi, R. P. and Dholakai, U. (1999). Goal setting and goal striving in consumer behavior. *Journal of Marketing*, 63:19–32.

Bardecki, M. J. (1984). Participants' response to the Delphi method: an attitudinal perspective. *Technological Forecasting and Social Change*, 25(3):281–92.

Barnes, J. L. (1987). *An International Study of Curricular Organizers for the Study of Technology*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA. Unpublished.

Beal, G. M., Rogers, E. M., and Bohlen, J. M. (1957). Validity of the concept of stages in the adoption process. *Rural Sociology*, 22:166–8.

Beech, B. (1999). Go the extra mile - use the Delphi technique. *Journal of Nursing Management*, 7(5):281–8.

Bell, D. (1976). *The Coming of Post-Industrial Society: a Venture in Social Forecasting*. Basic Books, New York, NY, USA.

Berdou, E. (2007). *Managing the Bazaar: Commercialization and Peripheral Participation in Mature, Community-led Free/Open Oource Software Projects*. PhD thesis, London School of Economics, London, UK.

Berger, P. L. and Luckmann, T. (1966). *The Social Construction of Reality*. Anchor Books, New York, NY, USA.

Berk, M. L., Mathiowetz, N. A., Ward, E. P., and White, A. A. (1987). The effect of prepaid and promised incentives: Results of a controlled experiment. *Journal of Official Statistics*, 3(4):449–57.

Bermejo, J. and Dai, N. (2006). Open source strengths for defining software product line practices. Presented at the First International Workshop on Open Source Software and Product Lines, 10th International Software Product Line Conference (SPLC 2006).

Boehm, B. W. (1981). *Software Engineering Economics*. Advances in Computing Science & Technology. Prentice-Hall.

Boland, R. J. (1985). Phenomenology: A preferred approach to research in information systems. In Mumford, E., Hirschheim, R., Fitzgerald, G., and Wood-Harper, T., editors, *Research Methods in Information Systems (IFIP WG 8.2 Colloquium Proceedings)*, pages 193–201, Amsterdam, NL. North-Holland Publishing Co.

Brehm, J. (1994). Stubbing our toes for a foot in the door? Prior contact, incentives and survey response. *International Journal of Public Opinion Research*, 6(1):45–63.

Burns, T. and Stalker, G. M. (1994). *The Management of Innovation*. Oxford University Press, Cary, NC, USA, 2nd edition.

Buxton, W. and Sniderman, R. (1980). Iteration in the design of the human-computer interface. In *Proceedings of the 13th Annual Meeting, Human Factors Association of Canada*, pages 72–81.

Chan, S. (1982). Expert judgments under uncertainty: some evidence and suggestions. *Social Science Quarterly*, 63(1):428–444.

Chau, P. Y. (1996). An empirical assessment of a modified technology acceptance model. *Journal of Management Information Systems*, 13(2):185–204.

Chau, P. Y. K. and Tam, K. Y. (1997). Factors affecting the adoption of open systems: an exploratory study. *MIS Quarterly*, 21(1):1–24.

Chesbrough, H. (2003). *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press.

Church, A. H. (1993). Estimating the effect of incentives on mail survey response rates: a meta-analysis. *Public Opinion Quarterly,*, 57(1):62–79.

Clark, H. H. and Brennan, S. E. (1991). Grounding in communication. In Resnick, L. B., Levine, J. M., and Teasley, S. D., editors, *Perspectives on Socially Shared Cognition*, pages 127–49. American Psychological Association, Washington, DC.

Coates, J. F. (1975). In defense of Delphi: a review of Delphi assessment, expert opinion, forecasting, and group process by h. sackman. *Technological Forecasting and Social Change*, 7(2):193–4.

Cohen, W. M. and Levinthal, D. A. (1990). Absorptive capacity. *Administrative Science Quarterly*, 35(1):128–52.

Coleman, E. G. (2005a). *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.

Coleman, E. G. (2005b). Three ethical moments in Debian: the making of an (ethical) hacker, part III. In Coleman [2005a], chapter 6.

Conboy, K. (2006). *A Framework of Method Agility in Information Systems Development*. PhD thesis, University of Limerick, Limerick, Ireland.

Converse, J. M. and Presser, S. (1986). *Survey Questions: Handcrafting the Standardized Questionnaire*. Quantitative Applications in the Social Sciences. Sage Publications.

Crowston, K. and Howison, J. (2003). The social structure of open source software development teams. In *Proceedings of the OASIS 2003 Workshop (IFIP 8.2 WG)*.

Crowston, K., Howison, J., and Annabi, H. (2006). Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, 11(2):123–48.

Cyphert, F. R. and Gant, W. L. (1970). The Delphi technique. *Journal of Teacher Education*, 21:422.

Dalkey, N. and Helmer, O. (1963). An experimental application of the Delphi method to the use of experts. *Management Science*, 9(3):458–67.

Dalkey, N. C. (1967). Delphi. Technical Report P-3704, The RAND Corporation, Santa Monica, CA, USA.

Dalkey, N. C. (1972). The Delphi method: an experimental study of group opinion. pages 13–54. Lexington Books, Lexington, MA, USA.

Dance, F. E. X. and Larson, C. E. (1976). *The Functions of Human Communication: a Theoretical Approach*. Holt, Rinehart & Winston, New York, NY, USA.

Daniels, N. (1978). Merit and meritocracy. *Philosophy and Public Affairs*, 3:206–23.

Davidow, W. H. and Malone, M. S. (1992). *The Virtual Organization*. Harper Collins, New York, NY, USA.

Davis, F. D. (1986). *A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results*. PhD thesis, Massachusetts Institute of Technology, Sloan School of Management.

Davis, F. D., Bagozzi, R. P., and Warshaw, P. R. (1989). User acceptance of computer technology: a comparison of two theoretical models. *Management Science*, 35(8):982–1003.

Debian Project (2004a). The Debian Free Software Guidelines. Last accessed: 31 July 2006.

Debian Project (2004b). Debian Social Contract. Last accessed: 31 July 2006.

Debian Project (2005). *The Debian Policy Manual*. Last accessed: 31 July 2006.

Debian Project (2006). Constitution for the Debian Project. online. Last accessed: 4 October 2007.

Debian Project (2007a). The Debian New Maintainer process. online. Last accessed: 8 October 2007.

Debian Project (2007b). *How to Report a Bug in Debian*. Last accessed: 10 October 2007.

Delbecq, A. L., van de Ven, A. H., and Gustafson, D. H. (1975). *Group Techniques for Program Planning*. Scott, Foresman, and CoS, Glenview, IL, USA.

Deutschmann, P. J. and Borda, O. F. (1962). La communicacíon de las ideas entre los campesinos Columbianos. Technical report, Universidad Nacional de Colombia.

Deutskens, E., de Ruyter, K., Wetzels, M., and Oosterveld, P. (2004). Response rate and response quality of internet-based surveys: an experimental study. *Marketing Letters*, 15(1):21–36.

Dickson, G. W., Leitheiser, R. L., Wetherbe, J. C., and Nechis, M. (1984). Key information systems issues for the 1980's. *MIS Quarterly*, 8(3):135–59.

Dillman, D. A. (2000). *Mail and Internet Surveys: the Tailored Design Method*. John Wiley & Sons, New York.

Donnellon, A. and Scully, M. (1994). *Teams, Performance, and Rewards — Will the Post-Bureaucratic Organization be a Post-Meritocratic Organization*, chapter 3, pages 63–90. In Heckscher and Donnellon [1994].

Downs, G. W. and Mohr, L. B. (1976). Conceptual issues in the study of innovation. *Administrative Science Quarterly*, 21(4).

Eccles, R. G. and Nohria, N. (1990). The post-structuralist organization. Technical Report 92-003, Harvard Business School, Boston, MA, USA.

Farrell, J. and Saloner, G. (1985). Standardization, compatibility, and innovation. *The RAND Journal of Economics*, 16(1):70–83.

Farrell, J. and Saloner, G. (1986). Installed base and compatibility: Innovation, product preannouncements, and predation. *The American Economic Review*, 76(5):940–55.

Feller, J. and Fitzgerald, B. (2002). *Understanding Open Source Software Development*. Addison-Wesley, London, England.

Festinger, L., Schachter, S., and Back, K. (1950). The spatial ecology of group formation. In Festinger, L., Schachter, S., and Back, K., editors, *Social Pressure in Informal Groups*, chapter 4. Stanford University Press, San Francisco, CA, USA.

Fichman, R. G. (1992). Information technology diffusion: a review of empirical research. In DeGross, Becker, and Elam, editors, *Proceedings of the Thirteenth International Conference on Information Systems (ICIS)*, pages 195–206, Dallas, TX, USA.

Fichman, R. G. (2004). Going beyond the dominant paradigm for information technology innovation research: Emerging concepts and methods. *Journal of the Association for Information Systems*, 5(8):314–55.

Fishbein, M. (1967). Attitude and the prediction of behavior. In Fishbein, M., editor, *Readings in Attitude Theory and Measurement*, pages 477–92. Wiley.

Fishbein, M. and Ajzen, I. (1975). *Belief, Attitude and Behavior: an Introduction to Theory and Research*. Addison-Wesley.

Fitzgerald, B. (1997). *Mehodology-in-Action: the Nature of Usage of Systems Development Methodologies in Practice.* Doctorate thesis, University of London, London, UK.

Fortuna, M. A., , and Melián, C. J. (2007). Do scale-free regulatory networks allow more expression than random ones? *Journal of Theoretical Biology*, 247(2):331–6.

Fowler, F. J. (1993). *Survey Research Methods*. Sage Publications, Thousand Oaks, CA, USA.

Fowler, F. J. and Mangione, T. W. (1990). *Standardized Survey Interviewing: Minimizing Interviewer-related Error*. Sage Publications, Newbury Park, CA, USA.

Frambach, R. T. and Schillewaert, N. (2002). Organizational innovation adoption: a multi-level framework of determinants and opportunities for future research. *Journal of Business Research*, 55(2):163–76.

Franklin, K. K. and Hart, J. K. (2007). Idea generation and exploration: Benefits and limitations of the policy Delphi research method. *Innovative Higher Education*, 31(4):237–46.

Garfield, M. J. (2005). Acceptance of ubiquitous computing. *Information Systems Management*, 22(4):24–31.

Garzarelli, G. and Galoppini, R. (2003). Capability coordination in modular organization: Voluntary FS/OSS production and the case of Debian GNU/Linux. Working paper, University of the Witwatersrand - School of Economics and Business Sciences.

Gazpacho (2007). Night. Audio recording.

Gladwell, M. (2002). *The Tipping Point: How Little Things Can Make a Big Difference*. Back Bay Books, Nashville, TN, USA.

Glaser, B. G. and Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Co., Chicago, IL, USA.

Goetz, E. G., Tyler, T. R., and Cook, F. L. (1984). Promised incentives in media research: a look at data quality, sample representativeness, and response rate. *Journal of Marketing Research*, 21(2):148–54.

Goldschmidt, P. F. (1975). Scientific inquiry or political critique? Remarks on Delphi assessment, expert opinion, forecasting, and group process by H. Sackman. *Technological Forecasting and Social Change*, 7:195–213.

González-Barahona, J. M., Pérez, M. A. O., de las Heras Quirós, P., González, J. C., and Olivera, V. M. (2001). Counting potatoes: the size of Debian 2.2. *UPGRADE: the European Journal for the Informatics Professional*, 2(6):60–6.

González-Barahona, J. M. and Robles-Martínez, G. (2003). Unmounting the "code god" assumption. Technical report, GSyC, Universidad Rey Juan Carlos.

Goodman, C. M. (1987). The Delphi technique: a critique. *J Adv Nurs*, 12(6):729–34.

Gordon, F. M. (1994a). *Bureaucracy — Can we do better? We can do worse*, chapter 8, pages 195–210. In Heckscher and Donnellon [1994].

Gordon, T. and Pease, A. (2006). RT Delphi: an efficient, "round-less" almost real time Delphi method. *Technological Forecasting and Social Change*, 73(4):321–33.

Gordon, T. J. (1994b). The Delphi method. Technical report, AC/UNU Millennium Project, The Futures Group.

Gordon, T. J. and Hayward, H. (1968). Initial experiments with the cross impact matrix method of forecasting. *Futures*, 1(2):100–16.

Gordon, T. J. and Helmer-Hirschberg, O. (1964). Report on a long-range forecasting study. Technical Report P-2982, RAND Corperation.

Hackman, J. R. (1980). The psychology of self-management in organizations. In Pallack, M. S. and Perloff, R. O., editors, *Psychology and Work: Productivity, Change and Employment*, pages 89–134. American Psychology Association, Washington, DC, USA.

Hackman, J. R. and Oldham, G. R. (1980). *Work Redesign*. Addison-Wesley, Reading, MA, USA.

Haegerstrand, T. (1967). *Innovation Diffusion as a Spatial Process*. University of Chicago Press, Chicago, IL, USA. translated from Swedish (1953) by Allan Pred.

Halloran, T. and Scherlis, W. L. (2002). High quality and open source software practices. In *Proceedings of the 24th International Conference on Software Engineering*.

Hassinger, E. (1959). Stages in the adoption process. *Rural Sociology*, 24:52–3.

Hawkins, D., Coney, K. A., and Jackson, D. W. (1988). The impact of monetary inducement on uninformed response error. *Journal of the Academy of Marketing Science*, 16(2):30–5.

Heckscher, C. (1994). *Defining the Post-Bureaucratic Type*, chapter 2, pages 14–62. In Heckscher and Donnellon [1994].

Heckscher, C. and Donnellon, A., editors (1994). *The Post-Bureaucratic Organization: New Perspectives on Organizational Change*. Sage Publications, Inc., Thousand Oaks, CA, USA.

Helmer, O. (1977). Problems in futures research: Delphi and causal cross-impact analysis. *Futures*, 9(1):17–31.

Helmer, O. and Rescher, N. (1959). On the epistemology of the inexact sciences. *Management Science*, 6(1):25–52.

Hertel, G., Niedner, S., and Hermann, S. (2003). Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, 32:1159–77.

Hildreth, P. and Kimble, C. (2004). *Knowledge Networks: Innovation through Communities of Practice*. Idea Group Inc., London, UK.

Hildreth, P. M. and Kimble, C. (2002). The duality of knowledge. *Information Research*, 8(1).

Hill, K. Q. and Fowles, J. (1975). The methodological worth of the Delphi forecasting technique. *Technological Forecasting and Social Change*, 7(2):179–92.

Hirschheim, R. (1985). Information systems epistemology: an historical perspective. In Mumford, E., Hirschheim, R., Fitzgerald, G., and Wood-Harper, T., editors, *Research Methods in Information Systems (IFIP WG 8.2 Colloquium Proceedings)*, pages 3–36, Amsterdam, NL. North-Holland Publishing Co.

Hsu, C.-C. and Sandford, B. A. (2007a). The Delphi technique: Making sense of consensus. *Practical Assessment, Research & Evaluation*, 12(10).

Hsu, C.-C. and Sandford, B. A. (2007b). Minimizing non-response in the Delphi process: how to respond to non-response. *Practical Assessment, Research & Evaluation*, 12(17).

Hung, H.-L., Altschuld, J. W., and Leec, Y.-F. (2008). Methodological and conceptual issues confronting a cross-country Delphi study of educational program evaluation. *Evaluation and Program Planning*, 31(2):191–8.

Häfliger, S., von Krogh, G., and Späth, S. (2008). Code reuse in open source software. *Management Science*, 54(1):180—-93.

Janis, I. L. (1972). *Victims of Groupthink*. Houghton Mifflin Company, Boston, MA, USA.

Jensen, C. and Scacchi, W. (2005). Modeling recruitment and role migration processes in OSSD projects. In *Proceedings of 6th International Workshop on Software Process Simulation and Modeling*.

Judd, R. C. (1972). Use of Delphi methods in higher education. *Technological Forecasting and Social Change*, 4(2):173–86.

Kaplan, L. M. (1971). The use of the Delphi method in organizational communication: a case study. Unpublished master's thesis, Ohio State University, Columbus, OH, USA.

Katz, M. L. and Shapiro, C. (1985). Network externalities, competition, and compatibility. *The American Economic Review*, 75(3):424–40.

Katz, R. and Allen, T. J. (1982). Investigating the not invented here (NIH) syndrome: A look at the performance, tenure, and communication patterns of 50 R&D project groups. *R&D Management*, 12(1):7–20.

Kearns, K. P. (1992). Innovations in local governments: a sociocognitive network approach. *Knowledge, Technology & Policy*, 5(2):45–67.

Keeney, S., Hasson, F., and McKenna, H. (2006). Consulting the oracle: Ten lessons from using the Delphi technique in nursing research. *Journal of Advanced Nursing*, 53(2):205–12.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*. Norton, New York, NY, USA.

Kerlinger, F. N. (1973). *Foundations of Behavioral Research*. Holt, Rinehart, and Winston, Inc., New York, NY, USA.

Kimble, C., Hildreth, P., and Wright, P. (2001). Communities of practice: Going virtual. In Malhotra, Y., editor, *Knowledge Management and Business Model Innovation*, chapter 13, pages 220–34. Idea Group Publishing, London, UK.

Klein, H. K. and Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 23(1):67–93.

Klein, J. A. (1994). *The Paradox of Quality Management — Commitment, Ownership, and Control*, chapter 7, pages 178–94. In Heckscher and Donnellon [1994].

Kline, S. K. and Rosenberg, N. (1986). An overview of innovation. In Landau, R. and Rosenberg, N., editors, *The Positive Sum Strategy: Harnessing Technology for Economic Growth*, pages 275–304. National Academy Press, Washington, DC, USA.

Kojo, T., Männistö, T., and Soininen, T. (2003). *Towards Intelligent Support for Managing Evolution of Configurable Software Product Families*, volume 2649/2003 of *Lecture Notes in Computer Science*, chapter 7, pages 86–101. Springer Verlag, Berlin, Germany.

Krackhardt, D. (1994). *Constraints on the Interactive Organization as an Ideal Type*, chapter 9, pages 211–22. In Heckscher and Donnellon [1994].

Krafft, M. F. (2005). *The Debian System — Concepts and Techniques*. Open Source Press, Munich, Germany.

Kuhn, T. S. (1970). *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, IL, USA, 2nd edition.

Kwon, T. H. and Zmud, R. W. (1987). Unifying the fragmented models of information systems implementation. In Boland, R. J. and Hirschheim, R. A., editors, *Critical Issues in Information Systems Research*, chapter 10, pages 227–51. John Wiley & Sons Ltd.

Lakhani, K. R. and Wolf, R. G. (2005). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In Feller, J., Fitzgerald, B., Hissam, S. A., and Lakhani, K. R., editors, *Perspectives on Free and Open Source Software*, pages 3–22. MIT Press, Cambridge, MA, USA.

Lameter, C. (2002). Debian GNU/Linux: The past, the present and the future. online. Presented at the Free Software Symposium 2002 on October 22, 2002 15:30 at the Japan Education Center.

Lave, J. and Wenger, E. (1991). *Situated Learning. Legitimate Peripheral Participation*. Cambridge University Press.

Led Zeppelin (1973). Houses of the Holy. Audio recording.

Lee, A. S. (1991). Integrating positivist and interpretive approaches to organizational research. *Organization Science*, 2:342–65.

Lee, Y., Kozar, K. A., and Larsen, K. R. (2003). The Technology Acceptance Model: Past, present, and future. *Communications of JAIS*, 12.

Leibenstein, H. (1950). Bandwagon, snob, and veblen effects in the theory of consumers' demand. *The Quarterly Journal of Economics*, 64(2):183–207.

Levy, S. (1984). *Hackers: Heroes of the Computer Revolution*. Penguin Books, New York, NY, USA.

Lewin, K. (1952). Group decision and social change. In Newcomb, T. M. and Hartley, E. L., editors, *Readings in Social Psychology*, pages 459–73. Henry Holt.

Lin, N. and Zaltman, G. (1973). Dimensions of innovations. In Zaltman, G., editor, *Processes and Phenomena of Social Change*, pages 93–115. Wiley Interscience, New York, NY, USA.

Linstone, H. A. (1975). Eight basic pitfalls: A checklist. In Linstone and Turoff [2002], chapter VIII, pages 573–86.

Linstone, H. A. (1978). The Delphi technique. pages 271–300. Greenwood Press, Westport, CT, USA.

Linstone, H. A. and Turoff, M. (1975a). Introduction. In Linstone and Turoff [2002], chapter I, pages 3–12.

Linstone, H. A. and Turoff, M. (1975b). Introduction to the chapter "computers and the future of Delphi". In Linstone and Turoff [2002], chapter VII, pages 483–9.

Linstone, H. A. and Turoff, M., editors (2002). *The Delphi Method: Techniques and Applications*. Information Systems Department, New Jersey Institute of Technology.

Ludwig, B. G. (1994). Internationalizing extension: an exploration of the characteristics evident in a state university extension system that achieves internationalization. Unpublished doctoral dissertation, The Ohio State University, Columbus, OH, USA.

Ludwig, B. G. (1997). Predicting the future: Have you considered using the Delphi methodology? *Journal of Extension*, 35(5).

Mahajan, V., Muller, E., and Bass, F. M. (1990). New product diffusion models in marketing: a review and directions for research. *Journal of Marketing*, 54(1):1–26.

Marchant, E. W. (1988). Methodological problems associated with the use of the Delphi technique — some comments. *Fire Technology*, 24(1):59–62.

Markus, M. L. (1990). Toward a "critical mass" theory of interactive media universal access, interdependence and diffusion. In Fulk, J. and Steinfield, C. W., editors, *Organizations and Communication Technology*, volume 14, pages 491–511. Sage Publications, Newbury Park, CA, USA.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502.

McKenna, H. P. (1994). The Delphi technique: a worthwhile research approach for nursing? *Journal of Advanced Nursing*, 19(6):1221–5.

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

Michlmayr, M. (2005a). Quality improvement in volunteer free software projects: Exploring the impact of release management. In Scotto, M. and Succi, G., editors, *Proceedings of the First International Conference on Open Source Systems*, pages 309–10, Genova, Italy.

Michlmayr, M. (2005b). Software process maturity and the success of free software projects. In Zieliński, K. and Szmuc, T., editors, *Proceedings of the 7th Conference on Software Engineering, KKIO 2005*, chapter 1, pages 3–14. IOS Press, Krakow, Poland. accepted.

Michlmayr, M. (2007). *Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management.* PhD thesis, University of Cambridge, Cambridge, UK.

Michlmayr, M. and Hill, B. M. (2003). Quality and the reliance on individuals in free software projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pages 105–9, Portland, OR, USA.

Michlmayr, M., Hunt, F., and Probert, D. (2005). Quality practice and problems in free software projects. In Scotto, M. and Succi, G., editors, *Proceedings of the First International Conference on Open Source Systems*, pages 24–28, Genova, IT.

Michlmayr, M., Hunt, F., and Probert, D. (2007a). Release management in free software projects: Practices and problems. In Feller, J., Fitzgerald, B., Scacchi, W., and Silitti, A., editors, *Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software*, pages 295–300. Springer Verlag, New York, NY, USA.

Michlmayr, M., Robles, G., and Gonzalez-Barahona, J. M. (2007b). Volunteers in large libre software projects: a quantitative analysis over time. In Robles et al. [2007], chapter 1, pages 1–24.

Michlmayr, M. and Senyard, A. (2006). A statistical analysis of defects in Debian and strategies for improving quality in free software projects. In Bitzer, J. and Schröder, P. J. H., editors, *The Economics of Open Source Software Development*, pages 131–48. Elsevier Science Publications B.V.

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63:81–97.

Miller, L. E. (2006). Determining what could/should be: the Delphi technique and its application. Paper presented at the meeting of the 2006 annual meeting of the Mid-Western Educational Research Association, Columbus, OH, USA.

Mintzberg, H. (1979). An emerging strategy of "direct" research. *Administrative Science Quarterly*, 24(4):582–9.

Mitchell, T. R. and Larson, Jr., J. R. (1987). *People in Organizations: an Introduction to Organizational Behavior*. McGraw-Hill, New York, NY, USA.

Mitchell, V. W. (1991). The Delphi technique: an exposition and application. *Technology Analysis & Strategic Management*, 3(4):333–58.

Mitroff, I. I. and Turoff, M. (1975). Philosophical and methodological foundations of Delphi. In Linstone and Turoff [2002], chapter II.B, pages 17–36.

Mohr, L. B. (1982). *Explaining Organizational Behavior.* Jossey-Bass, San Francisco, CA, USA.

Moore, G. A. (1991). *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*. Harper Business Essentials.

Mullen, P. M. (2003). Delphi: Myths and reality. *Journal of Health Organisation and Management*, 17(1):37–52.

Murray, J. P. (1992). Expectations of department chairpersons: a Delphi case study. *Journal of Staff, Program & Organization Development*, 10:13–21.

Murray, T. J. (1979). Delphi methodologies: a review and critique. *Urban Systems*, 4(2):153–8.

Nelms, K. R. and Porter, A. L. (1985). EFTE: An interactive Delphi method. *Technological Forecasting and Social Change*, 28(1):43–61.

Nohria, N. and Berkeley, J. D. (1994). *The Virtual Organization*, chapter 5, pages 118–28. In Heckscher and Donnellon [1994].

Novak, J. D. and Cañas, A. J. (2006). The theory underlying concept maps and how to construct and use them. Technical report, Florida Institute for Human and Machine Cognition, Pensacola, FL, USA.

Oezbek, C. (2008). Introducing innovations into open source projects. Milestone, Internal Report 1, Freie Universität Berlin, Berlin, DE.

Oezbek, C. (2009). Introducing innovations into open source projects. Milestone, Internal Report 2, Freie Universität Berlin, Berlin, DE.

Oezbek, C. (TBA). *Introducing Innovations into Open Source Projects*. Dissertation, Freie Universität Berlin. To appear.

Oezbek, C. and Prechelt, L. (2007). On understanding how to introduce an innovation to an open source project. *Upgrade*, 8(6):44–4.

Oh, K. H. (1974). Forecasting through hierarchical Delphi. Unpublished doctoral dissertation, Ohio State University, Columbus, OH, USA.

Okoli, C. and Pawlowski, S. D. (2004). The Delphi method as a research tool: an example, design considerations and applications. *Information & Management*, 42(1):15–29.

O'Mahony, S. and Ferraro, F. (2004). Managing the boundary of an 'open' project. Working Paper 03-60, Harvard University.

O'Mahony, S. and Ferraro, F. (2007). The emergence of governance in an open source community. *The Academy of Management Journal*, 50(5).

Oppenheim, A. N. (2001). *Questionaire Design, Interviewing and Attitude Measurement*. Continuum, London, UK.

Oppermann, M. (1995). E-mail surveys — potentials and pitfalls. *Marketing Research*, 7(3):28–33.

Ozanne, U. B. and Churchill, G. A. (1971). Five dimensions of the industrial adoption process. *Journal of Marketing Research*, 8:322–8.

Parkinson, C. N. (1957). *Parkinson's Law and Other Stories*. Houghton Mifflin, Boston, MA, USA.

Patton, M. Q. (2001). *Qualitative Evaluation and Research Methods*. Sage Publications, Newbury Park, CA, USA, 3 edition.

Paul, C. L. (2008). A modified Delphi approach to a new card sorting methodology. *Journal of Usability Studies*, 4(1):7–30.

Peiró Moreno, S. and Argelaguet Portella, E. (1993). Consensus doesn't always mean agreement. *Gaseta sanitària de Barcelona*, 7(39):292–300.

Perry, J. L. and Kraemer, K. L. (1978). Innovation attributes, policy intervention, and the diffusion of computer applications among local governments. *Policy Sciences*, 9(2):179–205.

Pierce, J. L. and Delbecq, A. L. (1977). Organization structure, individual attributes and innovation. *Academy of Management Review*, 2(1):27–37.

Pijpers, A. G. M. (2001). Senior executives' use of information technology. *Information and Software Technology*, 43:959–71.

Pill, J. (1971). The Delphi method: Substance, context, a critique and an annotated bibliography. *Socio-Economic Planning Sciences*, 5(1):57–71.

Popper, K. (1972). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge and Kegan Paul, London, England.

Popper, K. (1994). *Knowledge and the Body-Mind Problem: In Defence of Interaction*. Routledge, Abingdon, UK.

Porcupine Tree (1995). The Sky Moves Sideways. Audio recording.

Poster, M. (1990). *The Mode of Information: Post-Structuralism and Social Context*. University of Chicago Press, Chicago, IL, USA.

Powell, C. (2003). The Delphi technique: Myths and realities. *Journal of Advanced Nursing*, 41(4):376–82.

Presser, S. and Blair, J. (1994). Survey pretesting: Do different methods produce different results? *Sociological Methodology*, ,:73–104.

Presser, S., Couper, M. P., Lessler, J. T., Martin, E., Martin, J., Rothgeb, J. M., and Singer, E. (2004). Methods for testing and evaluating survey questions. *Public Opinion Quarterly*, 68(1):109–30.

Rabin, M. and Schrag, J. L. (1999). First impressions matter: a model of confirmatory bias. *Quarterly Journal of Economics*, 114(1):37–82.

Raghavan, S. A. and Chand, D. R. (1989). Diffusing software-engineering methods. *IEEE Software*, 6(4):81–90.

Rahim, S. A. (1961). *The Diffusion and Adoption of Agricultural Practices: a Study in a Village in East Pakistan*. Pakistan Academy for Village Development, Comilla, East Pakistan.

Rauch, W. (1979). The decision Delphi. *Technological Forecasting and Social Change*, 15(3):159–69.

Raymond, E. S. (1999). *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Reid, N. G. (1988). The Delphi technique, its contribution to the evaluation of professional practice. In Ellis, R., editor, *Professional Competence and Quality Assurance in the Caring Professions*, pages 230–62. Croom-Helm, Beckenham, Kent, UK.

Rennie, D. (1981). Consensus statements. *New England Journal of Medicine*, 304:665–6.

Rieger, W. G. (1986). Directions in Delphi developments: Dissertations and their quality. *Technological Forecasting and Social Change*, 29(2):195–204.

Robertson, T. S. and Gatignon, H. (1986). Competitive effects on technology diffusion. *Journal of Marketing*, 50(3):1–12.

Robles, G., Barahona, J. M. G., Michlmayr, M., and Amor, J. J. (2006a). Mining large software compilations over time: Another perspective of software evolution. In *International Workshop on Mining Software Repositories*, Shanghai, China.

Robles, G., Dueñas, S., and Gonzalez-Barahona, J. M. (2007). Corporate involvement of libre software: Study of presence in Debian code over time. In *Open Source Development, Adoption and Innovation: IFIP Working Group 2.13 on Open Source Software*, volume 234/2007 of *IFIP International Federation for Information Processing*, pages 121–32. Springer Verlag, Boston, MA, USA.

Robles, G. and Gonzalez-Barahona, J. M. (2006). Contributor turnover in libre software projects. In Damiani, E., Fitzgerald, B., Scacchi, W., and Scotto, M., editors, *Proceedings of the 2nd International Conference on Open Source Systems*, pages 273–286, Como, Italy.

Robles, G., González-Barahona, J. M., and Herraiz, I. (2005a). Challenges in software evolution: the libre software perspective. Paper presented at the Workshop "Challenges in Software Evolution".

Robles, G., González-Barahona, J. M., and Michlmayr, M. (2005b). Evolution of volunteer participation in libre software projects: Evidence from Debian. In Scotto, M. and Succi, G., editors, *Proceedings of the First International Conference on Open Source Systems*, pages 100–7, Genova, Italy.

Robles, G., López, L., González-Barahona, J. M., and Herraiz, I. (2006b). Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering*, 1.

Robles, G., Merelo, J. J., and Gonzalez-Barahona, J. M. (2005c). Self-organized development in libre software: a model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling*, St. Louis, MO, USA. co-hosted at the ICSE2005.

Rogers, E. M. (1962). *Diffusion of Innovations*. Free Press, London, UK, 1st edition.

Rogers, E. M. (2003). *Diffusion of Innovations*. Free Press, London, UK, 5th edition.

Rohlfs, J. (1974). A theory of interdependent demand for a communications service. *The Bell Journal of Economics and Management Science*, 5(1):16–37.

Rugg, G. and Mcgeorge, P. (2005). The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, 22(3):94–107.

Sackman, H. (1974). Delphi assessment: Expert opinion, forecasting, and group process. Technical Report R-1283-PR, RAND Corporation, Santa Monica, CA, USA.

Sackman, H. (1975). *Delphi Critique*. Lexington Books.

Salancik, J. R., Wenger, W., and Helfer, E. (1971). The construction of Delphi event statements. *Technological Forecasting and Social Change*, 3:65–73.

Salkind, N. (2006). *Exploring Research*. Pearson Education, Upper Saddle River, NJ, USA.

Sandford, B. (2002). *A National Assessment of the Activities, Perceived Instructional Needs and Appropriate Methods of Delivering Professional Development for Part-Time Technical and Occupational Education Faculty in the Community College of the United States.* PhD thesis, Ohio State University, Columbus, OH, USA.

Scheele, D. S. (1975a). Consumerism comes to Delphi: Comments on Delphi assessment, expert opinion, forecasting, and group process by H. Sackman. *Technological Forecasting and Social Change*, 7(2):215–219.

Scheele, D. S. (1975b). Real construction as product of Delphi interaction. In Linstone and Turoff [2002], chapter II.C, pages 37–71.

Schmidt, R. C. (1997). Managing Delphi surveys using nonparametric statistical techniques. *Decision Sciences*, 28(3):763–74.

Schultze, U. (2000). A confessional account of an ethnography about knowledge work. *MIS Quarterly*, 24(1):3–41.

Schumpeter, J. (1934). *The Theory of Economic Development: an Inquiry into Profits, Capital, Credit, Interest and the Business Cycle*. Harvard University Press, 1949 edition.

Schumpeter, J. (1939). *Business Cycles: a Theoretical, Historical, and Statistical Analysis of the Capitalist Process*, volume 2. McGraw-Hill.

Schumpeter, J. (1942). *Capitalism, Socialism and Democracy*. George Allen & Unwin.

Self, K. M., Whysall, P., and Bowmer, W. (2002). Why Debian rocks. online. Last accessed: 11 October 2007 [OFFLINE].

Senyard, A. and Michlmayr, M. (2004). How to have a successful free software project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*. IEEE Computer Society.

Sheehan, K. B. and Hoy, M. G. (2006). Using e-mail to survey internet users in ihe united states: Methodology and assessment. *Journal of Computer-Mediated Communication*, 4(3).

Silva, L. (2007). Post-positivist review of technology acceptance model. *Journal of the Association for Information Systems*, 8(4):256–266.

Silverman, D. (2005). *Doing Qualitative Research*. Sage Publications, London, UK.

Skulmoski, G. J., Hartman, F. T., and Krahn, J. (2007). The Delphi method for graduate research. *Journal of Information Technology Education*, 6:1–21.

Solomon, D. J. (2001). Conducting web-based surveys. *Practical Assessment, Research & Evaluation*, 7(19).

Sowe, S., Stamelos, I., and Angelis, L. (2006). Identifying knowledge brokers that yield software engineering knowledge in OSS projects. *Information and Software Technology*, 48(11):1025–33.

Sowe, S. K., Stamelos, I., and Angelis, L. (2007). Understanding knowledge sharing activities in free/open source software projects: an empirical study. *Journal of Systems and Software*, in press.

Spinelli, T. (1983). The Delphi decision-making process. *Journal of Psychology*, 113(1):73–80.

Sproull, L. and Kiesler, S. (1991). Computers, networks, and work. *Scientific American*, 9:119.

Sproull, L. S. (1986). Using electronic mail for data collection in organizational research. *Academy of Management Journal*, 29:159–69.

Späth, S., Stürmer, M., Häfliger, S., and von Krogh, G. (2007). Sampling in open source software development: The case for using the Debian GNU/Linux distribution. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS)*.

Späth, S., Stürmer, M., and von Krogh, G. (2008). Incentives and costs in implementing private-collective innovation: a case study. Technical report, ETH Zürich.

Späth, S., Stürmer, M., and von Krogh, G. (2010, forthcoming). Enabling knowledge creation through outsiders: towards a push model of open innovation. *International Journal of Technology Management, Special Issue on Open Innovation*.

Späth, S., von Krogh, G., Stürmer, M., and Häfliger, S. (2009). A lightweight model of component reuse in open source software. Working paper, ETH Zürich, Zürich, Switzerland.

Steinlin, G. (2009). Kooperation in der open source entwicklung — eine empirische untersuchung des debian projekts. Master's thesis, Universität Bern, Bern.

Stellman, A. and Greene, J. (2005). *Applied Software Project Management*. O'Reilly Media.

Strader, T. J., Ramaswami, S. N., and Houle, P. A. (2007). Perceived network externalities and communication technology acceptance. *European Journal of Information Systems*, 16:54–65.

Strauss, A. L. and Corbin, J. (1998). *Basics of Qualitative Research*. Sage Publications, 2nd edition.

Stürmer, M. (2005). Open source community building. Master's thesis, University of Berne, Berne, Switzerland.

Stürmer, M. (2009). *How Firms Make Friends: Communities in Private-Collective Innovation*. Doctoral dissertation, ETH Zürich, Zürich, Switzerland.

Subramanyam, R. and Xia, M. (2008). Free/libre open source software development in developing and developed countries: a conceptual framework with an exploratory study. *Decision Support Systems*, 46:173–86.

Surowiecki, J. (2004). *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Random House, New York, NY, USA.

Swanson, E. B. (1994). Information systems innovation among organizations. *Management Science*, 40(9):1069–92.

Sánchez-Fernández, J., Muñoz-Leiva, F., Montoro-Ríos, F. J., and Ángel Ibáñez-Zapata (2008). An analysis of the effect of pre-incentives and post-incentives based on draws on response to web surveys. In *Quality and Quantity*. Springer Netherlands.

Tarde, G. (1903). *The Laws of Imitation*. University of Chicago Press, Chicago, IL, USA.

The Beatles (1967). Sgt. Pepper's Lonely Hearts Club Band. Audio recording.

Tornatzky, L. G. and Klein, K. J. (1982). Innovation characteristics and innovation adoption-implementation: A meta-analysis of findings. *IEEE Transactions on Engineering Management*, 29:28–45.

Turkle, S. (1984). *The Second Self: Computers and the Human Spirit*. Simon and Schuster, New York, NY, USA.

Turoff, M. (1970). The design of a policy Delphi. *Technological Forecasting and Social Change*, 2(2):149–71.

Turoff, M. (1975). The policy Delphi. In Linstone and Turoff [2002], chapter III, pages 80–96.

Turoff, M. and Hiltz, S. R. (1996). Computer-based Delphi processes. In *Gazing Into the Oracle: The Delphi Method*. Kingsley Publishers.

Venkatesh, V. and Bala, H. (TBA). Tam 3: Advancing the technology acceptance model with a focus on interventions. Manuscript in preparation.

Venkatesh, V. and Davis, F. D. (2000). A theoretical extension of the Technology Acceptance Model: four longitudinal field studies. *Management Science*, 46(2):186–204.

Venkatesh, V., Morris, M. G., Davis, G. B., and Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS Quarterly*, 27(3):425–78.

von Hippel, E. (1986). Lead users: A source of novel product concepts. *Management Science*, 32(7):791–805.

von Hippel, E. (1988). *The Sources of Innovation*. Oxford University Press.

Wallach, H., Allan, M., and Harries, D. (2005). The Debian New Maintainer process: History and aims. In *Proceedings of the 6th Debian Conference*, Helsinki, Finland. Debian.

Walsham, G. (1995). The emergence of interpretivism in is research. *Information Systems Research*, 6(4):376–94.

Wasko, M. M. and Faraj, S. (2000). "it is what one does": Why people participate and help others in electronic communities of practice. *The Journal of Strategic Information Systems*, 9(2–3):155–73.

Weaver, W. T. (1971). The Delphi forecasting method. *Phi Delta Kappan*, 52(5):267–73.

Weber, M. (1947). *The Theory of Social and Economic Organization*. The Free Press, New York, NY, USA.

Weber, R. (2004a). The rhetoric of positivism versus interpretivism: A personal view. *MIS Quarterly*, 28(1):3–xiii.

Weber, S. (2004b). *The Success of Open Source*. Harvard University Press, Cambridge, MA, USA.

Wejnert, B. (2002). Integrating models of diffusion of innovations: a conceptual framework. *Annual Review of Sociology*, 28:297–326.

Wejnert, B. (2005). Diffusion, development, and democracy, 1800–1999. *American Sociological Review*, 70(1):53–81.

Welty, G. (1972). Problems of selecting experts for Delphi exercises. *Academy of Management*, 15:121–4.

Wenger, E. (1998). *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press, Cambridge, UK.

West, J. and Gallagher, S. (2005). Patterns of open innovation in open source software. chapter 5. Oxford University Press.

White, Jr., G. and Kaplan, A. (2002). Incentives in a business survey: A study in improving response rates. In *Proceedings of the Joint Statistical Meetings 2002 New York*, Alexandria, VA, USA. American Statistical Association.

White, R. T. and Mackay, L. D. (1973). A note on a possible alternative to likert scales. *Research in Science Education*, 3(1):75–6.

Witkin, B. R. and Altschuld, J. W. (1995). *Planning and Conducting Needs Assessment: a Practical Guide*. Sage Publications, Inc., Thousand Oaks, CA, USA.

Wixom, B. H. and Todd, P. A. (2005). A theoretical integration of user satisfaction and technology acceptance. *Information Systems Research*, 16(1):85–102.

Wolcott, H. F. (2001). *Writing up Qualitative Research*. Sage Publications, London, UK, 2 edition.

Ye, Y., Nakakoji, K., Yamamoto, Y., and Kishida, K. (2004). The co-evolution of systems and communities in free and open source software development. In *Free/Open Source Software Development*, pages 59–92. Idea Group Publishing, Hershey, PA, USA.

Yin, R. K. (2003). *Case Study Research: Design and Methods*. Sage publications, London, UK, 3rd edition.

Young, M. (1959). *The Rise of the Meritocracy*. Penguin Books, New York, NY, USA.

Yousuf, M. I. (2007a). The Delphi technique. *Essays in Education*, 20:80–9.

Yousuf, M. I. (2007b). Using experts' opinions through Delphi technique. *Practical Assessment, Research & Evaluation*, 12(4).

Zahra, S. A. and George, G. (2002). Absorptive capacity: a review, reconceptualization, and extension. *Academy of Management Review*, 27(2):185–203.

Zmud, R. W. (1982). Diffusion of modern software practices: Influence of centralization and formalization. *Management Science*, 28(12):1421–1431.

Zmud, R. W. (1984). An examination of 'push-pull' theory applied to process innovation in knowledge work. *Management Science*, 30(6):727–38.

Appendix A

Background information on Debian

A.1. Terminology

A.1.1. System

In the Debian world, one will often encounter the word "system." The Oxford English Dictionary[1] does not offer a concise definition, but in WordNet, I found a system to be defined as "instrumentality that combines interrelated interacting artifacts designed to work as a coherent entity". [2] When applied in the Debian context, the word can take any of three meanings:

1. from the perspective of an individual user, a Debian system is a computer device controlled by one of the operating systems produced by the Debian Project, such as Debian GNU/Linux.
2. from the perspective of a system administrator, who manages multiple Debian systems, the Debian System comprises Debian's administration concepts and techniques.[3].

[1] `http://www.askoxford.com/dictionaries/compact_oed` [4 Nov 2007]
[2] `http://wordnetweb.princeton.edu/perl/webwn?s=system` [02 Sep 2009]
[3] This is the meaning to which I allude with the title of my book: *The Debian System — Concepts and Techniques*.

3. from the perspective of the Debian Project, its developers and contributors, the Debian System is the common vision of an operating system, observing Debian's foundation documents (see appendix A.6 and appendix A.1.5).

A.1.2. Packages

The fundamental unit of development in the Debian Project is the package (see [Krafft, 2005, ch. 5] for an exhaustive exploration of Debian packages). A package is a container for software to be installed on (or removed from) a target system. We need to distinguish between two types of Debian packages:

Binary packages — are what users install onto their system to augment its functionality. Packages are usually handled by programmes known as package managers, which keep track of the changes introduced by a package during installation, and hence allow for later manipulation of installed packages as a unit, *e.g.* enabling their traceless removal.

Binary packages consists of two components, which are concatenated into a single The Debian package file format (DEB) file (extension `.deb`):

Payload/data — executable programmes, libraries, scripts, data, and configuration files installed onto the target system, which are necessary to run a programme, or which implement a set of related commands or features. Every Debian package's payload must furthermore include copyright information and a change log. Documentation and examples may be included.

Control information — meta data, which include: descriptions targeted at the human operator, dependency information relating the package to other packages, scripts to interact with the installing user, and scripts to fine-tune the integration of installed files with the rest of the target system, and to coordinate their later upgrade or removal.

Even though DEB files are best handled by the family of Debian package managers,[4] they can also be manipulated with standard Unix tools.

[4] `dpkg` is the package handler (which prevents actions violating the Debian Policy; see appendix A.1.5), while Advanced Package Tool (APT)-based tools are higher-level package managers (which can use the packages' metadata to pre-emptively resolve problems).

Source packages — provide the sources needed to generate ("build") binary packages. In this context, sources may include code to be compiled, data transformed to binary formats during the build process, or files transferred verbatim onto the target system. A source package consist of two or three files, which can be manipulated with standard Unix tools.

Most software in the Debian archive is developed by another FLOSS project (called "upstream"), but has been made available as a non-native Debian package. On the other hand, native Debian packages contain software developed specifically for the Debian System.[5]

When users talk about "packages" without further specification, they likely refer to binary packages. On the other hand, developers usually refer to source packages.

One of the central activities of Debian Contributors (DCs) is (source) package maintenance, which involves:

- importing new upstream versions.
- analyse bug reports, correspond with the submitter, forward information upstream, and/or fix the problem (this is known as "bug triaging").
- look after the package's meta data and make sure that relationships to other packages are correct and current.
- follow and implement Debian Policy changes.
- develop new features, or better means of integration with the rest of the Debian System.

Section 2.3.2 illustrates the workflow of Debian package maintenance.

A.1.3. Bugs and bug reports

The Debian Project project makes extensive use of its publicly-accessible[6] Bug Tracking System (BTS),[7] publishes instructions on how to submit bug reports [Debian Project, 2007b], and maintains a number of interactive bug reporting tools to facilitate the process.

[5] In the case of a native Debian package, the source package consists of two files: the source archive ("tarball") and the `.dsc` file with control data; non-native packages are made up of three files: the original source "tarball", a patch with Debian-specific changes ("diff"), and the `.dsc` file with control data.

[6] The project promises in its Social Contract (see appendix A.6) that it "will keep [the] entire bug report database open for public view at all times."

[7] `http://bugs.debian.org`

A bug report is made up of the original report and any number of replies sent to the bug report's address. All interaction takes place via e-mail. Reports and replies may include attachments. Every bug report in the Debian BTS belongs to one (or more) packages and has a unique ID, which is usually prefixed by a pound sign when mentioned in writing: *e.g.* #123456. Ideally, a bug report describes a single fault in the corresponding package.[8] For each report, the BTS tracks submitter, responsible person ("owner"), severity, tags, and the software versions to which this bug applies, all of which may be manipulated by anyone, but a log of all actions is kept.[9] The available severities are: critical, grave, serious, important, normal, minor, wishlist. The first three form the class of release-critical (RC) bugs.

In Debian parlance, "bug report" is often abbreviated to "bug", *e.g.* "have you replied to my bug yet?" In this thesis, an attempt will be made to use the separate terms accordingly. A bug is thus a fault in the software contained in the package, or its packaging, and bug report refers to the discussion thread following and including the initial report, encompassing the meta data tracked by the BTS.

A.1.4. Archives

When a project member uploads a package,[10] it is processed by the archive management scripts. These perform a number of tests on the package, checking *e.g.* for integrity and authenticity. If the package passes these, it is installed into the archive's package pool.[11] Package indices refer users (or rather: package manager software invoked by users) to the correct location within this pool.

The Debian archive comprises five different collections of package indices: unstable, testing, stable, oldstable, and experimental. These are themselves often referred to as archives. In this text, the use of the word "archive" in the context of Debian will be qualified: Debian archive refers to the entire Debian package pool and all indices, while *e.g.* the unstable archive identifies the collection of indices referencing packages considered to be unstable, together with these packages, and analogously for the other collections: testing, stable, oldstable, and experimental. The collection names by themselves are synonymous to the respective archives: "this package is not yet in testing." All archives except for the experimental archive can be used to install

[8] Debian bug reports can be split ("cloned") and merged to achieve this one-to-one correspondence.
[9] Manipulation happens via e-mail: `http://www.debian.org/Bugs/server-control` [11 Oct 2007]
[10] That is, the files that constitute a (signed source) package, together with a file identifying the upload (the `.changes` file), which also has to be cryptographically signed for the upload to be considered.
[11] `http://ftp.debian.org/debian/pool/`

computer systems; a user tracking the unstable archive is said to run Debian unstable, or just "unstable".

Every package starts its life-cycle in the unstable archive, replacing any previous versions. It remains there for a given period of time to allow for potential problems to surface. If during this period, no new RC bugs are reported against the package, and a number of other criteria are met, the package "migrates to testing," which means that it propagates to the testing archive, again replacing any previous versions.

Packages in testing remain in this archive until a new stable version is released. Leading up to such a release, the testing archive is frozen, and only packages fixing RC bugs are allowed to propagate from unstable. During the freeze period, developers are encouraged to concentrate on fixing bugs.

At the end of the freeze, packages with remaining RC bugs are reassessed and generally removed from the testing archive (they continue to exist in unstable). Finally, a new stable release is made: the current stable replaces oldstable, and testing replaces stable. A new stable release is thus made up of packages without any (known) RC bugs.

The experimental archive holds packages which are not yet polished enough to enter the regular package life-cycle. Project members may make use of it at their own discretion, and no automatic migration takes place.

Figure A.1 on the next page on the facing page depicts a package's life cycle.

A.1.5. The Debian Policy

The Debian Policy Manual is a fundamental Debian document, which defines the "technical requirements that each package must satisfy to be included in the distribution" [Debian Project, 2005]. The manual helps to lessen the number of decisions a package maintainer has to make, and ensures that packages, which have been built in accordance with its rules, will be compatible with the Debian System and the tools used for its management. A compliant package can furthermore coexist with thousands of other packages without conflicts. Familiarity with the policy is required of each Debian developer, and policy compliance is a core requirement in package maintenance.

The Debian Policy has been described as

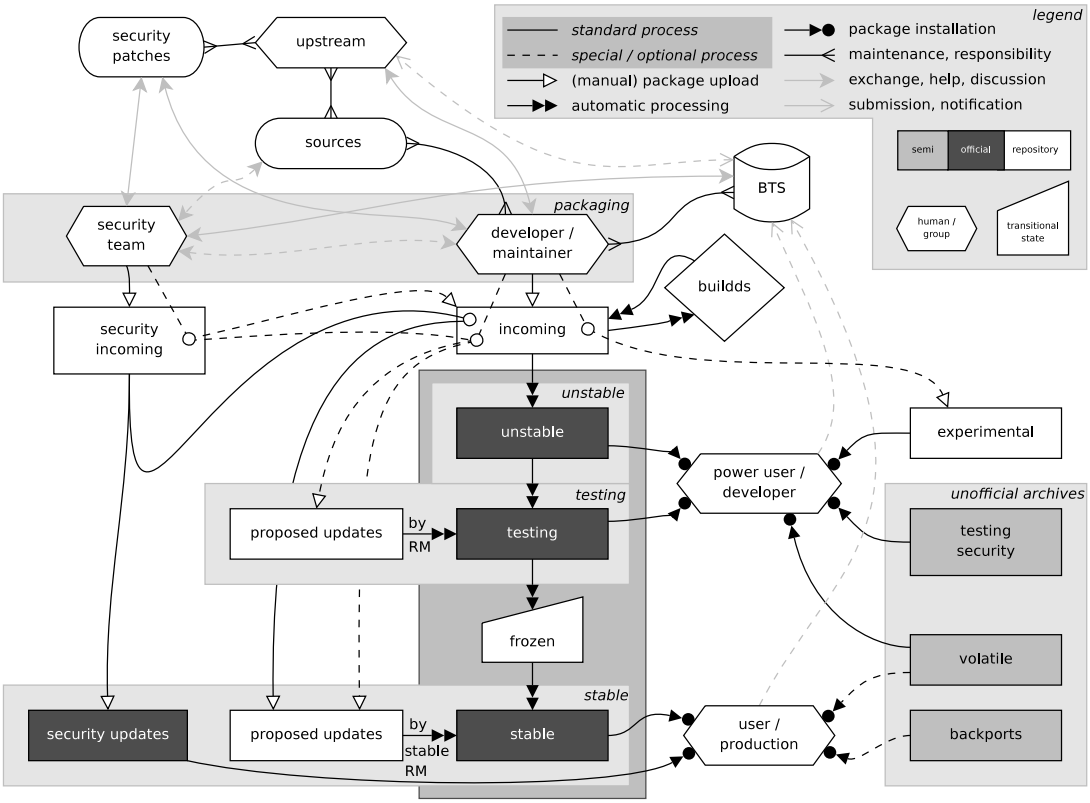

Figure A.1.: *The life-cycle of a Debian package: a complex flow between various stages and involving various parties at different times; for this text, it suffices to remember the level of complexity, it is not necessary to understand the graph.*

> the crux, the narthex, the throbbing heart of Debian and what makes it so utterly superior to all other operating systems. [...] What Policy defines are the bounds of Debian, not your own actions on the system. [...] In essence, Policy introduces a new class of bugs, policy bugs. Policy bugs are release-critical — **a package which violates policy will not be included in the official stable Debian release**. [...] That is the whole secret. [Self et al., 2002, their emphasis]

In my book, I used the metaphor of an orchestra [Krafft, 2005, p. 202ff]:

> The Debian System is not unlike a symphony: the musicians are the developers who prepare numerous packages for installation on the system. If developers simply create packages to their own liking, synergy cannot emerge. Therefore, the developers have agreed on a set of rules by which to abide, just like the members of an orchestra agree on a score to follow. Within the Debian System, the role of the conductor is taken by the package management tools, which observe certain

rules and ensure that packages harmonise. The rules as well as the tools have been around for years, and developers have had ample time to practise their use, and to correct problems.

To continue the example, the score played by the Debian Developers (DDs) and observed by the package management tools is the Debian Policy; without the policy, the Debian distribution would be Just Another Linux. But it is not. The policy is the soul of the Debian System, it is its throbbing heart, it is the reason why Debian can put the same tools to better use than others. The policy is Debian's cookbook, with years of scrutiny perfecting each single recipe.

A.2. Communication media apart from e-mail

In section 2.2.7, I introduced the use of e-mail in the Debian Project. The following is a brief overview of other communication channels in use, separated into asynchronous (correspondents need not be active at the same time) and synchronous media (correspondents are interacting in real-time).

Asynchronous communication There have been a number of attempts to operate web forums on Debian topics,[12] but never have such forums entered the domain of development — they exclusively served as support platform for users of the system. The project has also never officially supported such a forum.

Other asynchronous media include the Debian Wiki,[13] a collaborative web platform where multiple people can collect and maintain information on topic-specific pages, as well as their relations. The Wiki is accessible only via a web browser, which is a frequent source of criticism, because it makes it harder for a contributor to use the text editor with which s/he is accustomed. Nevertheless, solutions to this problem exits, and the use of the Wiki has been continuously increasing.

Finally, Planet Debian[14] is a weblog aggregator, which accumulates Debian-related journal entries from the Web pages of a vast number of contributors. Every now and then, discussions take place between weblogs, in part because the nature of the platform provides for greater exposure

[12] *e.g.* `http://forums.debian.net/`
[13] `http://wiki.debian.org`
[14] `http://planet.debian.org`

of each entry ("post") and is thus often favoured. Since such discussions are even harder to follow than mailing list threads, and since they leave out contributors who do not operate a journal themselves, technical discussions are generally frowned upon.

Synchronous communication The chief synchronous communication medium is Internet relay chat (IRC),[15] a chat system which is organised around topical channels in which any number of people may discuss issues in real-time. Even though it is possible to exchange private messages, messages are generally seen by everyone present in the channel at the time. Several channels host so-called "bots", which relay information from another source in the project infrastructure to the channel, such as commit messages of a version control system (VCS), or bug report mutations[16].

IRC is a useful way to stay current, and its real-time nature paves the way for rapid collaboration. Furthermore, the medium fosters social interaction and allows contributors to chat and joke with each other. Even though many channels are logged (and thus searchable *post-hoc*), the channels are considered informal communication platforms in the project, and participants are generally encouraged to forward important insights or solutions to the rest of the project on the appropriate mailing list.

Last, but not least, Debian has a tradition and culture of real-life meetings, in the form of conferences,[17] Bug Squashing Partys (BSPs), "sprints" (collaboration sessions), and parties. Due to the global spread of contributors, contributors frequently cannot take part in those sessions. In general, DCs try to keep the outside world in the loop through Planet Debian and summary messages to the various mailing lists.

A.3. Evolution of membership in Debian and the role of contributors

A.3.1. Debian's organisational structure

The Debian project is organised according to the structure described in its constitution (see appendix A.3.4). which establishes the decision-making bodies and the processes for making

[15] The Debian Project uses the OFTC network, accessible via `irc.debian.org`.
[16] *e.g.* the `#debian-devel-changes` channel.
[17] `http://debconf.org`

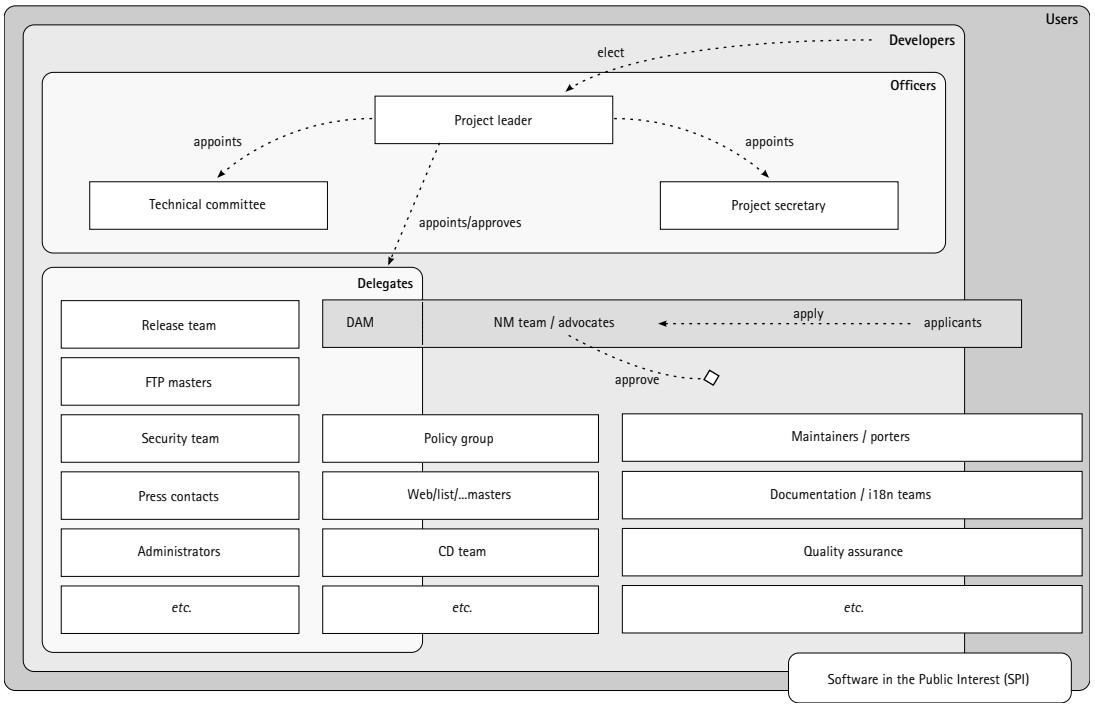

Figure A.2.: *A rough sketch of Debian's organisation*

decisions within the project. Figure A.2 on the current page is a rough approximation of the structure of the Debian project.

Software in the Public Interest (SPI) is the legal entity of the Debian Project, founded by Bruce Perens in 1997 to hold all trademarks for Debian, own all of its monetary and material assets, and represent the project in legal matters. SPI has no authority over decisions cast within the Debian project.

The following sections recount the history of Debian with respect to the evolution of membership in the project.

A.3.2. 1993–1996: The beginnings

In its early years, the Debian Project was just a small interest group, and most users of Debian GNU/Linux were themselves developers of the system Sowe et al. [*cf.* 2007], who knew each other and cooperated closely. During that era, anyone who took the time to express an interest in the development of the system was given access: Steve McIntyre recalls in his 2007 Debian Project Leader (DPL) platform that in 1996, he "installed Debian, then two days later [...] mailed Bruce

Perens [the DPL at the time] with a Pretty Good Privacy (PGP) key and asked him for a Debian login. [Bruce] responded almost immediately with account details."[18]

At that time, it was commonly accepted practice (but not policy) for members of the project to sign new versions of packages uploaded to the Debian archive with their respective PGP keys, which allowed for the verification of authenticity and integrity of new packages.

A.3.3. 1996–1997: Exploding growth

With increasing media attention following the project's first public release, Debian 1.1 ("buzz"[19]), project members started to raise concerns over quality and security issues, as new "members were being admitted at rates faster than the project's ad hoc social systems could integrate them" [Coleman, 2005b]. At time of release of "buzz", the project had 90 members; at the time of release of the successive release, Debian 1.2 ("rex") six months later, this number had grown to 120.

The project started to require signed package uploads around the time of the release of Debian 1.3 ("bo"; June 1997; 200 members), and restricted the set of people allowed to make uploads to the official project members. But Debian's roster continued to mushroom, minimising the effect of this new security measure. Long-winded, year-long discussions ensued on the project's mailing list, in which the topic of verification of new project members continuously percolated to the top. After Perens instituted Debian's infamous Social Contract (see appendix A.6) in 1997, "prospective members needed only to informally demonstrate their technical competence and to claim knowledge of and adherence to the project's Social Contract and policies before being admitted to the project" [Coleman, 2005b].

Over debates on membership verification and requirements, many started to voice concerns, such as Manoj Srivastava: "I'm not sure about competence or integrity requirements; somehow it goes against the grain for someone who is not issuing my paycheck."[20] Meanwhile, project leader Perens, who was solely in charge of accepting new members at the time, persistently made a case for the importance of identity verification to guard against rogue maintainers, and following the rejection of a number of applicants on the basis of trust issues, several members began questioning his authority and the role of the leader.

[18] `http://www.debian.org/vote/2007/platforms/93sam` [3 Oct 2007]
[19] Debian releases are named after characters of the computer-animation series *Toy Story*; see appendix A.5.
[20] `http://lists.debian.org/msgid-search/87ybc8l3qo.fsf@tiamat.datasync.com` [6 Oct 2007]

A.3.4. 1998: The Debian Constitution

After taking over leadership in January 1998, the third DPL, Ian Jackson, initiated the public drafting of the Debian constitution in March to document the rights and obligations of project members and their leader.[21] The document was ratified by public vote in December 1998,[22] a few months after the release of Debian 2.0 ("hamm"); the number of project members had again doubled and grown beyond 400.

The Constitution [Debian Project, 2006] allows any member of the project to

- make any technical or nontechnical decision with regard to their own work;
- propose or sponsor draft general resolutions;
- propose themselves as a project leader candidate in elections;
- vote on general resolutions and in leadership elections.

The Constitution also manifests the volunteer nature of the project:

> Nothing in this constitution imposes an obligation on anyone to do work for the Project. A person who does not want to do a task which has been delegated or assigned to them does not need to do it. However, they must not actively work against these rules and decisions properly made under them. [...]
>
> A person may leave the Project or resign from a particular post they hold, at any time, by stating so publicly.

The document furthermore specifies the powers of the project leader and defines the procedure of leadership:

> The Project Leader should attempt to make decisions which are consistent with the consensus of the opinions of the Developers.
>
> Where practical the Project Leader should informally solicit the views of the Developers.
>
> The Project Leader should avoid overemphasizing their own point of view when making decisions in their capacity as Leader.

[21] `http://lists.debian.org/msgid-search/E0yFXzq-00009z-00@anarres.greenend.org.uk` [6 Oct 2007]
[22] `http://www.debian.org/vote/1999/vote_0000` [6 Oct 2007]

Finally, the Constitution places the member collective at the top, stating, among other rights, that "the Developers by way of General Resolution or election" may "appoint or recall the Project Leader, amend this constitution, [...] and make or override any decision authorised by the powers of the Project Leader or a Delegate. [...]"

The Constitution does *not* specify the following three privileges, which are generally considered part of the developer status, but which are granted by the project delegates specific in brackets:

- upload *any* package to the Debian archive management queue, where it will be processed and installed into the archive [FTP masters];
- access (or request access to) machines owned and administered by the Debian Project, so-called developer machines [system administrators];
- receive emails via the `debian-private` mailing list [list masters].

In the context of this thesis, I assume the definition of Debian's New Maintainer (NM) team: a Debian developer is "a Debian Project member, who has gone through the NM process and had their application accepted" [Debian Project, 2007a]. To make sense of this definition, we have to continue following the evolution of membership in the Debian Project.

A.3.5. 1997–present: The New Maintainers process

The discussions on the verification of integrity and identity of new members led to the delegation of a new maintainer authority, consisting of two members, in April 1997.[23] After a wave of new applicants were accepted, the first signs of stagnation showed in August of the same year,[24] until James Troup and Martin "Joey" Schulze "hatched a plan to basically hijack new-maintainer in order to rescue it from stagnation" [James Troup, cited in [Wallach et al., 2005]] in September. The two were in charge of processing applicants and made weekly announcements about new maintainers for the better part of 18 months. Shortly after Troup and Schulze took over, the project instituted the `debian-mentors` mailing list[25] as a forum where new maintainers could ask for help from the more experienced. Around that time, the concept of package sponsorship first surfaced. A sponsor is "a Debian Project member who acts as the mentor of an applicant: s/he checks packages provided by the applicant and helps to find problems, [...]

[23] Message <m0wKAwZ-00IdecC@golem.pixar.com> to `debian-private` [15 Oct 2007]
[24] Message <19970818203920.51870@dungeon.inka.de> to `debian-private` [15 Oct 2007]
[25] `http://lists.debian.org/msgid-search/199711102134.QAA02106@hmm.nowhere` [7 Oct 2007]

to improve the packaging, [and to] upload [the package] on behalf of the applicant to the Debian archive. The applicant is recorded as the maintainer of such a package." [Debian Project, 2007a].

In July 1999, Schulze announced the freeze of the new maintainer process,[26] citing quality issues related to uncontrolled growth as the reason, and claiming that Debian should take a step back and return to quality as top priority, instead of further growth. In addition, more rigorous checks and vetting of applicants were needed to maintain the integrity of the distribution, as well as to keep the workload for the NM team manageable. This freeze was meant to stay private but found its way to the public within a few weeks.[27] The project was facing a crisis as project members and outsiders alike demanded the recommencement of new maintainer processing.[28]

Troup announced his intent to resign from NM in October. In response, the fourth DPL, Wichert Akkerman, who had stepped up to the throne in January, proposed the creation of a NM committee,[29] composed of "experienced Developers, with strong views in favour of free software, who could be trusted more than ordinary active Developers, and who understood the responsibility involved in accepting new Developers to Debian" [Wallach et al., 2005]. The proposal required applicants to identify themselves in person towards at least one project member, using an officially issued identification document; this identity verification was recorded by exchanging cryptographic signatures.

Under the lead of Dale Scheetz, the details of this proposal were worked out and NM officially relaunched towards the end of the year and ran smoothly for the next years, despite repeated complaints about the slowness of the process by applicants who were growing impatient.

In 2001, Martin Michlmayr, the seventh DPL, added the requirement for NM applicants to present a body of contributions with their application, to be backed by a project member ("advocate"), who would confirm the contributions and vouch for the new maintainer. In general, a package in the archive, maintained by the applicant, was sufficient. The concept of sponsorship was formalised.

[26] Message <19990702115419.D27941@finlandia.infodrom.north.de> to debian-private [15 Oct 2007]
[27] http://lists.debian.org/msgid-search/19990805074250.a528@box [6 Oct 2007]
[28] Message <oaiu4nr4by.fsf_-_@burrito.onshore.com> to debian-private [15 Oct 2007]
[29] http://lists.debian.org/msgid-search/19991017121536.A6299@mors.net [6 Oct 2007]

A.3.6. 2007: The Debian Maintainers role

In later years, the requirement for applicants to maintain a package was subject of many criticisms and statements of frustration as Debian's body of contributors grew to encompass translators, documentation writers, graphic designers, testers, people fixing bugs and submitting patches, and mailing list participants. Such contributions were generally appreciated: "It cannot be stated enough that it takes more than just package maintainers to make Debian work as a distribution."[30] To the contributors, however, not being given access to project resources or the right to vote was not very motivational. Suggestions on instituting an official Debian contributor status were made every couple of weeks,[31] but it took until July 2007 for these efforts to achieve the goal: in an official vote, the status of Debian Maintainer (DM) was created.[32]

A.3.7. 2008: Debian contributors

In October 2008, the Debian account manager published a proposal for new role definitions to the developer announcement list[33], suggesting to augment the set of official project membership roles (developers and maintainers) with project members and contributors. The rationale was provided as follows: "Debian is about developing a free operating system, but there's more in an operating system than just software and packages. If we want translators, documentation writers, artists, free software advocates, *et al.* to get endorsed by the project and feel proud for it, we need some way to acknowledge that."

Unfortunately, the proposal was met with heavy opposition, although it was not the proposed changes that were criticised, but mainly the cabalist, top-down approach by which the proposal had been drafted and published (*cf.* the discussions on resistance and consensus in *cf.*[sections]influence-resistance and 7.2.6.1). To date, no advancements have been made into this direction, but the use of the term "DC" grew, in my opinion, which has led me to speak of contributors throughout this work (see section 2.2.1).

[30] `http://lists.debian.org/msgid-search/20020129214344.GA31327@netexpress.net` [15 Oct 2007]
[31] *e.g.* `http://lists.debian.org/msgid-search/200308081807.46433@fortytwo.ch` [15 Oct 2007]
[32] `http://www.debian.org/vote/2007/vote_003` [15 Oct 2007]
[33] `http://lists.debian.org/debian-devel-announce/2008/10/msg00005.html` [23 Jan 2009]

#	Variable	Value
	Timestamp	04 Mar 2010
	Age of the Debian Project	16 years
	Current Debian stable release	5.0.4, codenamed "lenny"
1.	Official member count	1002 + 83
2.	DD origin countries	55
3.	Contributors to Debian	around 2,500 regular contributors
4.	Number of supported architectures	12
5.	Number of source packages	15,840
6.	Number of binary packages	27,724 (amd64)
7.	Number of derivatives	102

Table A.1.: *Debian figures and facts*

A.4. Numbers and their sources

The Debian Project is active around the clock and in as such a constantly moving target. At various points in this thesis, I quote figures and facts on the project (see table A.1 on this page). As no official source of these data exists, this section explains how they were gathered.

1. Official member count — The official member count is made up of two figures: the number of DDs, and the number of DMs (see appendix A.3.6). The number of developers I established from the official membership database and the `debian-keyring` package. The number of maintainers is derived from the `debian-keyring` package alone. I needed to consult the database for the number of developers, because some developers had GNU Privacy Guard (GPG) and PGP keys in the keyrings (for historical reasons), and thus I needed to remove the duplicates. DMs do not have duplicates, because PGP has been deprecated for years.

   I should explicitly note that this figure only includes official project menbers. There are many contributors actively working to further Debian who are not included in this number.

2. DD origin countries — To determine the number of different countries, one needs access to a developer machine to run the following command; the country attribute is access-protected:

```
ldapsearch -x -LLL -H ldap://db.debian.org \
  -b ou=users,dc=debian,dc=org gidNumber=800 c \
  | sed -ne 's,^c: ,,p' | sort -u | wc -l
```

The number does not include countries of DMs or other contributors.

3. Number of DCs — The number of regular contributors to Debian was estimated by Steinlin [2009] to be 2,500.

4. Number of supported architectures — The number of supported architectures can be gleaned from the Debian archive, *e.g.* by counting the number of `binary-*` subdirectories under `http://ftp.debian.org/debian/dists/stable/main`.

5. Number of source packages — The number of source packages is equivalent to the number of matches of the regular expression `/^Package:/` in the source package index of the unstable archive, *e.g.* `http://ftp.debian.org/debian/dists/unstable/main/source/Sources.bz2`. Note that the count is limited to free packages, and packages in the contrib and non-free sections of the archive are not included (*cf.* appendix A.7).

6. Number of binary packages — The number of binary packages is defined as the number of packages that can be installed on a system of the `i386` architecture, and is lower for other architectures. It can be determined by counting the number of times the regular expression `/^Package:/` matches in the binary package index of the unstable archive, *e.g.* `http://ftp.debian.org/debian/dists/sid/main/binary-i386/Packages.bz2`. Similar to the source package count, the number only includes packages in the main archive, not in the non-free archives.

7. Number of Debian derivatives — The number of Debian derivatives was determined from the Wikipedia list of Debian, Knoppix, and Ubuntu-based derivatives[34], which is likely incomplete, but probably also includes projects that have been discontinued. As an estimate, the count suffices.

A.5. Debian releases

To date, Debian has made 13 releases of Debian GNU/Linux, as well as several updates to each release. Table A.2 on the next page collects the data of the major releases, and figure A.3 on the following page depicts the timeline over the last 16 years.

[34] `http://en.wikipedia.org/wiki/List_of_Linux_distributions` [21 Sep 2009]

Date	Version	Name	#DDs (DMs)	#Pkgs (Src:i386)	# Architectures	Notes
Aug 1995	0.96r3	n/a	60	250	1 (i386)	*a,b*
Jun 1996	1.1	buzz	90	474	1	*a,b*
Dec 1996	1.2	rex	120	848	1	*a,b*
Jun 1997	1.3	bo	200	1,197	1	*a,b*
Jul 1998	2.0	hamm	400	1,115: 1,958	2 (+m68k)	*a,c*
Jun 1999	2.1	slink	413	1,580: 2,269	4 (+alpha,sparc)	*c,d*
Aug 2000	2.2	potato	448	2,647: 3,889	6 (+powerpc,arm)	*c,d*
Jul 2002	3.0	woody	892	5,218: 8,273	11 (+mips/el,hppa,ia64,s390)	*c,d*
Jun 2005	3.1	sarge	972	8,728: 15,196	11	*e,f*
Apr 2007	4.0	etch	1036 (0)	10,222: 18,110	11 (+amd64, -m68k)	*e,g,h*
Feb 2009	5.0	lenny	1013 (74)	12,121: 22,309	12 (+armel)	*h,i,j*

Notes (a) (approximated) number of developers from Lameter [2002]; (b) (approximated) number of packages from Lameter [2002]; (c) number of packages from indices available at `http://archive.debian.org` [7 Nov 2007]; (d) number of developers from Amor-Iglesias et al. [2005b]; (e) number of packages from indices available at `http://ftp.debian.org/debian/dists` [7 Nov 2007]; (f) number of developers from `http://www.debian.org/vote/2006/vote_002` [4 May 2008]; for lack of a better number (see `http://bugs.debian.org/295527` [7 Nov 2007]); (g) number of developers from `http://www.debian.org/vote/2007/vote_001` [7 Nov 2007] for lack of a better number (see `http://bugs.debian.org/295527` [7 Nov 2007]); (h) number of DMs from the `debian-maintainers` package as included in the respective release; (i) number of packages from indices available at `http://ftp.debian.org/debian/dists/lenny` [21 Sep 2009]; (j) number of developers from `http://www.debian.org/vote/2009/vote_001` [21 Sep 2009] for lack of a better number (see `http://bugs.debian.org/295527` [21 Sep 2009]);

Table A.2.: *Debian releases and their figures*

A.6. Debian's Social Contract

The Social Contract is a foundation document of the Debian project [Debian Project, 2004b] and, to my knowledge, distinguishes the Debian Project from most other FLOSS projects, which do not obey a manifest of this kind.

Coleman [2005b] recounts the genesis of the Social Contract as follows:

> Ean Schuessler came up with the idea for the Contract after a conversation at a conference with Bob Young, the co-founder of a then-emergent commercial Linux

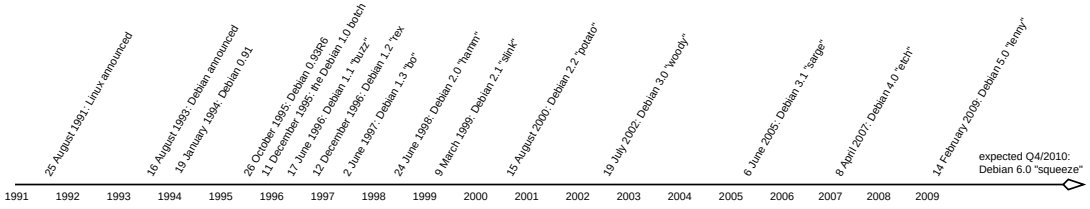

Figure A.3.: *The time-line of Debian GNU/Linux releases.*

> distribution, Red Hat. Ean suggested that Red Hat might want to guarantee in writing that as they grew larger they would always provide GPLed software. Young replied that "would be the kiss of death," implying that such a guarantee made to the users of free software could prove disastrous to his business. Ean (who was himself a business owner) was both amused and disturbed by Young's answer, and with other developers at the conference he decided that it would behoove Debian to provide such a guarantee in writing.

The Social Contract was drafted in 1997 by Bruce Perens, after Ean Schussler suggested the concept of a Linux distribution stating its "social contract with the free software community" to him. In a month-long e-mail conference, several other developers joined to refine the text, before Perens, project leader at the time, announced it as publicly stated policy of the Debian project.

The project ratified a number of changes via an official vote in the summer of 2004[35] but postponed their institution until after the release of Debian 3.1, codenamed "sarge" in June 2005.

The following is thus the updated version of the document, which applies to Debian's most recent release, Debian 5.0.4 ("lenny").

Debian Will Remain 100% Free We provide the guidelines that we use to determine if a work is "free" in the document entitled "The Debian Free Software Guidelines" (see appendix A.7). We promise that the Debian system and all its components will be free according to these guidelines. We will support people who create or use both free and non-free works on Debian. We will never make the system require the use of a non-free component.

We Will Give Back to the Free Software Community When we write new components of the Debian system, we will license them in a manner consistent with the Debian Free Software Guidelines. We will make the best system we can, so that free works will be widely distributed and used. We will communicate things such as bug fixes, improvements and user requests to the "upstream" authors of works included in our system.

[35] `http://www.debian.org/vote/2004/social_contract_reform.3` [26 Sep 2007]

We Won't Hide Problems We will keep our entire bug report database open for public view at all times. Reports that people file online will promptly become visible to others.

Our Priorities are Our Users and Free Software We will be guided by the needs of our users and the Free Software community. We will place their interests first in our priorities. We will support the needs of our users for operation in many different kinds of computing environments. We will not object to non-free works that are intended to be used on Debian systems, or attempt to charge a fee to people who create or use such works. We will allow others to create distributions containing both the Debian system and other works, without any fee from us. In furtherance of these goals, we will provide an integrated system of high-quality materials with no legal restrictions that would prevent such uses of the system.

Programs That Don't Meet Our Free-Software Standards We acknowledge that some of our users require the use of works that do not conform to the Debian Free Software Guidelines. We have created "contrib" and "non-free" areas in our archive for these works. The packages in these areas are not part of the Debian system, although they have been configured for use with Debian. We encourage CD manufacturers to read the licenses of the packages in these areas and determine if they can distribute the packages on their CDs. Thus, although non-free works are not a part of Debian, we support their use and provide infrastructure for non-free packages (such as our bug tracking system and mailing lists).

A.7. The Debian Free Software Guidelines

The Debian Free Software Guidelines (DFSG) [Debian Project, 2004a] regulates the availability of software in Debian's archive according to its licence. Software, whose licence is in accordance with *all* terms of these guidelines, may be included in the official Debian archive. Software licenced incompatibly with these guidelines might also be distributed, but in a secluded section of the archive, called "non-free". Software which is itself free, but which needs non-free components to work, is included in the "contrib" archive.

Free Redistribution The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing

programs from several different sources. The license may not require a royalty or other fee for such sale.

Source Code The program must include source code, and must allow distribution in source code as well as compiled form.

Derived Works The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Integrity of The Author's Source Code The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software. (This is a compromise. The Debian group encourages all authors not to restrict any files, source or binary, from being modified.)

No Discrimination Against Persons or Groups The license must not discriminate against any person or group of persons.

No Discrimination Against Fields of Endeavor The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Distribution of License The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

License Must Not Be Specific to Debian The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.

License Must Not Contaminate Other Software The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.

Example Licenses The "GPL", "BSD", and "Artistic" licenses are examples of licenses that we consider "free".

Appendix B

Background information on diffusions research

B.1. Rogers elements of diffusion

Rogers [2003] identifies four "elements" of diffusion:

- the innovation itself,
- the process of communicating the innovation (the diffusion),
- the adoption process (the time element),
- and the social system in which the diffusion occurs.

These elements of diffusion are often referred to as Rogers' framework, which has been used to assess and engineer diffusions (descriptive and prescriptive use). His tome is possibly the most cited work in diffusion research, even though it is not without problems (see section 3.2.2.1). It is thus in order that I offer a brief description of each.

B.1.1. Innovation

The first set of characteristics in Rogers' theory deal with the individual adopter's perception of the innovation itself [Rogers, 2003, ch. 6]. These attributes are highly subjective, and singular in

the sense that they pertain to each individual in isolation from the rest of the subjects targeted by a diffusion.

Rogers makes no claim that these are the only attributes of an innovation that affect the rate of adoption, just that they are the ones that "explained most of [an] innovation's rate of adoption" [*ibid.*, p. 226]. In a study by Kearns [1992, cited in [Rogers, 2003]], the five attributes of an innovation quoted by Rogers could account for 26% of the 27% of variance that could be explained by a total of 25 perceived attributes; the other 20 attributes hardly had any relevance in the eight diffusions analysed as part of the research.

Relative advantage — a subject is more likely to adopt an innovation that s/he perceives as having an advantage (economic, social, personal, ...) over the *status quo*.

Compatibility — an innovation compatible with the *status quo* — thus fitting in with social norms and requiring little effort — is likely to be more popular.

Complexity — innovations that are difficult to grasp, learn, or use, are likely to be adopted at a slower rate (accessibility).

Trialability — innovations that can be tried without much effort are likely to be adopted quicker, mainly because of the ability to reduce uncertainty about the innovation by using it (learning by doing).

Observability — innovations whose results are observable will likely be favoured among adopters (return on investment, gratification).

B.1.2. Communication channels

Rogers defines communication as "the process by which participants create and share information [...] to reach a mutual understanding" [Rogers, 2003, p. 18]. Rogers isolates two aspects of the channels of communication:

Channel/medium — information may be exchanged face-to-face (interpersonal media), or pushed to a broad audience (mass media). While in the former case, communication is often two-way (discussions, question & answer sessions, presentations), mass media communication is usually a one-way means to reach a larger number of people (radio, newspapers, announcement lists). Innovations which are primarily communicated through interpersonal channels are likely to reach a higher adoption rate, because they are "more effective in persuading an individual," and since potential adopters "depend mainly upon

a subjective evaluation [...] from other individuals like themselves who have already adopted the innovation" [*ibid.*, p. 18f].

Homophily — two peers who are "similar in certain attributes, such as beliefs, education, socioeconomic status, and the like" [*ibid.*, p. 19] are called homophilous.[1] With greater degree of homophily, communication between peers becomes more effective, but the spread of ideas also becomes more horizontal (as it spans areas of competence of peers at the same status level), which can dampen the adoption rate.

B.1.3. Time

The third element of diffusion describes the phases leading up to an adoption and including its verification, which Rogers calls the "innovation-decision process." The process begins with initial knowledge of an innovation and ends with the confirmation of the decision to adopt or reject this innovation by in individual (or other decision-making unit). The author presents a model that breaks the process into five stages [Rogers, 2003, ch. 5]:

Knowledge — at the first stage, the individual learns of an innovation and how it works. The knowledge may be gained following a need, or the need may be developed as a consequence of the knowledge. At this stage, knowledge is mostly cognitive (emotional, nonverbal), but the individual already starts to investigate how the innovation functions.

Persuasion — after being exposed to an innovation, an individual forms an interest or disinterest therein. In case of interest s/he actively seeks to reduce the uncertainty about an innovation's expected consequences. The knowledge of the innovation becomes affective (factual).

Decision — the individual then decides to adopt or reject the innovation, possibly after trying it or witnessing a demonstration. A favourable decision means the individual will move to incorporate the innovation into ongoing practice. A rejection the innovation may be either active or passive: deciding against the innovation *vs.* never considering it.

Implementation — in the forth phase, the individual incorporates the innovation into ongoing practice. At this stage, re-invention[2] can occur. The implementation phase ends when the innovation has become a regular part of current operations.

[1] The peers must not be homophilous with respect to their technical grasp of an innovation, or no diffusion can take place.

[2] Rogers defines re-invention as the "degree to which an innovation is changed or modified by a user in the process of its adoption and implementation" [Rogers, 2003, p. 180].

Confirmation — finally, the individual may confirm the decision of adoption (to reduce dissonance), or reverse it (discontinuance). This stage is not exhibited by all adopters, but it can also take place long after adoption has occurred, when the individual ceases to use the innovation, gradually discontinues, or replaces the innovation with a better solution.

While developing the time element, Rogers also identifies the source of information as a trait of the communication process that influences the success of a diffusion [*ibid.*, p. 207]: cosmopolite channels transport information into a system, whereas localite channels are contained within the system. This characteristic overlaps functionally with homophily, a feature of the second element of diffusion, which may be the reason why Rogers did not include cosmopoliteness in his framework.

B.1.4. Social system

The fourth element of diffusion deals with the social system in which a diffusion is taking place and is divided into the following aspects [Rogers, 2003, p. 23ff]:

Structure and communication — applies to both, the social structure within an organisation (pecking order), as well as the communication structure (interaction networks). A communication structure usually (and frequently) grows between homophilous people to satisfy their social needs; it mostly relies on interpersonal exchange. The hierarchical organisation of a system, which is often definitive of the system's social structure, allows for the prediction of behaviour to a certain degree, given that individuals at lower ranks are expected to carry out orders issued by those higher up. This flow of command and the separation of responsibilities between levels serve to decrease uncertainty surrounding the process, which can benefit diffusions; communication down the hierarchy can be either interpersonal or public, whereas it will be almost exclusively interpersonal upwards [*ibid.*, p. 337ff].

Norms — refer to the established standards among members of a social system and relates to the compatibility attribute of an innovation (see appendix B.1.1): unless an innovation squares with the norms of the targetted social system, its diffusion will be less successful [Rogers, 2003, p. 26].

Opinion leadership — is the "degree to which an individual is able to influence other individuals' attitudes or overt behaviour informally in a desired way with relative frequency" [*ibid.*, p. 27], a status which is attributed by the respecting followers rather than determined by structure, and does not have to be explicit. Opinion leaders are thus at the centre

of interpersonal communication networks and maintain this position through technical competence, social accessibility, and conformity with the system's norms. They are generally more innovative and cosmopolite, and enjoy a higher social status than the average member of the system. Nevertheless, they are not the most innovative members of the system, who are often perceived as deviants and attributed low credibility [*ibid.*, p. 26]. Obviously, opinion leaders do not have to agree with each other: an opinion leader could also reject an innovation.

Change agents — are individuals who influence clients' innovation-decisions in a direction deemed desirable by a change agency, whether the goal is adoption or rejection [*ibid.*, p. 366]. They are usually professional and heterophilous, which might limit the effectiveness of their communication. For this reason, change agents sometimes employ aides, who are less professional and more homophilous, to facilitate communication with the client.

Types of innovation-decisions — Depending on the nature of the social system, three motivations can drive adoptions. In general, these are to be seen as a continuum, and diffusions can lie anywhere between these motivations:

optional/voluntary — the decision to adopt is made by the individual at his/her own discretion and independent from the other members of the system.

collective — the adoption decision is made by consensus among the members of the system.

authoritarian — the decision is made by a few at high levels of the social structure and the innovation must be adopted by all members of the system.

Decisions made by authority are usually adopted the fastest, collective decisions the slowed [*ibid.*, p. 29]. Given that in those two cases, the individual still has a decision to make (to go with the flow *vs.* rebel against it), a forth type of decision is inherent: contingent decisions, made by the individuals only after a collective or authoritarian decision has been made.

Consequences of innovations — Rogers attributes the study of consequences an ever increasing importance in diffusion research [*ibid.*, p. 440] and differentiates between three dimensions of consequences of diffusions: desirable/undesirable, direct/indirect, and anticipated/unanticipated, and these usually go together in that anticipated consequences are usually direct and desirable, while unanticipated ones are undesirable and indirect. These

dimensions are per consequence, not per diffusion; any given diffusion will have a number of consequences, some of which may be desirable, others less so [*ibid.*, p. 442ff].

Another distinction of consequences is between form, function, and meaning; while the first two are more readily assessable by diffusers and adopters alike, a grasp of the latter is only really available to the adopters themselves [*ibid.*, p. 451].

B.2. Wejnert's integrated model of innovation diffusion

The following is a slightly more thorough description of the variables of the framework proposed by Wejnert [2002], which was introduced in section 3.2.2.3.

Characteristics of innovations

public versus private consequences — This variable deals with the spatial and temporal contingency between source and adopter, including aspects of communication, coercion, and peer/social pressure.

benefits vs. costs — Costs can be monetary or non-monetary, direct or indirect. For instance, learning time is a direct, non-monetary cost, while possible infrastructural changes and delays would be considered indirect costs.

Benefits are not explicitly mentioned in the model.

Characteristics of innovators

societal entity — The size of target group for a diffusion, as well as the strength of social ties affect the nature of the diffusion process.

familiarity with the innovation — People are naturally risk averse and cautious towards the new. Social networks, opinion leaders, and the media come into play here to increase familiarity, which drives adoption. Sources closer to an individual (and thus more subjective) are usually more effective.

status characteristics — An individual's status determines the likelihood of adoption. Lead users [*cf.* von Hippel, 1986] adopt early and drive others to adoption, while those with lesser social status and less prestige to lose might adopt controversial innovations.

socioeconomic characteristics — This category includes variables of the individual, such as education level and cosmopolitanism, as well as variables relating to the collective, such as technological advancement. Labor market practices, and the consideration whether economical decisions are centralised or market-oriented are part of the latter.

position in social networks — Network interconnectedness (size, distance, frequency, level of privacy), helps spread adoptions, as well as pressure to conform to peers in tight networks. Structural equivalence of members modulates homogeneity of behaviours, and may also activate competition [*cf.* Robertson and Gatignon, 1986].

personal characteristics — Self-confidence, as well as independence, readiness to take risks, and exposure of individual success drive adoption rates; collective interaction and collaboration also have a positive effect.

Environmental context

geographical setting — Geographic proximity affects the frequency and level of communication and facilitates spreading. Population density is closely related.[3]

societal culture — Local cultural values and norms can account for inertia effects, slowing adoptions, and increase perceived costs of adopting an innovation. On the other hand, traditions provide homogeneity.

political conditions — The political situation can have dampening effects on adoption rates (promotion of local technology, patents, censorship). The author does not identify positive effects.

global uniformity — Institutionalisation, world-wide spread facilitated by internationally active corporations, and global communication channels and media constitute this variable.

[3]The literature generally speaks of geographic proximity, but that term does not make much sense in an Internet setting. It is arguable, whether the flow of information and thus the rate of adoption is actually *higher* in Internet-connected groups. However, geographical proximity also suggests cultural proximity and thus homogeneity. While a group such as the Debian developers is rather homogeneous, the impact of cultural difference to the spread of ideas might be significant.

Appendix C

Additional information on the research approach

C.1. Insufficient approaches to reducing the data set during Delphi round three

I entered the third round of the Delphi study (see section 6.2.6) with a massive body of data, which I needed to reduce to be able to manage it, and to prevent overloading the panelists. I considered the following three techniques, but found them inadequate:

C.1.1. A top-down approach to categorisation

Many of the statements related to each other to form clusters. I deemed it useful at first to identify a set of clusters that were considered unanimously relevant by all participants. One strategy towards that goal was to identify a larger number of clusters, and to refer to the panel to identify those of relevance.

As a first step in handling 281 statements, I adopted a top-down strategy, and created a number of higher-level categories, in part from input I had received from participants during the second round [Russ Allbery, round two[1]], in which I called out to the participants:

[1] <87wsep9a04.fsf@windlord.stanford.edu>

> If you do think you see a structure, or you feel the urge to regroup the statements,
> I am definitely interested in your ideas.

The six categories were:

- Technical issues
- Communication
- Network effects
- Management challenges and duties
- Personal matters
- Social system

Most of the 281 statements related to more than one group and it was mostly arbitrary to pick which one. Therefore, I let statements belong to more than one group, but this seemed to increase the size of the data, rather than decrease it through clustering. At the end, each group had more than 50 statements attached, and I had to further subdivide them.

Unfortunately, this procedure did not lead anywhere, as I found no sub-categorisation that didn't end up too broad ($>$ 20 statements) or too specific ($<$ 5 statements). Furthermore, I found frequent overlaps between sub-groups of different categories, which led me to conclude that this path led nowhere.

C.1.2. Participant clustering

Unable to find a categorisation that could reduce the complexity of the data and make the volume more manageable, at least not without introducing significant bias, I considered to let the panelists categorise the statements in round three, but this approach fell short on two counts.

First, processing 281 statement in an hour would mean that the participant had only 12 seconds to read and understand the statement, put it in context, and fit it into a categorisation scheme, which is being evolved at the same time.

Second, I would expect the output of such an approach to be 21 categorisations. I am unaware of and could not find any reference to a strategy to consolidate potentially disjunct categorisations into a single, comprehensive representation. I briefly considered the approach

taken by Paul [2008], but discovered that the only reason she was able to engage Delphi panelists in a card sorting exercise is because she built consensus iteratively, and linearly without feedback. Since I had to question the worth of consensus built in this way, I discarded this method.

C.1.3. Statement show-down

During discussions with colleagues, I came across the "magical number seven plus or minus two", an estimate of the limited capacity of working memory [Miller, 1956]. It was not plausible to confront panelists with the whole set of data, and thus I explored the idea to present a participant with seven statements and ask for the most important one. This could be implemented dynamically (on the premise that the participants would not all take part at the same time, which could be incorporated into the design).

However, such an approach would require departure from the e-mail medium (*cf.* section 5.5) Furthermore, I could not find a study exploring the validity of such results, nor an implementation of this technique; writing, testing, and debugging an application to implement this strategy would have been well outside of the scope of this study and could have potentially introduced further delays of several months. Nevertheless, I reflect upon the choice of medium in section 8.4.3.2.

C.2. Emails sent during the study

C.2.1. Panel selection

I sent the following e-mails as part of the participant selection phase of the research, described in section 6.1.5.

Listing C.1: *Snowball sampling: the initial call for nominations.*

```
Subject: Discussion panel on Debian packaging techniques

[GREETING] [FIRST NAME],

I need your help in assembling a group to discuss Debian packaging and
package maintenance techniques. My goal is to streamline package
maintenance and help improve the scalability of our project. At this
stage, I am trying to find as many valuable members for such a group as
I can, such as yourself. I am asking for 5-15 minutes of your time, and
```

```
I would appreciate a response by 5 October 2008.

[SOMETHING PERSONALISED]. I am sure you are aware of some of the
different packaging techniques that are (or have been) floating around
in Debian, and perhaps you've been part of discussions about them. You
probably know a few people who have interesting things to say on the
topic of why some techniques have been widely adopted while others have
not, because they work in teams, employ cool tools, have good reasons
to keep using the same old tools as always, or for some other cause.

Could you please reply with names of people you think would be good to
talk to about package maintenance techniques in the Debian Project? If
you can, would you also write a bit about why you are nominating each
person?

Feel free to include yourself if you would like to be part of such a
discussion group!

I would appreciate your nominations no later than by 5 October 2008. If
you cannot make this deadline, it would be great if you could let me
know.

People you suggest will become candidates for the discussion panel and
might receive an invitation e-mail. If they chose to accept, I will
compensate them for their time and effort; if they decline, I shall not
bother them further. I will not disclose who nominated whom.

This effort is part of my Ph.D. research on the use and adoption of new
tools and techniques in the Debian Project. If you want to read more
about my research, please visit http://phd.martin-krafft.net. For your
convenience, I have attached the bits of the web page detailing my
motivation and my research approach to this email.

[OPTIONAL POSTSCRIPTUM]

Thank you very much,
[...]
```

Listing C.2: *Soliciting frequently nominated people to apply to the panel.*

```
Subject: Apply to join a discussion panel on Debian packaging

[GREETING] [FIRST NAME],

I am assembling a discussion panel to find out why some packaging tools
and techniques are widely adopted in the Debian project, while others
never gain popularity. We will explore the topic from as many angles as
possible. My goal is to streamline processes in our project and help it
scale better.

In a recent survey of Debian contributors, you were frequently
nominated as having insights on the issue[SOMETHING PERSONALISED].
```

Therefore, I invite you to be a candidate for the panel! To help me select a diverse group, I ask you to reply to a few questions in this email by 20 October 2008; this should take you about 15-20 minutes.

Over the following paragraphs, I present a rough outline of the discussion process to give you an idea of what will be involved. Feel free to ask if anything is unclear. Further down, you can find the questions that I need you to answer (deliminated by ~ ~ ~ ~ ~).

The discussion will happen via email, using an approach called the "Delphi method", which is optimised to minimise the effort required from each participant. Furthermore, participants will be reimbursed for their time. You can find additional information on the approach in the attached documents. The process is as follows:

1. Each panelist receives the same question via email, and I ask them to reply to me within one week. I may follow-up for clarification.
2. I anonymise the replies and check with each panelist to make sure that his/her views are still correctly represented.
3. Next, everyone on the panel receives a collated summary of the anonymised replies of the whole group. This helps ensure content to be evaluated based on merit, not the reputation of their authors.
4. On the basis of the collated summary, the panelists are then given a chance to revise their previous answers, if they want.
5. A collated summary of the anonymised, revised answers is made available, and the process is repeated 1-3 times with the same panel and a modified or new question, until consensus is found, or the degree of differing opinions has been established.

I expect the discussion to last until Christmas 2008, but there may need to be one more round in January 2009.

Everyone who participates in the discussion to its end will be compensated for their time and efforts with their choice of one of the following gadgets:

- An OpenMoko Freerunner phone
- A Nokia N810 Internet tablet
- A Thecus N2100 home entertainment station
- An eeePC (exact model to be determined)

I realise that you are very busy and that your time is probably worth more than one of these gadgets. However, please keep in mind that your participation could help our project scale better, so there is fame and glory in this! :) Furthermore, I expect the discussion to be interesting for all parties.

Since the supply of each of these gadgets is limited, please give me a list of your preferences. I will meet these preferences on a first-come-first-served basis. If you own all of these gadgets already, talk to me, since I have some flexibility.

Part of this study's goal is to help further the use of the Delphi method in open-source research by documenting each step and making all

data available under a Free licence. Furthermore, to increase the study's credibility and usefulness, I aim for maximum transparency. Thus, I will encourage each panelist to give permission to publish all responses at the end of the study, anonymously if requested.

~ ~ ~ ~ ~

For a successful discussion, the Delphi approach relies on diversity in the panel. To help me select the participants to ensure this diversity, I need you to answer a few questions. Unfortunately, I cannot guarantee a position on the panel, but I'll owe you a favour, for sure.

Remember, I need your responses by 20 October 2008. Please let me know if you won't be able to make the deadline.

Due to the study's anonymity requirements, I ask you not to tell others about your involvement with it. The Delphi method purposely concentrates only on what panelists have to say, not who they are. At the end of the discussion, every participant has the choice to stay anonymous, or have his/her name disclosed alongside the answers.

Remember that I am trying to approach the question of why some packaging tools and techniques spread very quickly, while others do not, from every possible angle. It helps if you keep that in mind when answering the questions. Please feel free to provide as much (or as little) detail as needed.

Let's start off with some questions about your involvement with the Debian Project:

- How long have you been contributing to Debian?
- How would you describe your role in Debian?
- Roughly how much time do you spend on Debian per week? If you get paid to work on Debian, please specify how much time you spend each week for your employer, and how much you work on Debian outside of your contract.
- By what means do you primarily interact with other Debian contributors?

Next, I have a few questions about your work on Debian:

- How do you spend your time on Debian (e.g. 30% package maintenance, 50% bug triaging, 20% discussions)?
- With roughly how many people have you regularly collaborated on packaging tasks in total in the past year?
- How much of your work is in teams, and how much do you work alone?
- How many of your packages are related to each other, and how many are unrelated?
- Do you use the same packaging tools and techniques for most of your packages, or do you use (or deal with) various approaches? How many of them, approximately?
- To what degree do you experiment with new packaging tools and techniques?
- Are you more interested in improving the way in which you work, or do you mainly concentrate on getting your work done?

```
Finally, it would help if you could also answer the following
questions, which are about your potential participation in the
discussion:

  - The study will end after four rounds, or earlier. For each round,
    you will have one week to respond. I expect that to take about an
    hour per round, but you might find that you have more to say and
    want to put extra time into it. How much time can you comfortably
    and confidently allocate for each of those rounds?
  - May I publish your responses to the above questions (under a Free
    licence) with your name, only in anonymised form, or not at all?
    (Your answer to this question does not have an influence on the
    panel selection.)

If you have any questions about my endeavour, or need more information,
please do no hesitate to write back!

This effort is part of my Ph.D. research on the use and adoption of new
tools and techniques in the Debian Project. If you want to read more
about my research, please visit http://phd.martin-krafft.net. For your
convenience, I have attached the bits of the web page detailing my
motivation and my research approach to this email.

Thank you very much,
[...]
```

Listing C.3: *The reply sent to each applicant to keep up the pace of the study.*

```
Subject: Debian packaging discussion panel: thanks, and next steps

[GREETING] [FIRST NAME],

Thank you for taking the time to reply to the previous round of
questions on tool and technique adoption in the Debian project. This
message confirms that you are now a candidate of the panel (but the
final decision has yet to be made). This e-mail is otherwise purely
informational and requires no action from you.

I will spend the next week going through all the emails I received. I
am going to use the information you provided to select a diverse panel
and ensure a balanced discussion. I will inform you about the outcome
on Friday, 31 October 2008.

Taking part in the discussion would require you to set aside an hour
for each of the 3-4 discussion rounds. I hope to complete the study
before Christmas. If you have any concerns about that, please contact
me as soon as possible!

I shall send out a tentative timeline with Friday's e-mail. I have a
certain amount of flexibility to accomodate for your scheduling needs.
Please let me know well in advance if there is anything I should know.
```

```
I expect the discussion to be interesting and engaging. Therefore, you
might end up spending more time on each round, at your own discretion
even if not required. I am sure you are aware of that, but I feel
better stating it explicitly. :)

At the end of the discussion, you will receive your choice of gadget
(see last message for details). I am happy to give away these gadgets,
and I expect you to take this study seriously in return, since sloppy
responses would hurt the study. I do *not* require you to go out of
your way in participating, or that you perfectly formulate your
responses. Instead, I hope that you take part with dedication, and that
you communicate your input and ideas to me so that I will know what you
mean -- they do not have to be proper English, as long as I can
understand them -- and I can always ask for clarification.

I speculate the outcome of the study to be of benefit to the Debian
project, maybe even Free software and other volunteer endeavours at
large. Since I assume that this is also in your interest, I am sure you
understand where I am coming from.

Thank you very much for your attention,
[...]
```

Listing C.4: *The message sent to panel candidates who did not get a position on the panel.*

```
Subject: Debian packaging discussion panel selection

[GREETING] [FIRST NAME],

I've received an unexpected, overwhelmingly high level of responses to
my questions about the Delphi discussion panel I sent out a while ago.
Before anything else, I have to thank you all for the wealth of
information you provided, and the amazing dedication you put into our
project!

It was very difficult to pick people for the discussion panel. I spent
four entire days digging up additional information and talking to
people. But at the end of the day, I only had one sponsored position on
the panel for every two applications I received. I regret to inform you
that I could not give one of those positions to you.

However, you are still welcome to participate in the discussion in any
of the following ways:

  - as a substitute for any panelist who drops out before the second
    discussion round, provided that you have a similar profile;
  - as a regular panel member without compensation. There are some
    panelists who claimed that they would participate without the
    compensation, so we could attempt to match you up if you wanted a
    compensation more than them, but I cannot promise anything.
  - as a "fly in the room", listening in and helping prevent me from
```

```
    making stupid mistakes and/or introducing bias into the study;

For your information, I have attached the e-mail I sent to the panel
members to start the first discussion round. If you want to
participate, please drop me an email so that I can send you a
personalised copy with its own message ID.

As before, please respect the anonymity requirements of the study and
refrain from talking about it or the selection process with others. I
am sorry that this sounds somewhat clandestine, but it is an important
feature of the study.

I very much appreciate the time and effort you have put into this. As I
said in the invitation message, I owe you a favour and sincerely hope
that one day, I can be as helpful to you as you have been to me!

Have a nice weekend!
[...]
```

C.2.2. The Delphi study

Listing C.5: *The message sent to the participants, welcoming them to the panel and presenting the first round question.*

```
Subject: Debian packaging discussion panel: round 1

[GREETING] [FIRST NAME],

I am pleased to inform you that I selected you as panelist for the
Delphi discussion on the adoption of packaging tools and techniques in
the Debian project.

There were two applicants for each panel position, and it took me four
days of additional research to be able to make the decisions. Those
efforts paid off and I can claim with confidence that the panel is very
diverse with representatives from most of the important
packaging-related camps in Debian. I am very much looking forward to a
productive Delphi-style discussion with you.

Welcome aboard! :)

Let me thank you up front for the time and effort you have already put
into this study, and for the exciting responses you sent in. My offer
to compensate you for your part in this study stands, but please
remember that I need you to participate through to its end to qualify.

If you don't think you will be able to participate after all, please
let me know as soon as possible.

~ ~ ~ ~ ~
```

Without further ado, let's jump right into the first discussion round. I will present you with an instruction, and **I need your response by Sunday in a week, 9 November 2008**.

Please keep the anonymity requirements of this study in the back of your head and refrain from talking to people about it. I am sorry that this sounds somewhat clandestine, but it's important that others don't know that you are taking part in this discussion so that they judge only based on what you have to say.

When following the instruction, please remember that you should try to answer from the point of view of your fellow Debian contributors and the Debian project at large. Obviously, your own experiences count as well, but please try to focus more on the project and its contributors. If you do talk about yourself, make sure to be explicit about it.

In this first round, I would like you to respond to the following:

> Debian package maintainers occasionally find new tools or techniques that could change their packaging work. At times, they might decide in favour of one technique and adopt it; at other times, they might reject a technique without further thought.
>
> While deciding whether or not to adopt a tool or technique, people normally weigh many options, and take various points into consideration which influence their decision.
>
> Please describe at least six such influences you have witnessed in the Debian project, and which you expect to witness again on future occasions.

It is very important to communicate exactly what you have in mind. Therefore, please provide enough details, examples, thoughts, corner cases, anything. Go ahead and write down whatever comes to your head! Do not worry too much about language; this is not an essay, this is a braindump, and I will rephrase your thoughts and anonymise them anyway (under the supervision of a few sworn helpers), as long as I can understand what you are telling me. Make sure that you include everything that is relevant. If you wish, I am also happy to call you and listen to what you have to say.

Feel free to ask some of your peers for input, but try not to make it obvious that this is done as part of this study.

If you have any questions, please don't hesitate to ask, in an e-mail, or by *private* IRC message (I am madduck on irc.debian.org).

Please remember that I aim to obtain your consent to publish the important data at the end of the study under a Free licence for credibility and usefulness. I will consult you when the time has come.

Please return your response to me *by Sunday, 9 November 2008*.

```
As soon as I have received all replies, I will take a few days to
analyse, anonymise, and collate it with the others. I expect to present
you with the result around 14 November and give you two days to revise
your answers in the light of what everyone had to say. This is optional
and allows you to clarify your response, change your mind, or be
persuaded, anonymously. We will start the second round, which will
follow a different format, around 17 November.

Please keep me informed on any scheduling concerns you may have!

Thank you very much for your participation. Have a great weekend.
[...]
```

Listing C.6: *The message sent to the panelists at the start of the second Delphi discussion round.*

```
Subject: [Delphi] Second round discussion

[GREETING] [FIRST NAME],

I am looking back at a tough, but very exciting week. You and the other
panelists have sent in pages and pages of very thoughtful and
interesting responses to my first round question on influences to
Debian package maintainers' decisions for or against new tools or
techniques. I am greatly in your debt and would like to shout a big
"thank you" to everyone of you!

It has taken me way longer than expected to process all the responses,
but I feel that the extra time I allocated for it was worth it, and I
hope that I managed to meet your expectations with the result.

What you will find in this (admittedly very long) email is a condensed
collection of the input you have collectively submitted. I've done my
best to collate and merge everyone's thoughts, and grouped them into 15
chunks of related statements.

Your task for the next round is to go through and read this collection,
and to comment whenever you disagree or would like to set something
straight. If you leave a statement uncommented, I will assume that you
_generally_ agree. I will not hold you to it, obviously, but I am
trying to encourage you to feed back qualifications and refutations of
those statements, unless you think along the same lines.

I kindly ask you to return your reply to me by Monday, 1 Dec 2008.

I realise that it is probably going to take you longer than the
hour-per-round I initially advertised. I did not expect this. However,
I hope you still continue to participate. The third round will take
substantially less time, and if there will be a fourth round, it will
be optional. Plus, I don't think I am promising too much when I claim
that there is a lot of very interesting stuff in what you all had to
say.
```

```
If you foresee scheduling conflicts, please get in touch with me. I
have a certain amount of flexibility to accomodate for such cases.

The approach I took to this summary reduced ~3500 lines of pure text to
around 1250, so it's only logical that not all of your statements will
appear verbatim. However, I exercised utmost care not to drop anything
that added value to the whole; often, more than one panelist said the
same thing, in which case I only kept one formulation. If you find you
are missing a statement you've made, then please accept my apology and
let me know.

I've refrained from adding glue text, or from making the quotes more
consistent, mostly in the interest of brevity, but also because it
would have taken me another week. Therefore, you will encounter
statements made in the first person ("I think that..."), but those are
not my opinions, they are the opinions of one of the members of the
discussion panel. Also note that a paragraph can be made up of
statements by different people.

In fact, I have tried the best I could not to let my opinions flow into
this document at all. Obviously, bias can never be fully elminated, but
you can rest assured that I never explicitly amended or engineered
statements to make a point. After all, I have a truly genuine interest
in getting a valid result from this study. If you read about the
greatness of Git, keep in mind that it's not I who's talking! :)

I've tried to anonymise as best as I can, to allow you to judge on
content, not reputation of the source. You can improve that experience
for yourself by not trying to guess.

I've split the responses into 15 chunks, such that statements within
each chunk are related. Please bear in mind that I did this only for
readability and manageability, not because I am trying to hint at some
higher-level structure.

If you do think you see a structure, or you feel the urge to regroup
the statements, I am definitely interested in your ideas.

I added fold markers to the text so that when you reply with Vim or
Emacs, you should be able to fold away everything but the current chunk
of statements you are working on. Please Refer To Folding Manual of
your editor to find out how. Emacs supposedly turns folding on
automatically, while Vim needs to be told. There's a modeline at the
bottom of this email, which should work unless your signature gets in
the way.

You can trim quoting in your reply to your liking.

-------------------------------------------------------------------------

** Exposure, visibility, marketing, publicity, availability {{{

There are a large number of small packaging related scripts/tools/
tricks that one might not know about. Once you run across one, you just
```

start to use it because it's so obviously an improvement. Often it's just a matter of discovery.

I was explaining to another developer that building with pbuilder is very clean but painful and slow, and he just told me to use cowdancer. Two days later, I was using it.

The daily job consumes too much time to keep up with latest development issues, and new tools remain "hidden" for a large number of developers. One challenge we have is how to get information about new tools and techniques to the people.

Our communication media are numerous. However, the time that each of us can put on this is not extensible. So, we probably all tend to focus on a subset of those media, and probably tend to form some smaller well-connected groups that might not well interact with each other. Despite the development of communication media, I'm not sure whether we are more closely connected than before. Also, I feel like Debian is getting older. The advent of new, more "hyped" distros tend to attract newcomers more easily than we can. Our contributor base is ageing and spends less time to look around for new stuff, which slows knowledge spread, or provides for better filtering. :)

Marketing definitely helps spread the word on new ideas, or tools, or simply new features. Blogging about it on Planet and mentioning it on the #debian-* IRC channels are good first steps. Presentations at Debian conferences or events with a significant-enough number of highly skilled Debian contributors are excellent for discovering tools and technique.

Advertising influences us, but it's more about giving a first glimpse of a new tool technique. After that glimpse, other factors take over.

Looking at what is actually being used in packages can shed light onto new tools and how well they work for the maintainers. Some tools are the type that you can easily notice when working on a package (e.g. build-dependencies, version control systems), others don't show up in the package's source, those are harder to detect (lintian, piuparts, *-buildpackage).

In general, working on other people's packages (for instance when doing an NMU) seems a good way to learn about new stuff and get your hands wet. Developers often learn new techniques through exposure to what other team members use, either through necessity or by recommendation, and new tools for collaborative development are often forged through experience in teams. When packages are adopted, or when developers join teams, they are often forced to make use of multiple techniques rather than living in a monoculture, thus finding out about and giving new tools a chance to thrive if they can get a foothold in a single area.

Sometimes it is necessary to push a new technique to make it successful, such as providing patches to move to a new infrastructure as a way to market very effectively something new. (The patch not only helping the switch, but also teaching maintainers about it.)

People will use tools or techniques that they can readily obtain and apply to their work. One that's significant for Debian is that some developers will tend not to use tools that aren't available in stable, or at least for stable from backports.org. I think the use of stable as a development platform is a lot more widespread than it appears from reading the project mailing lists (although I could be wrong on this), and while developers have access to sid, a lot of them also spend a bunch of time in stable.

This all goes double for anything that happens as part of the package build and therefore affects the build dependencies. For example, many packaging teams have a policy against introducing debhelper dependencies that can't be satisfied in stable, which I'm sure is currently limiting the speed of adoption of the new capabilities of debhelper 7 until the release of lenny.

}}}
** First impression, gut feeling, heritage, prejudice {{{

It should not take more than a few minutes to understand the overall rationale of a new tool, as well as its principles. This is helped by a good and concise description, targeted at all potential users. Often, the designers of our tools tend to assume that all future users will have their knowledge, skills and wisdom, which is a fallacy. The availability of quality documentation helps adopters to get a good first impression.

In one project I work on they attempted to move to DVCS style package management to make team coordination easier. Unfortunately the combination of the size of the packages, the DVCS chosen, and the infrastructure used to host the repository produced a system that was so slow it was extremely painful to use. This negative initial experience has made me very reluctant to use that system again even though many people describe it as "much faster now".

At first, gut impression can be very important. If the first impression of something is that it's causing someone else problems, or isn't interesting, or exciting, or useful, or has an obvious problem, it can take a long time to get over that initial impression, even if it was unjustified. Even knowing that this is likely to happen, developers can take significant amounts of persuasion to reassess the first impression against later releases or problems, depending on other factors.

The first time I try out a new system, am I able to do anything (even something silly) with it in the first 10 minutes of using it? If not, I'm likely to leave it there and try it again "eventually", but I'll tend to pick it up with a rather pessimistic approach, given the previous failure. A bad "first 10 minutes" experience has sometimes been undone by finding an article/blog post/whatever that shows the right way to think about the tool.

Sometimes I see small ways to do things that are just so obviously

better that it doesn't even take any real thought to change, but one follows gut instinct. Group-maintenance and low-threshold-NMUs are two ideas I picked up basically after reading about them and thinking, "why not?". After all, I trust the rest of the project to be - at least on average - better skilled than myself, and careful enough when messing with other people's packaging.

People look at the goals of the new thing, and the design, and to some extent, its implementation. We look at whether the plan/idea for something was intially discussed (ie, on a mailing list). In such a discussion, the obvious flaws and shortcomings tend to come out. It's illuminating to see whether those flaws get dealt with. If not, that's obviously not a good sign. But it's also bad if the original design gets a lot of stuff tacked on the side to deal with specific peoples' pet concerns. The condition of pre-alpha code and the design decisions made by that point in development are factors in assessing the judgement of the developer and the likely condition of later releases.

Teams and developers who have an existing problem that the tool is supposed to fix may well have been involved in discussions around the design of the new tool. How well the tool matches up to the results of those discussions will predicate how the tool is evaluated at release. The opinions of such developers, when published, will also have strong effects on wider adoption.

An important considerations is whether the tool can do what was claimed and most developers are willing to put up with a small amount of awkwardness in the initial setup, particularly if the problems are related to other packages that need to be modified to work alongside the tool. The amount of tolerable inconvenience is inversely proportional to the amount of time the tool has already existed or been in development and approaches zero if the tool has already been part of a stable release. It may seem unfair to determine the adoption of a tool by the effectiveness of underlying packages but if there were likely to be such problems with the dependencies, the design of the tool itself is often regarded as flawed. Perceptions arise that the design should have been changed to take account of those problems, possibly changing the language of the tool concerned - before release. Tools that simply do not work but have a good design and responsive developers with a good reputation, can still be adopted if the problems are fixable, expected or due to other packages.

When a respected member of our community announced something I've always been looking for, it didn't matter how much time it took me to learn it, get it running, adapt it to my needs, I "knew" that it was what I needed, and that was enough to keep me going. At first, it lacked a feature I considered essential and I had to write it myself. If I had not been convinced that the tool was the right thing for me, I wouldn't have wasted the time to make it work better for me. In that lies the core of how expectations greatly influence uptake. Maybe it wasn't just features that created expectations, maybe it was also my high level of respect of the work of its author.

It also seems to be a bad sign if it starts off too perfect, with every

```
concern anticipated. We seem to like things to start off hairy and get
smoothed off through encounters with other people. Sort of like how a
Wikipedia page that's only had one editor tends not to be as trusted as
one that might have started out very bad but has had a lot of people
work on it. If a tool consists of many components, judgements are made
about whether the developer has tried to do too much too soon or to get
the basics implemented before adding corner cases and other components.

Personal opinions about the author of a tool/technique probably have
more of an influence than we'd like to admit. I think we're pretty good
at accepting something whose author is unknown, though it can be hard
for them to bring it to our attention at first. And we don't tend to
uncritically accept work by someone we admire. Or at least, if we do,
someone else will not admire them, and criticize it instead. But, we
don't seem to be very good at recognising good ideas and good work from
people we dislike or have written off for past failures.

Tools can gain a perception of bias from the initial development
targets of the project or affiliations of the main developers. Why and
by whom the tool was developed can become more important than the
actual performance of the tool. If the heritage matches the interests
of the developer, significant flaws can be ignored in the hope of
future fixes. If the developer has no wish to be involved in the
projects that gave rise to the tool, the tool can be dismissed as
"specialist", even if some functionality exists to broaden the support.

------------------------------------------------------------------------
}}}
** Motivation, curiosity, need {{{

I have seen people resisting migration to Git. Subversion just performs
sufficiently well for them.

There is a group of less active DDs. This includes many people in NM,
many people with very few packages in which they are interested for
reasons outside of Debian, and not very invested in the project at
large. Those people tend to use what is simple and can get the job
done, without thinking too much about later maintenance.

Core developers use a tool because it has intrinsics qualities, because
it's fun, sometimes because it's new, but mostly because it works well.

When there's no existing tools or methods to solve a specific (or new)
problem, the expectations and requisites of people are going to be
really low, so something that's slightly better than their current ways
but suboptimal might just become popular. (I guess cvs at the time
could be a good example of this).

Some people try out new tools just because they are new and while
trying them, they imagine how those tools could (further) improve their
Debian work. Or they feel like taking up the challenge of something
new. Or they find something written in your favorite scripting language
and want to hack on it to improve their skills, even if it has rough
edges.
```

Sometimes you need something to ease the pain, because else you would quit. New needs can lead to new tool adoptions. It's likely that such tools would be short-lived in their first form, and that they will be rewritten to comply with the KISS principle and similar other requirements.

Experimenting with new tools is just thrilling. There's this hunger for new concepts I just can't really explain. This is an attitude you'll find everywhere among people passionate about computer stuff. Some will get crazy about hardware, some about new algorithms, some about new tools, it all depends on your center of interest, but in the end, people crazy about computers will always exhibit that kind of hunger at some level.

There are people who always surf the bleeding edge, who just like what is new and shiny. Those people are very important because they are an influence to almost every other group, they are pioneers and invent or improve concepts. The danger is that such people turn blind and may not realize they're in love with a tool/concept/whatever, hence are not objective about it. This can hurt a lot in a team. Also, those people sometimes stick with a tool and get in the way of others, because they pretended for months that their tool rocked and it would make them look like fools to let go.

Making things even simpler is not accepted by everybody.

}}}
** Perceived advantage, cost-benefit, efficiency, productivity {{{

The tool should bring immediate benefit to its user. I think this is a fairly shared feeling. If I have to invest too much time in learning the tool/technique before I can really benefit from it, I'll probably be lazy and just look somewhere else, or ignore it. Anyone trying something new will form an opinion on how difficult the process was. And if it was difficult, this will put up a mental barrier that will be difficult to overcome.

People will choose to switch to a tool if they perceive some benefit over what they currently use. I say "perceived benefit" as they may concentrate on one aspect, or look at one tool more favourably due to other factors. I believe that a cost estimate is made, but I don't think it is always a fully considered estimate. One benefit may be emphasised and some of the costs ignored. When a discussion happens in public this effect can become stronger, with some inflating the estimation of the costs, and some playing down the costs and pretending that the new tool will solve world hunger.

The improvements have to be significant, in order to justify the efforts. If the inconvenience of learning it is more than the convenience of adopting it, there is no point. Yet, if everyone tells me it's worth it, then I'll be happy to go through more effort in learning it

I think the only good way to understand the costs vs benefits is with tutorial-like documents which shows how is the work-flow with the tool you are migrating to. In simpler cases, e.g. migrating to machine-parseable debian/copyright files, the migration time is likely to be proportional to the length of debian/copyrights which should be migrated. The availability of tool which perform the migration, for those cases where it make sense to have a mechanical/ automated migration, will help, but can also mislead, e.g. when the tool creates a subtle mess that has later to be cleaned up.

It's a lot harder to quantify time savings as a result of adopting a tool. However, that's not always of relevance. Sometimes, time is important (e.g. deadlines), but I (love to) think that DDs also have other motives.

Active developers often maintain several packages, and when choosing to switch to a new tool or technique they probably want to do that on all their packages. Hence, there is a multiplicative factor in the migration cost.

The tool must not get in the way. People usually don't like to have to think about how their tools work. They must come naturally. If they don't, it means lower productivity.

I expect that just about everyone maintaining a package in Debian will consider adopting a tool or technique that helps them automate manual labour.

I'm even ready to use "bad unclean hacks" as long as they help me save time and be more productive for instance.

All the developers work with limited time and whenever something can improve their workflow and thus their efficiency, they have to try it. This is one of the reasons why a winning technology always ends up with sister tools to optimize for some specific use cases.

Reducing the maintenance cost of packages you maintain is an incentive to move to new packaging techniques. For instance, cdbs helps keeping debian/rules shorter and mostly avoids repeating yourself across packages which are similar and would otherwise use boilerplate. quilt helps managing large stack of patches across new upstream releases. Not everybody loves quilt or cdbs, but they effectively lower the time spent on the job for people who use them, and this is an incentive for using them for these people.

I don't believe we really have an alternative to debhelper; sure some packages do it "by hand" calling the underlying dpkg-dev binaries/layer directly, but we don't have, say, a pyhelper which would provide the same services as debhelper. However, you can move 90% of the debhelper packaging bits into cdbs, and that's a service to the packager which allows focusing on the difference between a standard package and the current package, hiding most boilerplate.

I think it actually boils down to the very same thing: efficiency, save time, help do more in less time. And this happens at many levels, e.g. look at dpatch vs. quilt: quilt manages the patches in-place, dpatch expands the orig, lets you do your changes and then diffs. For small packages the time difference between quilt and dpatch is not big. But is obvious that dpatch does a lot more work than the work done by quilt, and this will bite with larger packages. Even though quilt requires an additional step - adding files to a patch (avoidable if you use 'quilt edit', which means slightly changing way of work, i.e. an additional "cost") it seems to win.

Some VCS is speedier than others. Avoiding debhelper might be speedier than using it, if you happen to know exactly what you need done.

To me the fact that the tool fits with what I do is part of its technical *merits* and benefits to some extent. I'm ready to change my workflow if the new one isn't exagerately different from mine, but still shows improvement: better efficiency, faster work, more elegant packaging, whatever, as soon as there is improvement in some area.

Aesthetics is part of the efficiency. I'm more prone to be efficient and willing to modify something that pleases me than something horrible and broken.

If the tool comes across as yet-another-foo, any problems that do not exist in foo will be significant barriers to adoption. If the tool is so distinctive that it distorts normal workflow patterns or requires adjustments to long-established patterns, the perceived "quality" of the tool will be diminished. As few developers will use any one tool in precisely the same way, this can be a difficult barrier to remove from the development perspective - especially if the goal is for the new tool to be a lighter version of an existing tool.

Something like pbuilder is the only reliable way to make sure the build dependencies are complete. I think at least one pbuilder run should be part of the development cycle of a package. Besides the fact that it is also not as widely known as it should be, there are people out there who consider pbuilder too cumbersome (setting up pbuilder, updating over slow connection, longer build process). I'd consider this a weak excuse but it exists.

Sometimes, someone has to simply shoulder the burden for the benefit of everybody. Curiouly this tends to be done as part of Debian QA, at least when it has to do with infrastructure.

In a project I work in, I was asked to submit my changes in a particular way. I found the experience very suboptimal for me as it imposed what seemed like a very large overhead for little benefit. This was the general consensus in the team too. This was tried for one release and not generally repeated.

}}}
** Quality, transparency {{{

Build systems are a way to help keep the archive closer to policy without that much manual developer interaction. It helps moving to some new technique when it eases policy compliance -- not necessarily because the old technique is forbidden by policy, but because the new technique helps comply with policy, e.g. the Python policy and support tools.

lintian noticing bugs early is probably unanimously considered an improvement in quality, at least as long as there are no false positives.

Choosing one tool/technique over the other can influence how easy it will be in the future to make mass-changes to related packages without requiring manual intervention for each package. A lot of tools have been developed to centralize build-behavior, dependencies, and stuff like that in single packages, so that changes in those few places can affect a lot of packages at once to lessen the burden of coordination, especially when maintainers go MIA and packages get neglected. Clusters of related source packages should factor out the behavior that affect their integration into Debian. That's the theory, in practice, the factorised package can break stuff, and its APIs might not be expressive enough for the behaviour you need.

Abstracting a tool for reuse across packages might make it more difficult to follow, or perhaps less reliable.

Debmake's greatest liability was condensing too much functionality into the single debstd command, which was deemed obscure/opaque. The design of CDBS is often accused of opaqueness, getting in the way of debugging, or otherwise interferes with fixing problems. Yet, making packages easier to debug may clutter the packaging code rather than keeping it simple.

In programming we frown upon code duplication. I've never understood why duplicating stuff in different packages (and the debhelper invocation sequence _is_ a duplication) should be treated differently; IMO it should not.

Even if the tool does multiple things, if it's transparent about what it's doing and the same tasks can be duplicated manually, that helps a lot in picking it up. For example, one can perform all of the operations that git-buildpackage performs, back them out individually, and understand each component, which makes it easier to become comfortable with the individual steps and then let git-buildpackage automate them. This is important for debugging: if the script breaks, one wants to be able to pick up the pieces by hand without having the tool be a mysterious black box.

Moving part of the maintenance burden of a package somewhere else usually comes with a loss of control. There is a tendency among DDs to be control freaks. They don't like what they don't understand.

Using tools to simplify the work by hiding many details and automating

```
many tasks is good, but not knowing the details is bad. Any tool makes
a compromise between those two aspects and not all developers find this
compromise acceptable.

debhelper might not be needed, but it helps understand the steps,
especially for others doing an NMU. Same goes for keeping a certain
naming structure for files, or for VCS branches.

Following the build of a package with CDBS is not always easy, and it
can lead to maintaners not fully understanding what they are doing.
Those that use without understanding what's going on are dangerous to
the quality of our project.

I have seen people leap at tools that appeared to shortcut the need to
learn things to do Debian packaging. While there are certainly aspects
of packaging that are amenable to increased automation, there is an
irreducable minimum of knowledge needed to produce reliable, policy
compliant packages. People who don't care enough to learn such things
are looking for shortcuts and sometimes getting solutions that at least
initially appear to work. Those people don't grok the danger of such
tools at time. They think rather short-term.

------------------------------------------------------------------------
}}}
** Sustainability, future-proof investment {{{

We don't want to adopt tools that will become obsolete. The quality of
documentation, including of its API and internals, examples, and e.g.
the level of care at which the tool author or promoter explains the
tool or technique, are all signs of higher life expectancy.

Reaction time of the maintainer, and turnaround time of bug and feature
reports are also relevant. The BTS can be a good source of information:
age of bugs, type of bugs, number of wontfix bugs or unattended
patches. Nobody wants to start relying on a tool suffering of the "dead
upstream syndrome": crowded BTS pages (both Debian's and upstream), no
releases for X years, no feedback on bug logs, ...

We know that tools can become neglected and we know that we cannot take
over the maintenance of all tools we need. Hence I've the feeling we
all have a notion of "trustworthy-ness" of our tools and of how much we
can rely on their authors to not become MIA.

Tools that simply do not work but have a good design and responsive
developers with a good reputation, can still be adopted if the problems
are fixable, expected to be fixed, or other packages' fault. Sometimes
it's not a case of "it scratches an itch now", but rather a "it looks
like it's going to scratch it".

On the other hand, implementation maturity is important. A tool or
technique with stable interfaces and relative freedom from serious bugs
over time will not require the adopter to spend time trying to track
changes. Many will agree that machine-parseable copyright files are a
good thing, nevertheless they haven't yet become widespread. This may
```

be due to the perceived instability of the proposal: the wiki page always changes, there is no DEP about it, it is not mentioned in "authoritative" packaging documents (e.g. developer's reference). Stability can be the factor that make a developer choose a work-flow which is not yet popular, on the basis that it will probably become popular in the near feature.

Maintenance level is a standard evaluation point for any software and applies to tools and techniques. The point of a tool, put very broadly, is to leverage other people's work so that you don't have to reinvent the wheel. Since Debian packages change and software changes and requirements change, the tool needs to evolve. This requires an active development community. Tools that aren't being actively developed end up being more work to use, since the problems that crop up aren't spread across a large userbase and hence fixed in a way that feels almost magical.

A question developers ask themselves is how likely it is that they will be able to collaborate with other developers on a package without one of you having to first learn a new tool. All other things being equal (and sometimes even when they're not), people usually prefer to use what other people are using. It's perceived as safer, more likely to be maintained in the future or at least have a migration path.

Upstream interaction is related: maintaining patches against upstream sources can account for a significant part of the effort involved in maintaining packages. Version control system choices are likely to be influenced by what upstream is using, particularly if they're using a DVCS that will allow bidirectional merges; patch systems will be chosen to give what the developer feels is the right balance between up-front learning curve and ongoing cost when porting patches to new upstream releases.

Core developers have a more long-term view of their actions. CDBS is a bad choice for them because it addresses a short-term issue, and may become a PITA in the future. Using CDBS is short-sighted because it's really bad in many not-so-uncommon areas.

A standard data format makes it a lot simpler to switch tools. It's relatively trivial to switch between CDBS' simple-patchsys, quilt, and dpatch, because they all rely on patch files. This played a role in the discussion of using quilt vs. git in the dpkgv3 source package format.

A tool using a familiar language or which relies on familiar protocols etc. gives a perception that the tool itself will be easier to debug and problems easier to fix, even without particularly helpful or responsive developers. Equally, the choice of language can change the perception of the developer and the tool by choosing either an unsuitable language or an overly problematic language for the task at hand. Security interfaces written in PHP would need careful assessment - tools written in python or scheme/guile would dissuade me from any attempt to debug problems myself.

I think choice of language is important because some developers aren't

comfortable with the idea of their packages depending on packaging systems whose code they don't understand fluently. I believe this was also a factor (though not the most important factor) in the low adoption rate of darcs, which is written in Haskell.

}}}
** Genericity, flexibility, adaptability, modularity, KISS, Unix {{{

It's often been clear that techniques that are general-purpose (applying to multiple packages) tend to defeat special-purpose techniques by sheer force of numbers.

Firstly, developers tend not to like to apply an overly wide spread of tools, and instead prefer to apply similar techniques to everything they work on: people usually have their favourite packaging helpers, their favourite patch systems, etc., and when creating a new package will often reach for the last vaguely similar one they did. Indeed, there is a level of inertia here in that changes also seem to need to demonstrate their general-purposeness. That's the the difference between debhelper and cdbs; the former adapts to just about every need while the latter is good for those situations it was designed for, but doesn't adapt as easily.

Secondly, new developers will look for example code, and statistically general-purpose tools will tend to dominate the field.

If a tool has overcome initial barriers to adoption, many developers will postpone the final decision until a series of specific tests have been performed, exploring corner cases, reviewing the source code, checking that the more unusual options actually work and using the package in ways that the original developer may not have tested or considered (e.g. cross-building or automation). Other developers may change their opinions of the tool based on the results of these tests, and the developers performing the tests may become significant contributors to the tool itself.

autotools is flexible because you can add shell script snippets where you need them, whereas with cmake it's much harder to do. There is a dialectic process going on here: abstraction vs fine control. If a system allows to break the abstraction layer (like autotools), then it'll be less portable, but it makes it easy (or possible) to solve problems. autoconf (not automake) is quite like debhelper: it expands macros inside shellscripts while debhelper provides external scripts for common functionality, but the result is in both cases a piece of shellscript where the common code is encapsulated behind a macro expansion or a script invocation, and in both cases you can add your own scripting inbetween macros or debhelper calls.

Tools that will adapt to your current process but leave open the option of evolving that process to something more "standard" are easier to adopt than tools that enforce their method of doing things. To use git-buildpackage as an example: all of the branches that it uses are configurable even though it has defaults. That lets you use the tool

with an existing branch naming structure and choose later whether you want to rename branches to match its defaults. It helps to have knobs to tune the tool when the problem changes, and those knobs shouldn't be tied to the internal implementation.

Python's setup.py has no obvious way to be extended to support extra corner cases, and therefore to be integrated with other systems. Worse off, I recall seeing someone asking about installing files to /usr/sbin in some mailing list, and receiving an answer along the lines of: "it's not setup.py's job: if you want files installed somewhere else, move them afterwards". Compare to Git: pristine-tar was trivial to integrate with Git. But in my search on how to get setup.py to install something in /usr/sbin/, I didn't find any way of extending it that wasn't reading its source code and tweak its internals from setup.py.

Hence, I'm happy to adopt Git knowing that if I need to have a git-foobar command it'll be straightforward to write (at least as much as interaction with Git is concerned). Same with debhelper. But I'll avoid depending on setup.py as much as I can, because my perception is that the day that I'll need to extend it somehow, I'll have to spit blood and I'll enter that vicious circle where it costs me less to painfully hack it than to migrate lots of systems to something else, and so I get locked into a spiral of increasing pain.

Some tools or techniques are too specific to reach critical mass; this can happen when they are bound to too specific use cases. It helps when a tool is generic to serve more developers or packages. For example, debcheckout covers many VCSes and is useful to anybody, while we don't have a generecized buildpackage tool -- cvs-bp in devscripts, svn-bp and git-bp in their respective packages, no svk-bp etc.

I think there are many examples of tools which are reusable because they aren't tied to a particular tool/technique. For instance, pristine-tar has only git bindings, but it's applicable to any VCS / package. Or quilt; that's not specific to the way your package is laid out.

Simplicity and the Unix-ish approach of separate tools and KISS are key to reusing techniques / tools, which is a clear benefit. Many DDs adopt tools based on their merits, the more KISS the better. Debhelper is KISS: a bunch of relatively well-documented tools, simple in what they do, yet achieving something great and complex as a whole.

The Unix way, tools that do a specific thing and can be strung together with other tools that also do specific and well-defined things, seems to fly well. debhelper or devscripts are examples of tools structured that way; CDBS and yada are examples of tools that aren't. With a Unix-like toolchain, each portion of that toolchain can be played with independently without requiring a change to the whole process, as long as you preserve the interface between the pieces. I think it's particularly instructive to look at how debhelper introduces new tools: the new script shows up but doesn't require existing debhelper users to start using it, the old method continues to work, and the new script will do something well-defined and self-contained and can be introduced

```
when the packager feels comfortable with it.

Similarly, devscripts bundles small tools that perform well-defined
actions that frequently are exactly what people are already doing by
hand. The easiest tool to adopt is the one that does exactly what
you're already doing, but automates it. devscripts is particularly good
at putting together tools like that, and then slowly adding more
optional features so that people can evolve their work process into a
more and more automated process.

------------------------------------------------------------------------
}}}
** Technical excellence, elegance, aesthetics, rightness feeling {{{

Debian contributors are very keen about technical excellence (even when
they're themselves not technically excellent, that's not exclusive).
So, for some of us, the quality of implementation might sometimes be an
important factor to decide about adopting a tool or not. Nobody can
impose tools or techniques to us, no matter which deadlines we have.
Our main work is package maintenance and we want to do it properly.

We are perfectionists, and one way in which this shows is that we seem
to dislike piggy-backing behaviour or results on top of unrelated
behavior or result. The new homepage field is clearly the right way
when compared to the previous abuse of the long description. The Vcs-*
fields were quickly adopted because their inclusion was "right". The
discussion about where to put watch files (either in the package or in
a central repo) is a counter-example where we have not yet found the
right way.

The Web is something that many DD don't like. They don't live in their
browser, but in their MUA and their terminal. Using a wiki makes them
live in a world they don't like. I contribute to Debian _because_ it's
technically aesthetic. So it's rather logical that I would use a tool
that I find aesthetic and elegant.

Elegance is made of many different points. The elegance of the
concepts, the elegance of how code is written, the language chosen to
write the tool, and so on. Of course elegance is really subjective. But
technical appreciation and aesthetics are not opposed to one another.

Being technically correct is an obvious requirement for an idea to gain
popularity, but there is also the "feeling". Some people just feel it
right to use debhelper (even super-short-DH7-style) instead of (god
forbid!) CDBS. Others feel the opposite. Both camps feel that their
choice is "right". Perhaps this roots in the efficiency too -- "CDBS is
a hard to learn black box (thus learning it is nearly impossible/very
time-consuming job)" vs. "CDBS does everything by itself (saves me a
lot of time in redundant work)".

CDBS is a good example, because it was doing the right thing
(factorization), but was doing it in a bad way, or at least in a way
which has not proven to be popular among DDs. In part because it uses
Makefile dark magic which only a few understood, in part because it is
```

```
not consistent and tries to give the feeling of things happening
"magically" without maintainer control, but there's a tendency among
DDs to be control freaks. In the case of CDBS vs. DH7, I think there
are several interconnected properties that matter -- simplicity,
transparency, extensibility. In other words, can you grasp what is
going on and is it easy to influence it.

Debian contributors, particularly longer-standing ones, generally seem
to have a notion of what kinds of approaches fit well into the Debian
system. Some of the important themes here seem to be modularity,
adaptability, and a general idea of enforcing just the right amount of
policy (very tricky) and leaving the rest up to the developer.

Yada failed to gain significant traction as a packaging helper tool
largely due to failing this criterion in the minds of many developers
(although it undoubtedly also suffered from lack of network effect and
from having prominent developers advise against its use). The way it
generated debian/rules and debian/control from another file often
caused confusion and ran counter to developers' instincts, and in
general it seemed to impose the "wrong amount of policy" on what debian
/rules did.

Despite what they say about Debian and having the choice, people tend
to dislike too much diversity, especially in packaging. There are some
very strict ideas on what packaging should looks like, and some
practices simply do not comply with some kind of "packaging ethics".
Yada is the perfect example of that. It was created to simplify a
packaging system based on many independent files - It was not adopted
because it felt like a completely different packaging system, closer to
RedHat's .spec than to our debian/*.

The important themes (modularity, adaptability, enforcing just the
right amount of policy) are also noticeable in that techniques benefit
from behaving in a similar way to other "nearby" things. The fledgling
machine-parseable debian/copyright proposal is only getting anywhere at
all because it feels somewhat like a debian/control file, and that's a
popular style of format in general. Indeed, you often see people
shooting down ideas for XML file formats because you can do a similar
job more readably with an RFC822ish format, which generally seems to be
regarded as more "Debianish".

On the other hand, there is a strong tendency towards "NIH" (Not
Invented Here) behaviour, to the point where the beloved RFC822-like
format is held unreasonably high and advocated over other, better
suited formats. Or the debate between FreeDesktop Debian menu formats,
to name another case.

------------------------------------------------------------------------
}}}
** Documentation, examples {{{

Sadly overlooked for most new tools is documentation -- developers are
still the worst documentation writers. Creating *usable* documentation
is generally the hardest part of development. The importance of
```

documentation is directly proportional to the amount of divergence from similar tools or existing workflow patterns.

For tools, thoroughness and completeness of the documentation is the most important factor, I think. This includes the details of how to reproduce or fix what the tool does manually, if necessary. For techniques, clarity and communication of the mindset and methodology are more important (for me) than step-by-step recipes, although I think this varies by person. (Probably the earlier of an adopter one is, the more one wants the mindset and background.) Excellent examples, both good and bad, abound in the Git tutorials and discussions of Git packaging layouts that were common on Debian Planet a while back.

Mailing list discussion is only useful for a brief period, many new tools suffer from not putting the results of discussions into the documentation, causing other potential adopters to reinvent the wheel or abandon the tool without further consideration. If the resulting thread gets too long or ugly or off-topic, you can't go back and read it and get a relatively quick view.

Documentation also involves the error and status messages within the tool as well as support for options like multi-depth verbosity so that cryptic debugging output is hidden behind -v -v -v -v or --debug.

A difficult challenge for new tools, where the rate of change is high, is translated documentation. Sometimes this can come down to having bi-lingual developers in the team. Where a tool requires a large divergence from current workflow or where the underlying ideas are complex, translation support can be important to getting a new tool to replace an established tool (which may already be translated into many languages and have existing multi-lingual support channels). If there is no established precedent, it can be less of an issue. Translation issues are also fertile ground for misunderstandings and disagreements around complex ideas, especially if all respondents in the resulting flame war have first languages other than English. High quality translations of important documentation would be valuable in defusing these situations.

A reason why popularity may be more important than good documentation, is that the temptation to work by "templates" is high. At the beginning it is way easier to copy and hack based on someone else's work, than to start from scratch after having RTFM.

A short "make it work in 5 minutes" example is a dream. The example doesn't really have to go into all details of the possible use of the tool, but make it clearer that it will be easy to adopt and allow for progressive learning.

}}}
** Complexity, conceptual change, learning curve {{{

What I've seen is that usually something revolutionary appears, the group is not yet ready, so only few people get into it, but in general

it gets completely ignored. Also, most of the time, those new solutions try to solve problems that people have not yet formulated in their heads, or which seem irrelevant compared to their current practices and problems, or where the advantages do not outweigh the perceived disadvantages. Groundbreaking ideas are thus at first rejected until the majority has understood and/or embraced them.

Then it starts pouring into the groups slowly (it might even take years!) through the "sedimenting" process. At some point most of the people in the group will have heard of that new thing, which does not seem like something completely alien. Then people might feel comfortable playing with it, might start liking it, and even use it for real work.

A great example in programming of this general problem is functional languages. People who use languages like Haskall, Erlang, or Objective CAML often swear by them and love their expressiveness, but there's a huge learning barrier because the way of thinking about the problem is so different than the normal imperative language structure.

The idea of having separate line of development for each change *seems* like an overkill. Perhaps the reason for this is that most of us are not that fluent with Git and still think that branches are a big deal. Switching from quilt to topgit requires a good understanding of Git. It's also a major change of worfklow and thinking, because it's just different. I don't want to insinuate that the average DD is stupid, but topgit just doesn't come natural to everyone.

Techniques that require a major mindset shift are most likely to be successful if they're explained by different people from different directions until there's an explanation that happens to match each different way of thinking about the problem space. Techniques that have that sort of learning barrier aren't adopted quickly and often aren't adopted at all until someone manages to rework them so that they can be adopted more incrementally, until someone finds a great way of explaining them, or until the general idea has been around for so long that people have slowly picked up the principles via an osmosis process and are more prepared for the conceptual leap. I think that last is what's happened with DVCSes, which have exactly that sort of learning barrier requiring a conceptual leap and which haven't really been made simpler so much as they've just been around long enough that people have slowly pieced together the basics.

A tool must be easy enough to allow someone to quickly access its most basic features. That's why Git has not had a lot of success at first: it's UI back then has been poor and complicated, needing to grok many concepts. This is beginning to change, because the UI is better, and that the tool is easier to explain.

Tools and techniques with a higher entry barrier need to bring more benefits than those that easily integrate into existing workflows. For example, dbconfig-common is a wonderful (albeit still a bit unpolished and very specific-purpose) tool for handling database configuration, but the amount of work required to understand and adopt it is quite

```
big.

Ideally, the best tools are those that have various degrees of use,
starting from very simple ones end eventually ending up in fairly
complicated use schemes.

Learning curves get really interesting when you have to make decisions
for a group. Given the level of knowledge among the group members, the
final choice is not usually the (technically) best solution, but one
that is accessible to contributors. Alternatively, or in addition, the
learning curve can be flattened through the creation of additional
documentation (e.g. tutorial specific for the group of people).

Similarity of interface to tools the user is already familiar with can
play an important part. Subversion underwent a period of rapid and
widespread adoption several years ago, which was aided by the
similarity of commandline interface with CVS, that almost everyone was
using up to that point. Benefits from "positive knowledge transfer"
could be and were leveraged in this transition. The learning curve for
a user unfamiliar with VCSes at all would be very different because
there's a lot more conceptual knowledge that needs to be acquired
first, not just familiarization with the interface to those concepts. I
don't think developers previously unfamiliar with the concepts are
major drivers of adoption of particular implementations.

I think this also leads to faster adoption of tools that are written in
a fairly transparent and readable language, such as well-structured
Python, Perl, or shell, over C binaries where one has to go digging for
source often obscured by multiple libraries.

------------------------------------------------------------------------
}}}
** Compatibility, gradual adoption, inertia, laziness {{{

A very important factor of success for a tool in Debian is to respect
the workflow of the developers. Most people don't like complete
overhauls of their working methods or tool behavior. Large, monolithic
changes to processes tend to take a lot of time, require a lot of
debugging, and be disruptive to a general goal of getting things done
rather than working on tools. It's therefore much easier to adopt a
tool or technique that can be applied in small chunks or in a
self-contained area, or slowly over time. If adopting a new tool would
mean that they were then unable to use one of their existing tools then
they may not adopt the new tool, at least until they are compatible,
depending the the perceived relative values.

People want to improve their _current_ way of doing stuff to be more
efficient, quicker, more precise, ... They don't really want to think
completely differently. Packaging with Git is not a whole brand new
thing. The tool is revolutionary in itself, but what we do with it is
not. We're just doing more precise work, faster, more elegantly. It is
all about continuation.

Being conscious about this and planning beforehand, the deployment time
```

can be greatly reduced. The trick would be to convert a revolution into an evolution, e.g. with a plan like:

1. create something revolutionary (do not push yet);
2. try to split it in smaller unrelated parts;
3. prepare the people with the simple concepts;
4. deploy the simple concepts (tools/techniques);
5. after some time present the unified now non-alien solution.

The difference is that in a revolution you present a massive change in behaviour and expect the receiver to process it when they might not know they need it or are interested in it. With evolution, the processing is done on the sending side, and the receiving end gets the already digested pieces.

I'm not sure if it's actually possible to instantly replace a current system in a single massive step without any previous exposure to the new idea/tool just, this is human nature and changing how people operate at large scale is a bit tricky. Witness, for example, how no packaging helper tool has ever become mandated, and policy maintenance has a very strong rule of not rendering packages instantly buggy. Nobody wants to undertake the task of persuading everyone in Debian to do the same thing, much less to do it at the same time.

dpkg-gensymbols has been successful partly because it gave us something lots of people wanted (weaker shared library dependencies when stronger ones are unnecessary) but also because it runs in parallel with shlibs files, and maintainers can just carry on using only shlibs files if they prefer. Compatible binaries, such as the scrollkeeper -> rarian-compat binaries, or the GnuTLS OpenSSL API are essential. Having a standard format for your data, e.g. patches, makes it simpler to switch patch systems.

Revision control migrations are about the only tool change I've seen that have ever really got away with such immediate changeovers. Even then, most people trying to do those for non-singular teams nowadays run multiple systems in parallel for a while to let people migrate their workflows gradually. Packages that already have VCSes in place don't usually get switched to a new system until there's an easy way to preserve the history, even if the maintainer already has strong reasons to prefer a different system over the one currently used.

Having conversion instructions, such as the "New Python Policy" Debian Wiki page, make it a whole lot easier to do the move, and avoid forgetting critical steps.

There's also an inertia or change resistance factor here: techniques that require intrusive packaging changes need much more testing, particularly for the most important packages, so they can't be adopted lightly. Don't change for the sake of it. You have settled on a workflow that "does the job" and even if it has some glitches, it is generally OK and the corner cases are rare.

It is just so much "easier" to let your hands repeat the same, tedious

routines over and over. And some tools are better at allowing that: Continue using same editor, same VCS or whatever - even if perhaps it would in principle be better to restructure the packaging workflow. People tend to stick to old habits and time shortage just leaves little room to adopt new techniques.

Many tools could be perceived as outdated by other developers, even a majority of other developers, but migrating the remaining developers away from those tools can only be finally achieved by making the tool redundant, providing a drop-in replacement and making the removal of the old tool part of the release goals. Developers going MIA can actually assist in this process because the affected packages become orphaned and other developers can remove the package or migrate it to the replacement tool via a QA NMU.

}}}
** Network effects, team adoption, cooperation {{{

Usually, a packaging team will have a team standard for tools and techniques, and however that standard is born or changes with time, once it's set, anyone new joining that team often has to follow that standard, unless the purpose of joining is to change the direction of the team itself.

Teams tend to be fairly conservative in their adoption of new tools and techniques to avoid creating too high of a barrier of entry, so they want to use general project standards where possible, and people who join those teams and become familiar with those tools and techniques tend to reuse them elsewhere outside the team because they're already familiar with them.

For all the developers who are not working alone, it is crucial that any change to one's current practices can be done harmoniously with the one of others. In some ways, svn-buildpackage is just the developer's choice which "wrapper around dpkg-buildpackage" to use. But which build dependency you are using is coded in the source of your package, and changing or hijacking that forces people to accept it.

Having a set of competing programs for doing the same thing causes a split in their users. Teams should try to settle down to use one of them to increase effectiveness of cooperation. Team members will approach a new tool with preconceptions and the team dynamic will determine the final choice, whether that be an autocratic or a collective decision. The history of such decisions will change the way that later decisions are handled such that prior experience of the last team evaluation can prevent or delay any discussion around a possible new tool.

A tool could be demanded by someone you would really want to join your team.

I have witnessed resistance to new tools or techniques in a derivative that diverged from the Debian toolset and making integration/

```
coordination with Debian harder (even though there were clear benefits
within the derivative itself for doing so), because the tools were not
used elsewhere and some volunteer developers rather spend time doing
productive work than learning new tools.

------------------------------------------------------------------------
}}}
** Recommendations, advocacy, peers, lead users, buzz {{{

One of the major influences I've seen is what I would call
"peercolation". I generally noticed that Debian contributors often
learn about new packaging techniques through informal discussion with
their peers, namely other Debian contributors. Having some buzz and
excitement around a new tool or technique seems to help. If several
people are blogging about using something, lots of other people will
become aware of it, and start thinking about using it.

Developers are as susceptible to hype as anyone else and just as likely
to change their preferences should the experience not match the hype as
to jump ship entirely if another task or tool generates a new round of
hype and rumour. It is possible that the initial hype around a tool is
(temporarily?) more powerful as an influence than some of the other
factors.

It may not be particularly technically noble, but it's hard to deny
that opinions of prominent developers make a difference to people's
default opinions on new techniques. This is even understandable without
having to decry anyone as merely a herd-follower: busy developers don't
necessarily have time to keep up with all the latest fads, and it's
perfectly sensible to assign some level of trust to other people's
judgements. If all the "cool kids" use a particular tool then they will
be more likely to use it. I think this effect will be more pronounced
in less experienced contributors, for whom it is probably a wise
strategy. The thought leaders probably have a substantial impact on the
decisions made in Debian.

There are multiple groups of what I would call core developers. Such
developers are those who make Debian, because they are part of
packaging teams of core packages (like Essential ones), part of vital
bits of Debian (d-i) or members of Core teams, or maintaining the
infrastructure, name-your-very-important-task. Those kind of people are
usually quite independent in their way of thinking, and won't use a
tool because it's trendy. Though this groups is not immune to trend
effects once shown that a tool is superior technically to the current
practices. The core developers are known for either strong character/
opinions, sound (or a certain kind of soundness) technical choices, ...
and many people trust them. They're in vital parts of Debian, and
wouldn't be there if the rest of Debian was really believing they did a
good job. In other words, when there is disagreement in a team, their
voice counts often a bit more.

I wouldn't fancy my chances of propagating any sort of potentially
controversial technique unless I had already canvased opinions of some
other well-known developers and made sure that they thought I was doing
```

the right thing. While this does mean that half-formed opinions or even just disinterest from key developers can hurt adoption of a new tool or technique, it does have the positive effect that proponents often need to put work into building consensus, so I don't expect this effect to go away any time soon.

The most persuasive advocates are the ones who can clearly explain why a new tool or technique is better, who are perceived as having significant experience within the area affected by that tool or technique, and who seem to have done all of the research and experimentation that we'd all like to do but usually don't have time for.

The perceived merit of the developer is affected by the amount of overlap of previous ventures with the priorities and preferences of the potential adopter.

Success stories and positive attitude are stimulating. Long threads with many branching points full of point-to-point responses are difficult to follow (and often become unfocused and have a low signal/noise ratio). For this, Planet Debian is definitely superior to debian-devel@l.d.o, where anybody can ruin a discussion by joining it to nitpick on something he misunderstood.

A tool's "coolness" or the "it seems this works great for @someone" factor can make you consider a new way of work (or new window manager/email client/...) time and again. If your expectations were exceeded, you could warmly^W cooly recommend it to others.

Recommendations are often regarded with a fair degree of suspicion and negativity, as if the person recommending the tool has some ulterior motive to propagating the use of the tool

A tool designed, written, promoted and packaged by the very same person has lower chances to survive, of course... unless that person is someone very reliable.

The informational influences are not limited to the discussion of new tools. For instance, after the "OpenSSL accident", a long discussion took place on debian-devel about divergence from upstream, and although no formal rule emerged from it, several people started to take better care to document and forward the patches.

A tool is not only who uses it, but also who makes it, who advocates it, ... all the ecosystem, the aura. Git has a tremendous adoption curve, because it's the kernel's tool, it's Linus' tool. OTOH bzr is pushed by Canonical (which is a bad start in Debian, it seems), is seen as slow, and even though it has had a lot of early adopters, especially since people packaging for ubuntu use it, I don't think it has had a big grow recently.

CDBS has been quite well marketed; it was introduced by developers which at the time were prominent, active and close to the core, and continue being well regarded and respected, so publicity and good

```
personal references probably helped in it being used for a quarter of
the packagers in our archive.

People who aspire to become core developers are more prone to "succomb"
to trends, because it's there that they are more likely find something
innovative enough to make them recognized among their peers.
Occasionnaly you find pearls in old concepts, but it's usually safer to
listen to new trends to find nice stuff. You become a core developer
through innovation.

-------------------------------------------------------------------------
}}}
** Popularity, standards, critical mass, support, derivative tools {{{

A significant proportion of the Debian community -- I'll even go out on
a limb and say most -- just want to use whatever is standard. I think a
lot of people have the (fairly reasonable, overall) perception that
there are many good ways of doing things, and that anything that's
broadly used is probably good enough to work once they learn it. They
therefore base their decision-making on what seems to have the most
momentum and mass-adoption, since that's the method that they're most
likely able to get support for.

Creating a sense of momentum can cause people to perceive something as
a standard.

Using what other people are using increases the likelihood to profit
from the "aftermarket". For example, if you use Git, you can use
pristine-tar; there's no inherent reason why pristine-tar couldn't work
with other revision control systems, but they're not currently popular
enough for someone to have ported it. Users bring feature requests and
patches to make the system work well in various contexts.

Popularity often brings better documentation and tools to lower the
entry barrier (think VCS-buildpackage, /usr/share/quilt/quilt.make,
etc.). In addition to documentation, there is the ready availability of
support. If a system is not optimal, but I can go on IRC, ask a
question and get a meaningful answer in a matter of seconds, then most
of its shortcomings can be worked around with little effort. I think
that CVS has been in this situation for a while. Even if simply
renaming a file was a pain in CVS, you could get a reminder of how to
do it within 5 seconds, either with a Google search or an question on
IRC. The same goes for e.g. UDD: even if I didn't know Postgres, or
SQL, I could go on IRC and ask and probably people will throw SQL
queries at me in no time. UDD is a new tool, but it leverages existing,
well-known, and popular technologies so well that adopting it is a no
brainer.

It's also easier to deal with others' work if they use a similar/
standardized approach to things.

Some will not want to use tools that are not used by others. This
effect will be magnified if the tool has an impact on what other people
see. If the tool a person used is impossible or very hard to detect if
```

you can't see how they work then they will feel more free to choose than if their choice of tool has a large effect on other people.

Once a new technigue has gained a critical mass of users in the community, it attracts even more users because many developers are curious followers and they want to verify everything they heard. It's not necessarily the best item that gets more popular, but the item that gets more popular first gets more attention and rises further. I believe that a good big deal of the success of git over mercurial has been the publicity derived by the fact that Linus was using it for the kernel, which brought loads of heavy adopters, and quickly made it so that it was easier to get help on git than on hg. Even if you basically didn't need any help to use hg but you needed help to use git, hg made you feel lonely.

All of this would be vaporware if there would not be the technical possibility to use the tools for real. For instance, would there not be svn-buildpackage, many would not be using Subversion. Would git-buildpackage emulate mergeWithUpstream, many would perhaps already have migrated to Git.

Often we'll become marginally aware of something, and decide we don't have time to deal with it, and come back to it later. Then we can look at what bugs it's accumulated over time, and how well they were dealt with, whether people are still using it, etc, and avoid having to be an early adopter.

}}}
** Top-down diffusion, infrastructure support, formal docs {{{

Setting a de-facto standard for packaging is the best thing to let people adopt the technique. A simple pagramatic fact is that core developers edit the developer reference or the policy. Convince them, and you have excellent leverage to make everyone aware of your tool.

While diversity is good to let many technique compete, there's a time when that diversity hurts more than anything else and the project will try to standardize on the best option. Standardization comes from the policy (or rather from lintian which dictates what maintainers do) or from core tools maintainers. But standardization is not always an explicit process, it's also often the result of a long-term network effect. Think how debhelper gained momentum.

A good example is the now almost universal convention for indicating multiple authorship in changelogs like this: "[Author's Name] ". This format grew from experience in the d-i team. I think it goes further than that, though: once everyone in the d-i team started using it, some of those people also spread the convention around to other teams they were involved in.

The multiple authorship convention is also a good example of how incorporation into key packages makes a difference. If something shows up in dpkg-dev, debhelper, or devscripts, then you can pretty much

guarantee that a very substantial chunk of developers are going to start using (and improving) it soon.

The reverse is not necessarily true: there are plenty of reasonably successful tools that are not in such key packages (quilt comes to mind). That said, there are cases where it's clear that the adoption curve goes nowhere fast unless the technique in question shows up somewhere nice and central: the next-generation dpkg source format is clearly unlikely to be adopted until dpkg-dev in unstable supports it, not to mention the archive maintenance scripts.

A more active kind of influence is when the change is pushed from the top. Would I have the choice, I would be very interested with the "Wig and Pen" format for binary packages. But the dpkg maintainers actively push the version 3.0 (quilt) that is similar and more powerful, but more complex. Unless they push so hard that they create some resistance, I will probably be using this format for the next release, not because I need all the features, but because it has been decided somewhere else that it is the new default.

By providing infrastructure, the project/core people can drive adoption: .debian.org, alioth and team maintenance infrastructure, BTS usertags. The project can kill off other efforts through inaction (git v3 source format), actively ban some (dynamically created debian/ control), and dictate others (through the policy for instance; although that's not tied to specific tools, po-debconf comes close).

A variant case of social influence is the debian-mentors mailing list. "Do this, do that, or I will not sponsor your package" (with some explanations) shapes the choices of our next generation of developers. I've seen misconceptions of patch management systems being required on debian-mentors.

Some decisions are literally forced on a developer in a particular area without any choice, and that's not necessarily a bad thing, because sometimes change needs to be driven. Generally, those seeking a big change within Debian know that this issue must be tackled early on and informal discussions take place to get the most obvious problems out of the way before hand.

Some ideas have sunk without trace after initial discussion on debian-devel - typically where no informal discussions were done beforehand and the original proposer lost motivation to pursue the idea after various developers with (or without) cross-distro roles criticised it. There was an idea to replace the flat-file apt and dpkg cache data with a relational database. If the original idea had set out from the start to say that libsqlite3 was the target and that support could remain optional, it may have found a lot more support.

Not quite the same issue: reportbug-ng and /usr/share/bug/ scripts is one situation of this general type. Suddenly, many developers using reportbug scripts started getting junk bug reports without important information, wasting a lot of their time and generating a lot of bad reviews of reportbug-ng. The situation escalated from there when the

```
developer of reportbug-ng refused to fix the problem. The new tool
relied on a cross-distro tool (the BTS) which all developers had to
use, so when the new tool neglected to ask or obtain data that the
people trying to fix the reported bugs had carefully enabled for their
packages, those developers felt that an unwelcome change had been
imposed on them, without their consent.

Standards documents obviously have a substantial influence,
particularly over the behaviour of new developers (long-established
developers might try to change policy if they think it's introduced
something wrong ...). Once a technique gets written down somewhere that
looks "official", a large number of developers will use it by way of a
default behaviour. I remember that Raphael Hertzog was particularly
keen to get packages.qa.debian.org described in the Debian Developer's
Reference, for example, and I seem to remember that its use increased
significantly once that was done.

Packages that enforce standards in some way exhibit a similar effect.
Lintian enforces quite a number of things that aren't mentioned in
policy but that are reasonably widely agreed by developers, and that
can have the effect of moving people over to newer techniques. For
example, I think it's clear that Lintian's not-using-po-debconf check
has helped to move a number of people over to using po-debconf rather
than the old debconf-mergetemplate system.

The Debian Policy is completely downstream of the whole process: things
get documented there when they are supposed to have become a "lieu
commun" for everybody. Basically, learning to package by studying the
Policy makes one overlook any recent best practice. Fortunately, many
sub-groups maintain their own group policies which add the best
practices that are not yet written in the Debian Policy (for instance,
to forward patches and document it has been done in their headers, or
to use a machine-readable copyright format).

--------------------------------------------------------------------------
}}}

Thank you for your time,
[...]

% vim:foldmethod=marker:foldlevel=0
```

Listing C.7: *The message sent to the panelists at the start of the third and final Delphi discussion round.*

```
Subject: [Delphi] Third round: identification of salient influences

[GREETING] [FIRST NAME],

Finally we can kick off the third round of the Delphi study on the
influences on adoption behaviour among Debian package maintainers. You
have provided me with incredibly insightful data and I worked hard to
pay due respect to your efforts. Despite the three month delay, the
```

study is still on, and I am more excited than ever about this research. Thank you for your dedication and patience.

For this third round, I need a reply from you by **Monday, 20 April 2009**. I have worked hard to make sure that you can complete this round in reasonable time. If you know you will not be able to meet that deadline, please let me know as soon as possible so that we can make alternative arrangements.

What follows is a recount of what we are doing and what happened so far. If you prefer to skip this introduction and get straight to the task, feel free to jump forward to the instructions for this round, marked with "INSTRUCTIONS".

First, let us recapitulate a bit what happened so far, simply because a lot of time has passed since the last round. The ongoing study is a Delphi study, in which a number of panelists respond to questions. Those responses are then anonymised and used as the basis for the question (or task) of the next round. You are one of the panelists for this study, because your experience and knowledge with respect to the Debian project is significant for the diversity of the panel.

In the first round (early November), I asked you and the other panelists to identify and describe six influences to adoption behaviour you have seen and expect to see again in the domain of Debian package maintenance. Every one of you responded with very valuable input, and it took me a bit of time to condense and collate the responses.

For the second round (late November and early December), I presented you with a very long email containing bits of those responses; the richness of the data you provided was so high that I was unable to shorten it more. This time, the task was to respond to each other's statements with counter-arguments and qualifications. Again, all of you responded, even though I asked you to spend way more than the hour-per-round I had initially quoted. In addition, you all helped further by engaging in subsequent discussion when I asked you for clarification of some of your statements, or when I proxied arguments between panelists.

I greatly appreciate everything you have done up to this point.

We are now at the start of the third round, where we build on the data collected so far. Once this is done, I can finally distribute the reimbursement gadgets I announced previously.

Please remember that part of this study's benefit will be the availability of all of the data under a Free licence, while respecting your anonymity requirements.

The application of the Delphi approach in a FLOSS environment is also part of this research. I owe it mostly to your dedication that this first-ever application of the Delphi method in the FLOSS context will end successfully. The volume of data well exceeded the limits of the Delphi method, which goes back to mistakes I have made during

```
preparation. I have learned much about the process and will publish all
details of the method, as well as the problems I have encountered,
afterwards. Since the Delphi method fits so well into the FLOSS
context, I hope to make it easier for future researchers to avoid these
pitfalls.

Thank you for your help in making this happen.

------------------------------------------------------------------------

INSTRUCTIONS

The task of the third and final round of this Delphi study is to
identify the most salient influences to the decision for or against
package maintenance tools/techniques among Debian contributors.

Further down in this email you will find 24 paragraphs, which are
derived from selected statements you and the other participants made in
previous rounds and follow-up discussions. The statements I used to
derive each paragraph are listed directly below it. I do not expect you
to read the statements in addition to the paragraphs, but I have
provided them in case you require more information. Fold markers are in
place to allow you to hide them.

The following is an example:

 1. Some keywords

    The paragraphs on which you will concentrate are just like this
    one. It can consist of multiple sentences, but there is a common
    theme in those: the paragraph will represent a single influence you
    and your fellow panelists have identified, and your task is to
    determine how important this influence is.

    {{{ supporting statements
      - These are statements following the paragraph.
      - Each statement comes from the data we created in previous
        rounds.
      - The gist of each of these statements flowed into the above
        paragraph, which summarises the collection of statements you
        see here.
      - You need not read these statements, but I have included them in
        case you are interested and want further information about a
        paragraph.
    }}}

 2. Some other keywords

    This is another paragraph...

    {{{ supporting statements
      - ... with supporting statements. Just one, for brevity of this
        example.
    }}}
```

In case you are interested in the numbers: I extracted 3600 lines of text from your replies, wrote almost 250 emails, and processed just as many replies. The total volume of text was about 200 pages at 62 lines per page. I identified 204 relevant statements in those data and used them as the basis for the current task.

THE TASK

Your task for round three is comprised of the following steps. The rough time estimate for each step is given in brackets. I am aware that these are absolute minima, and that you will realistically spend slightly longer on the steps. I was unable to further reduce the task or data without negative impact on the study and its results.

As before, you need not write full sentences or pay particular attention to language in your reply, as long as I can understand what you mean; this should save some time. However, pointers to background information, such as mailing list threads or other documents, are greatly appreciated.

1. Please read this email... [~5 minutes]

2. ... and all paragraphs that follow below and make sure you understand the central idea of each; consider taking notes along the way. Feel free to ask me in case of any questions or uncertainties. [~25 minutes]

3. I suggest to let it settle and think about the paragraphs a bit, e.g. while you do the washing, commute, or take a walk. [0 minutes :)]

4. Go back to the paragraphs and skim the list again to refresh your memory.

    a. Then, please select the three paragraphs which reflect the three strongest influences (positive or negative) you have experienced in the context of **your packaging work** in the Debian project, and share details about how these influences manifested themselves, and how you expect them to manifest themselves in the future in your immediate environment. [~15 minutes]

    b. Next, please shift your perspective away from your own work and consider the Debian project as a whole: identify the three paragraphs you deem most relevant to packaging tool/technique adoption behaviour in the **entire project**. Again, please provide details about instances of those influences you have witnessed in our project, and/or speculate where and why you expect to find these influences in Debian in times to come. [~15 minutes]

```
    The two sets of three paragraphs each may well overlap, but they
    could also be different. I am asking you to consider the influences
    from both angles -- your own, as well as the Debian project's
    perspective -- because I need you to be conscious about the
    difference; my main goal is the identification of influences at the
    project scale, but if no consensus can be found, then I will have
    to break the analysis down to the various cliques to which you
    belong. This is one of many aspects in which the diversity among
    the Delphi panelists is so valuable.

    If you have considerable difficulties choosing between two
    paragraphs because they overlap or are too close to each other,
    then combine them into one of your salient choices, and reply to
    both at once.

It goes without saying that any additional comments are welcome; if you
have a problem with a choice of word, if you feel like I have
misrepresented an influence while summarising the statements gathered
from your input, or if you are missing anything, please let me know.

------------------------------------------------------------------------

THE DATA

The paragraphs of data are in random order, and neither paragraph
length nor number of supporting statements are any indication of
popularity or importance, but merely a result of the processing.

You can fold away the supporting statements in your editor. In Vim, use
'set fdm=marker'. For emacs, you need to install the emacs-goodies-el
package, and then run the following commands once you opened the
message for replying (assuming '> ' is your reply prefix):

M-x mail-mode
M-: (load "folding" 'nomessage 'noerror)
M-: (folding-add-to-marks-list 'mail-mode ">     {{{ " ">     }}}" nil t)
M-x folding-mode

Here are the paragraphs. You can find a list of just the keywords with
one-line summaries at the end of this message.

 1. Return-on-investment

    Adopting a tool/technique incurs costs. We are conscious about
    these costs and seek to profit from the adoption decision. Active
    maintenance, other users, better documentation and support options,
    an aftermarket, evolution of features, and adaptation to changes in
    the environment are some of the returns on investment we require.
    For the final adoption decision, we also seek the comfort of the
    "multiple eyes and shallow bugs" principle.

    {{{ supporting statements
      - migration costs include learning of new processes and concepts,
```

```
      not just the efforts of the migration itself
    - developers cannot maintain all the tools they need, so they
      make a judgement as to their maintenance status
    - reaction time of the maintainer, and turnaround time for bugs
      and features are indications of good maintenance, but no
      guarantee
    - a responsive maintainer gives the impression to work for you,
      which can be quite motivational
    - tools/processes need to evolve and adapt; without active
      maintenance and input from users, tools end up being more work
    - reach out to people to avoid duplicate efforts and recruit new
      people, which is essential for a project's survival
    - a winning technology ends up with an aftermarket
    - popularity brings better documentation
    - reaching out to people lets the rest of the free software world
      know that a project is alive and kicking, which is often a
      prerequisite for adoption
    - adopting a popular tool means less of a risk of ending up with
      an unmaintained tool
    - developers generally try new tools for their own scenarios and
      rely on others to discover problems before making a final
      adoption decision
  }}}

2. "Peercolation"

  Information spreads through people networks, and team dynamics
  carry ideas beyond team boundaries. Those with significant
  experience in an area, and who can clearly explain a tool's
  benefits, get more respect. People tend to favour peers they trust,
  or with whom they have overlaps in interest or heritage; similarly,
  large teams and those in charge of important packages are more
  influential. The cost of investigating something new cannot be
  avoided, but those who present the facts which we could not
  research ourselves are the most persuasive as they appear to help
  and save us time. Popularity and critical mass seem more important
  than individual people using a tool/technique, however. With
  increasing experience, people become less reliant on the opinions
  of others. Somewhat unexpectedly, we apply less scrutiny to
  projects stemming from outside Debian.

  {{{ supporting statements
    - obtaining feedback increases the number of people who might
      later recommend a tool
    - overlap in interest and heritage causes favouring view
    - a tool is more interesting if it is praised or advocated by
      respected people
    - other people's advice can offset the cost-benefit analysis, but
      cannot replace the cost of investigating the new concept to
      make a decision
    - the most persuasive advocates can clearly explain why a new
      tool is better, are perceived as having significant experience
      in the area, and who appear to have done all the research we'd
      like to do if we had the time
```

```
    - people who always jump at the new and shiny or surf the hype
      are less influential than those who judge/advocate based on
      understanding, or they might even put others off
    - people who always jump at the new and shiny act as first-pass
      filter for tools
    - exposure in a team carries ideas beyond the team
    - less experienced people are more likely to copy what the "cool
      kids" are using
    - more experienced people are usually independent in their
      thinking and don't pick up every trend
    - the opinions of trusted people are influential
    - technical merits, popularity, and critical mass are more
      important than individual people using a tool
    - big teams can have quite an influence, but the importance of
      packages seems more significant than the number thereof
    - sponsoring/mentoring drives tool discovery and adoption, but
      can also be abused and could squander the capacity to innovate
      and benefit from few preconceptions of the less experienced
    - it is the very aura of the kernel that puts people off git
    - we scrutinise more tools developed inside Debian
    - tools from outside of Debian are more mature when they reach
      the project and thus scrutinised less
    - we're less familiar with the concepts underlying outside
      projects, so we criticise less
    - outside projects have a larger inertia and are thus harder to
      influence
    - people new to a domain might reinvent wheels badly
    - a project could come out of the blue if it has a credible
      pedigree attached to it
  }}}

3. Maturity

   A tool/technique must be usable to sustain followers. It should not
   change continuously and require users to re-learn or change their
   scripts. It should also be free from serious bugs that would make
   its use too much of a pain, especially if alternative tools/
   techniques do not exhibit such bugs.

   {{{ supporting statements
     - implementation maturity -- interface stability and freeness of
       serious bugs -- are important; stable software is less likely
       to require repeated/continuous adaptation
     - hype around a tool is important, but to gain majority, a tool
       must be mature
     - tools should get a simple job done early, then grow
     - an unfinished tool can be adopted if it looks promising
     - does the tool meet its claimed goal/purpose
     - the stability of the Debian policy adds to its credibility as a
       reference
     - problems of a tool that competing tools do not exhibit are
       significant barriers to adoption
     - expectations will be lower for tools solving new problems
     - tools that are good ideas and well designed will get rewritten
```

```
    in what is perceived to be a better language
  - tools that are implemented badly to get a job done might get
    rewritten to address the problems
}}}
```

4. Trialability

```
We evaluate new tools/techniques in the context of our own
use-cases to determine their worth. A tutorial, especially one
which identifies problems, and the ability to try out the tool with
minimal time-investment can help with that process.

{{{ supporting statements
  - a tutorial can help with cost-benefit analysis to a certain
    degree, and more so if it identifies problems and explains how
    to work around them
  - can i do something with it in 10 minutes?
  - a shallow, "make it work in 5 minutes" example facilitates
    adoption, especially if it makes it clear that the tool allows
    for progressive learning
  - people prefer trying out to research, first impression is more
    important than heritage
}}}
```

5. Genericity

```
Being able to streamline work by reusing the same (or similar) tool/
technique in different scenarios is an important feature. This
sort of flexibility of a tool/technique can be significant enough
to avoid an existing solution for a single task if it cannot be
used elsewhere. On the other hand, too much flexibility can result
in reduced usability. The usual Unix-principles -- KISS, modularity,
adaptability, and enforcing just the right amount of policy -- are key to
reusability.

{{{ supporting statements
  - active developers with many packages may incur higher
    transition costs across all packages
  - tools that solve a wider set of problems, or that work more
    generally, are adopted wider as people prefer to apply similar
    techniques to everything they work on, and there will be more
    support
  - people are likely to adopt tools that automate manual labour,
    if it does not take away flexibility
  - reduced maintenance cost is an incentive, but flexibility/
    scalability remains an important consideration
  - the KISS-principle is key to reusing techniques
  - important themes fitting well with the Debian system are
    modularity, adaptability, and enforcing just the right amount
    of policy
  - flexibility is arguably more important than precision
  - once a tool becomes too flexible (needs a shell script to run
    it with all the options), its benefit decreases
}}}
```

```
6. Quality assurance

   Automated quality assurance (e.g. lintian) can drive adoption, but
   only if there exists consensus about the rules; they must not be
   abused to recommend or enforce policies for which there is no
   consensus.

   {{{ supporting statements
     - test suites (incl. lintian) can drive adoption by turning
       consensus into recommendations
     - test suites (incl. lintian) must not be abused to enforce/
       recommend rules without consensus
     - machine-parseable copyright files make sense, but maintainers
       don't care enough as there is no benefit for them; dh_make and
       lintian will need to address that
   }}}

7. Transparency

   We are reluctant to pass up control. With functionality spread
   across abstraction layers, a tool is harder to understand and
   handle in the face of problems. Transparent design, a readable
   programming language, and resilience to bugs in underlying
   components all help to prevent loss of control. Tools automating a
   process have much higher transparency requirements than tools
   automating a single task, as there is more variance in the use
   cases.

   {{{ supporting statements
     - transparency of a tool and being able to tell/find out what it
       does decreases dependency, enables picking up the pieces if
       something breaks, and facilitates adoption
     - complete documentation includes details of how to reproduce
       manually what the tool does, and fix any problems it may cause
     - abstracting, or moving functionality somewhere else usually
       makes things harder to understand and/or comes with loss of
       control
     - an important consideration in adopting a tool is whether one
       can grasp what is going on and how easy it is to influence
       those steps
     - helper tools written in a familiar, or a transparent/readable
       languages are easier to adopt; tracing readable scripts is
       easier than debugging a separate code base, or code written in
       an unfamiliar language
     - concerns over opacity are alleviated by resilience to bugs in
       underlying components
     - workflow helpers (*-buildpackage) have a much greater
       transparency requirement than worker tools (lintian, gpg)
     - helper tools to automate existing problems are easy to write
       but harder to adopt since everyone was solving the problem
       somewhat differently
     - too simple tools make it possible to upload packages without
       fully understanding packaging, which may harm the project; we
```

```
      are engineers, not assembly-line workers, and there is a
      certain minimum of knowledge required
    - more important than the implementation is the public interface,
      including --help output and manpages
    - automation must be motivated by the desire to do more, not to
      do less
  }}}

8. Uniformity

  Consistency across the project as a whole can motivate change. A
  good reason for adoption of a tool/technique can be the desire to
  realign outlying factions, i.e. teams who are doing things
  differently. Driving that kind of change is tricky -- events, such
  as negative incidents, can be good vehicles for change. Support by
  the project infrastructure is vital, if applicable.

  {{{ supporting statements
    - maximum flexibility is good in the individual case, but in the
      distro-context, uniformity is also important
    - lack of uniformity make our life harder
    - packages using standard means and simple tools are easiest to
      patch, and facilitate NMUs and adoptions
    - we should strive towards greater standardisation and treat
      Debian more like a consistent, public interface
    - Debian as a whole should be considered as a public API: a
      defined infrastructure and some rules about how to interact
      with it
    - group-specific documentation/tutorials can flatten the learning
      curve, but can also lead to similar-but-incompatible workflows
    - incompatible solutions balkanise the project
    - Debian is really bad at deprecating techniques and approaches
    - it is a shame that we do not make one solution mandatory, even
      if a problem-space is well-explored
    - Debian may be about choice, but people dislike too much
      diversity in packaging, there are some strict rules, or ethics
    - you can never be sure that a migration is complete, i.e. that
      everyone has migrated
    - changing how people operate at large scale is tricky
    - at some time, diversity hurts more than anything else and the
      project will and has to try to standardise on one option
    - negative incidents are a good incentive for change: the OpenSSL
      debacle has caused many maintainers to reconsider patch
      maintenance
    - Debian is already standardised in some aspects, but the
      advantages of further standardisation would outweigh the
      disadvantages and we would not lose the diversity
    - support for a tool/technique by the project infrastructure is
      vital
  }}}

9. Quality documentation

  Documentation is a necessity, especially when a tool/technique
```

```
diverges far from existing processes. Mailing list archives and
source code are not sufficient for wide-spread adoption. Instead,
documentation needs to be kept up-to-date and trimmed to not become
overly verbose. Early adopters want background information and care
about motivation, while later adopters seek tutorials.

{{{ supporting statements
  - the importance of docs is directly proportional to divergence
    from similar tools or existing processes
  - mailing list discussion are only useful for some time, most
    projects fail to port the results into persistent, organised
    documentation
  - long-term, permanent docs, maintained so that they do not get
    too verbose
  - if a tool is so badly documented that you have to refer to the
    source code, the adoption rate will be low
  - early adopters care more for mindset/background documentation,
    later adopters seem to prefer howtos
  - popularity is more important than documentation
  - there is more fun in programming/doing something than
    documenting it
}}}

10. Scaled use

    Tools or techniques that offer various degrees of use are good; it
    should be painless to employ a tool's basic features, and then
    allow for progressively advanced use. Tools/techniques that
    implement a standard are easier to adopt if they are flexible
    enough to be used in non-standard ways, and support the convergence
    with the standard at a later point.

    {{{ supporting statements
      - a tool should be easy enough to understand to allow someone to
        quickly access its most basic features
      - the best tools are those with various degrees of use: simple to
        get started with, yet powerful for the advanced user
      - tools that adapt to your current process and leave it open to
        evolve it to some standard are easier to adopt
    }}}

11. Consensus

    Pioneering work is necessary, but concrete solutions need to follow
    from that. It is important to build consensus with experienced
    people, and proponents from all involved sides. Too much
    discussion, however, can cause loss of focus and hinder change.

    {{{ supporting statements
      - controversial ideas are best discussed first in small groups
        with proponents from all involved sides, as well as well-known,
        experienced developers, to build consensus
      - change can often be hindered by too much discussion and loss of
        focus
```

```
  - policy changes also need a maintainer who is willing to freeze
    a proposal and not succumb to polishing
  - we need pioneers, but also people who work on concrete
    solutions
}}}
```

12. Cost-benefit

```
Suboptimal tools can be painful to use, or become a nuisance in the
future, but they might just get the job done _now_. A simple tool
might also just be enough for a specific use-case. A high barrier
of entry -- having to spend a lot of time before being able to reap
benefits -- can deter potential adopters. While change for the sake
of change can rarely be justified, Debian contributors appear
keener to experiment than members of other projects. This either
offsets the cost-benefit analysis, or decreases its importance.

{{{ supporting statements
  - we are not always willing to invest a lot of time before being
    able to reap benefits, and changing habits can take a lot of
    time
  - some volunteers prefer to spend their time doing productive
    things, rather than learning new tools
  - sometimes hacks can help save time and boost productivity, then
    it is better to accept a good-enough-but-not-perfect tool and
    get on with other things
  - hacks often turn into long-term nuisances, unless they get
    implemented properly
  - simpler, possibly sub-optimal tools might just work well enough
    in a given scenario
  - there are two types of benefit: personal and project; these are
    weighed differently by individuals
  - Debian contributors are keener to experiment compared to other
    projects, possibly because they work more in their free time
  - change for the sake of change does not make sense
  - some people are more pragmatic than others when it comes to
    perfectionism, a core cultural trait of the project
  - process/tool churn distracts and keeps people from doing real
    work
}}}
```

13. Marketing

```
Buzz and excitement increase the exposure of a tool or technique,
and can drive evaluation, though not necessarily adoption. It is
important to use the right channels, and to provide repeated
exposure to the tool/technique one is promoting, not just a single
glimpse. Success stories and a positive attitude are stimulating,
especially when needs are met instead of created. A corporate link,
premature promotion, and TV-style marketing, on the other hand, can
have negative effects and ought to be avoided.

{{{ supporting statements
  - advertising, marketing: it's about multiple glimpses, not only
```

```
      the first glimpse
    - success stories and a positive attitude are stimulating
    - "coolness" or "it seems to work great for someone" can make you
      consider a new way of work
    - meet needs, instead of creating them
    - a buzz or excitement around a tool seems to help its exposure,
      which can drive evaluation of tools, but not necessarily their
      adoption
    - blog posts are more suitable for advertising than mailing list
      posts
    - people paid to work on Debian-related projects are sometimes
      hesitant to promote their tools due to the anti-corporate
      sentinment in the project
    - premature promotion reflects poorly
    - it may be better to plainly state what a tool does and what
      problems it solves, backed with examples and tutorials, instead
      of TV-style marketing, especially if the project is young
    - it can be hard for newcomers to bring work to wide attention,
      or they don't know how to
    - it is not always the best that gets most popular, but the most
      popular gets more attention, rises further, and improves
    - recommendations in our community are not seen as negative;
      rather, we are influenced by recommendations but tend to make
      our own decisions in an informed way
    - we are allergic to "overhype" and have grown "antibodies" (e.g.
      a cynical attitude) to hype, maybe due to a feeling of guilt of
      not keeping up with developments ourselves
  }}}

14. Compatibility

  Learning a new tool requires investing time. Tools that can be used
  without having to learn them, or which automate what you are
  already doing are readily adopted, especially when they come with
  migration instructions. Tools that build on known concepts or are
  "finger-compatible" are easier to learn. Only drop-in replacements
  can completely superseed a tool though.

  {{{ supporting statements
    - developers don't want to think about using a tool, it needs to
      come naturally, be finger-compatible
    - an excellent tool is one which you can just pick up without
      having to think or learn anything
    - tools that can leverage positive knowledge transfer (e.g. build
      on existing concepts) are easier to adopt
    - the easiest tools to adopt are those that automate what you are
      already doing
    - compatibility with existing practices is part of the merits of
      a tool; adoption occurs when little has to be changed and
      learnt
    - conversion instructions make it a lot easier to migrate and
      help not forgetting critical steps
    - drop-in replacements are about the only way to make a tool
      redundant and completely replace it
```

```
    - it is often easier to let your hands repeat the same, tedious
      steps over and over
    - wikis, albeit more and more popular, are disliked because they
      force people to leave their editors or go through extra steps
      to set them up to interoperate
  }}}

15. Elegance

  Working on something that pleases can help increase one's
  efficiency. The design, quality of implementation, and technical
  correctness of a tool/technique can be important factors to some,
  but there is also the "feeling", a personal preference which cannot
  always be qualified, and can result in irrational behaviour.

  {{{ supporting statements
    - design seems less important than need; we put up with annoyance
      of a tool if it satisfies a need
    - quality of implementation of our tools is important to some of
      us
    - our main work is package maintenance, and we want to excel at
      it
    - working on something that pleases helps increase efficiency
    - technical correctness seems an obvious requirement for a tool
      to gain popularity, but there's also "the feeling"
    - cost-benefit is not always about time; cleanliness of the
      approach and pride in the outcome can be important factors
    - we dislike piggy-backing behaviour on-top of other, unrelated
      behaviour; the project homepage is not part of the package
      description
    - we seem to somewhat unreasonably prefer the RFC822-like format
      because it's "Debianish"
    - the quality of a tool will always win
  }}}

16. Examples vs. documentation

  Examples are more practical and easier to grasp than documentation,
  and many of us (not just beginners) work off templates. However,
  examples do not replace documentation.

  {{{ supporting statements
    - examples are more accessible than docs
    - many, if not most people work off templates, not only newcomers
    - examples are not enough, they do not replace documentation
    - examples are useful at bootstrapping one back into the mindset
      after not having used the tool for a while
  }}}

17. Sedimentation

  Ground-breaking ideas need time to spread. People reject ideas
  until they understand the underlying problems, are able to
  formulate them in their head, and identify the benefits of a
```

```
solution. In order for everyone to understand the principles,
explanations must cover multiple perspectives.

{{{ supporting statements
  - new ideas are often rejected/discarded as overkill until people
    have understood the underlying problem, formulated it in their
    heads, and identified the benefits
  - ground-breaking tools/ideas, if solid, will slowly sediment
    into a group, making adoption somewhat unconscious
  - major mindset shifts require multiple explanations from
    multiple angles so that everyone gets a chance to grasp them
}}}
```

18. Resistance

```
While initial resistance to an idea can help separate good from bad
ideas, pushing too hard, or taking away the option to continue with
the old method, will yield further resistance. Resistance can only
be met with conversion instructions, support offers, and patience.
Lack of interest is closely related to resistance.

{{{ supporting statements
  - pushing/standardising a tool too hard can yield resistance,
    people want the option to follow or to stay put
  - resistance is good up to a point as it helps people separate
    good from bad ideas
  - to defeat resistance, conversion instructions, support offers,
    and patience are required
  - lack of interest and resistance are closely related and can
    both be addressed
}}}
```

19. Modularity

```
Highly granular solutions (e.g. debhelper, autotools) require
repeating the same sequence of instructions for standard tasks.
Such code duplication is often regarded as bad, but it also makes
it easier to understand what the steps are and what is happening
overall. Monolithic solutions, on the other hand, are frequently
inflexible and obscure. The ideal solution is somewhere in-between,
requiring one to only track local modifications, and therefore
minimising boilerplate, as well as divergence from the baseline.

{{{ supporting statements
  - one problem with the traditional debhelper approach is that
    only a subset of helpers is run; cdbs alwayes executes all
    steps and is thus predictable
  - dh_* calls could be regarded as code duplication
  - there is middle-ground between a monolithic, inflexible script,
    and a toolkit that requires the same sequence of tool
    invocations to be duplicated across the entire archive
  - systems that only expose local changes and hide the boilerplate
    seem future-prof
  - build systems allow to keep the archive closer to policy
```

```
    without manual interaction
  - factoring processes falls short when the resulting API is
    insufficiently expressive
  - code duplication can be more explicit and clearer
  - autotools and debhelper come from different angles but are both
    trying to iteratively improve the balance between portability
    and abstraction/fine control
  - when can a sequence of steps be turned into a simple one-shot
    action
}}}
```

20. Sustainability

    We want to put tools/techniques to use in our own ways, rather than change our ways to fit them. We want confidence that a tool/technique develops in the right direction, and that its maintainers incorporate feedback, allowing users to influence that direction; otherwise, time invested now might be lost later. Personal friction with/dislike of a tool's author can get in the way of cooperation.

    ```
    {{{ supporting statements
      - people adapt tools to their workflows instead of the other way
        around
      - adopters want influence, not a finished product, possibly
        shaped too much by the authors' development process
      - tools need a strong sense of direction, users must be confident
        that this direction is desirable, and authors need to
        incorporate their feedback
      - personal dislike can translate into inability to cooperate and
        unwillingness to adopt tools by that person
      - other things being equal, people usually prefer to use what
        other people are using, as it's safer, more likely to be
        maintained, or yield migration paths
      - people consider whether adopting a tool will allow others to
        collaborate, although this can be a weak criterion
    }}}
    ```

21. Network effects

    Collaboration in teams restrains people in their choice of tool/technique due to the effects on others and the barrier of entry. Tools that are optional do not raise the barrier of entry or force people out; if the tools are visible nonetheless, and fill a niche, the adoption rate can benefit from the exposure. Teams can also develop a group tolerance to software faults over time: individual annoyances decrease in relevance as the group learns to live with or work around the faults. All these network effects seem stronger in teams working on diverse software, as opposed to teams centred around uniform software, e.g. written in the same programming language. All the same considerations apply towards downstream, with respect to other people reusing the work.

    ```
    {{{ supporting statements
      - people are much restrained in their choice if it has an effect
    ```

```
    on others
  - given the different levels of knowledge, the final solution
    adopted by a group is not always the technically best one
  - developers working with others need to exercise care when
    changing processes to not conflict with or force their approach
    upon others
  - teams tend to be fairly conservative with adoptions of new tool
    due to inertia or to avoid creating too high a barrier of entry
  - code-centric teams have an easier time standardising than
    code-agnostic teams
  - a tool that "scales down" so that people can still contribute
    without using it never cuts people off
  - facilitate contributions by downstream
  - upstream's choices influence maintainers' decisions
  - as Debian, we ought to take care not to impose tools on those
    who reuse our packages
  - tools with dependencies that would need to be installed might
    be disfavoured for the security hazard of installing additional
    packages
  - availability in stable
  - fault tolerance in a tool decreases with time, group tolerance
    may take over
  - the adoption rate of visible tools/techniques benefit from the
    exposure, especially if the tool/technique just seems like a
    good idea and/or fills a niche
}}}
```

22. Chunking

    People have established workflows and adapt tools to fit those. They (might) want to improve/evolve those workflows, rather than replace/revolutionise them. Free time is fragmented and the adoption of a new tool/technique often requires to commit oneself over larger chunks of time. Revolutionary ideas are hard to spot initially, but if they can be identified, their supporters should try to break them down into an evolution with individual components that can be adopted piecewise.

```
    {{{ supporting statements
      - free time is often fragmented, so it's easier to do (and keep
        doing) tasks that require less time per instance; adopting a
        new tool is bound to take larger chunks of time
      - changes need testing, an additional burden and cost-factor
      - people are looking to improve/evolutionise their current
        workflow, not replace/revolutionise it
      - assuming one can spot a revolution up front, one ought to try
        to convert it to an evolution with piecemeal deployments
      - revolutions are not always well-understood by their authors,
        but develop as an idea unfolds
      - it is much easier to adopt a technique piecewise, over time;
        modular (as opposed to monolithic) tools/techniques are easier
        to adopt
    }}}
```

```
23. Standards

    Standards should define end results, not processes or tools. They
    are a good way to drive evaluation, though not necessarily
    adoption. Standards evolve or result from long-term network
    effects, they are usually not simply defined. Yet, by defining
    standards, one can enable progress. It is necessary to communicate
    standards and changes to everyone. Using an already wide-spread
    tool can be an effective vehicle, but care must be taken not to
    introduce problems.

    {{{ supporting statements
      - a sense of momentum can cause people to perceive something as
        standard, provided the tool lives up to that perception
      - settings standards is the best way to get people to adopt
      - forcing anyone is never an option in collaboration
      - evaluation is driven by standards, adoption not always as
        plenty of Debian people are perfectionists or have their
        established workflows
      - decisions from the top oblige people to move on, but driving
        change can be a lengthy process which requires careful
        preparation
      - regulars do not periodically reread standard project
        documentation
      - standards documentation have substantial influences,
        particularly over the behaviour of new developers
      - the trick to team standardisation is to standardise the right
        things: interfaces, not tools. build dependencies are just one
        way to achieve something
      - The Debian policy only describes the end result, not the
        process
      - the standards documents are slow, they do not include recent
        best practices and are thus not a good source for study
      - standardisation is often the result of a long-term network
        effect, or an evolutionary process
      - the proliferation of *-buildpackage scripts has led to some
        interesting ideas which may not have happened with a single
        vcs-buildpackage
      - noone can impose tools on volunteers, but some tools are so
        strongly suggested (e.g. chroot building) that the effect is
        the same in the end
      - sometimes it is possible for an individual to shoulder the
        burden for everyone's benefit
      - introducing a tool/technique by way of an already wide-spread
        tool yields wide-spread adoption if there are no problems
        associated with it
    }}}

24. First impression

    The first impression could yield further investigation by the
    potential adopter. Stating plainly what a tool's purpose is and
    what it does, documenting its benefits, and providing examples and
    tutorials facilitates the ensuing investigation. A concise
```

```
    description, which does not assume high levels of technical
    knowledge or skills, can help form a positive first impression.

    {{{ supporting statements
      - tools whose benefit is obvious or documented very clearly will
        be more widely adopted than tools whose benefit is more vague
      - negative first impression is hard to overcome and may spread
        through the grapevine; it is important to get the marketing
        right early to ensure positive first impressions of perceived
        benefit
      - a good first impression can be the foot in the door and lead to
        further investigation of the tool/technique
      - tools should have a clearly defined purpose
      - a good, concise description which does not assume high levels
        of knowledge or skills, can help form a proper first impression
      - the balance between concision vs. history/reasoning when
        telling people about (or to use) a tool
    }}}


--------------------------------------------------------------------------

For your convenience, the following is a list of the paragraph keywords
with one-line summaries:

 1. Return-on-investment: value of future benefits
 2. "Peercolation": people networks and flow of information
 3. Maturity: usability and stability without tracking development
 4. Trialability: ease of trying out a tool/technique
 5. Genericity: re-use of a tool/technique for other tasks
 6. Quality assurance: automated tests
 7. Transparency: automation and associated loss of control
 8. Uniformity: benefits of and needs for uniformity
 9. Quality documentation: maintenance, target audience, and verbosity
10. Scaled use: various degrees of use and gradual migrations
11. Consensus: just the right amount of discussion and consensus
12. Cost-benefit: investing time vs. getting things done
13. Marketing: increase awareness of a tool/technique
14. Compatibility: minimal effort for change
15. Elegance: personal preferences and irrationality
16. Examples vs. documentation: role of examples and templates
17. Sedimentation: the nature of spread of revolutionary ideas
18. Resistance: benefits of, and dealing with resistance
19. Modularity: granularity vs. monolithic solutions
20. Sustainability: confidence in the decision for a tool/technique
21. Network effects: dampening and enabling effects of teams
22. Chunking: revolutions and evolutions, and peace-wise adoption
23. Standards: creation and effects of standards
24. First impression: make sure the first impression is good

--------------------------------------------------------------------------

As usual, if you have any questions, please don't hesitate to ask at
any time. Email works, and additional contact details are on the
```

```
research info page, http://phd.martin-krafft.net/info.html

Thank you for your participation in this study,
[...]

% vim:foldmethod=marker:foldlevel=0
```

C.2.3. Feedback

Listing C.8: *The final message seeking feedback from the panelists.*

```
Subject: Feedback: impressions and suggestions

[GREETING] [FIRST NAME],

You have recently participated in my Delphi study on the influences on
adoption behaviour among Debian package maintainers. Unless my records
deceive me, I should have acknowledged your last reply and
participation with a separate message, and obtained your permission to
publish your comments. If you have not received such a message, please
let me know. The distribution of the promised compensation gadgets is
under way, but I don't have details yet.

With this message, I am seeking any feedback on the study you might
have. There is no obligation for you to reply. Nevertheless, I would
highly value and appreciate any input you might have.

This study was the first application of the Delphi method in a FLOSS
environment, and since the Delphi method fits so well with that
context, I strive to facilitate future applications by making all data
and the details of the approach available. In addition, I intend to
provide a thorough analysis of what worked well, and what could have
been improved.

Your input on this matter is of utmost significance. Many of you have
shared comments and suggestions throughout the study. If you have any
more feedback, please do not hesitate to write in.

 1. Do you think that e-mail was the right choice of medium for
    conducting the study? Do you think that e-mail would be a good
    choice for a study among people who might have less experience with
    e-mail than you do?
 2. How would it have affected your participation if we had used a
    synchronous communication medium, e.g. a web application, to
    conduct the study?
 3. Throughout the study, I called for free-form answers, giving you
    all the freedom and flexibility needed to express your thoughts.
    Can you imagine aspects of this study that would have profited from
    limiting the domain of possible responses?
 4. You surely remember the large volume of data at the start of the
```

```
    second round, and that it took you a lot more than one hour to
    process those. This was unforeseen and a planning mistake on my
    part. Can you fathom a different strategy for the first two rounds,
    which might have resulted in less data, and a lesser time
    requirement?
 5. The third round was long delayed. What effect do you think this
    delay had on your participation?
 6. Some participants found the third and last round to be the most
    difficult and least rewarding phase of the study, because I was
    asking you to pick the most salient influences out of a set of
    influences that were, in some ways, already the most important ones
    from the previous two rounds. How would you have done it
    differently?
 7. I kept the panel roster secret and you did not know who else was
    participating. How would you rate the effect of this anonymity on
    your participation?
 8. Is there anything else you would like to remark or suggest?

Thank you again for your participation, and any feedback you might
have.
[...]
```

C.3. Merging e-mails

In section 5.5, I discussed a number of considerations I made when sending e-mails to Debian Contributors (DCs). One of these considerations was to avoid the character of mass communication because "messages addressed to a group of people are less likely to be read than messages adressed to one personally." I chose to send individualised e-mails at all stages of the study to increase the response rates, and from the numbers presented in the detailed research account in chapter 6, this was a success.

The number of e-mails I sent throughout forced me to employ automated means to create the individual messages. Unfortunately, I could not find a usable solution and went ahead to implement one. The result is a small script I called `822merge` written in Python, which is mainly a wrapper around the Cheetah templating system[2], with some extensions for e-mail. The following is a quick illustration of how it works.

The script combines two input files: a template, and a data file. For instance, given the template shown in listing C.9 on the next page and the data file show in listing C.10 on the facing page, the script generates the e-mail message shown in listing C.11 on the next page.

[2] `http://www.cheetahtemplate.org/`

Listing C.9: *A simple template used by 822merge.*

```
From: Martin F. Krafft <phd@martin-krafft.net>
To: $rfc2231_utf8_fullname <$emailaddress>
Subject: Did you know?
User-Agent: madduck's sub-zero not-quite-mass-mailer

$greeting $firstname,

You are me, and I am you!
Martin
```

Listing C.10: *A simple data file used by 822merge.*

```
fullname: Martin Krafft
emailaddress: martin.krafft@ul.ie
greeting: Salve
firstname: self
```

Listing C.11: *The result of the merge by 822merge.*

```
From: Martin F. Krafft <phd@martin-krafft.net>
To: Martin Krafft <martin.krafft@ul.ie>
Subject: Did you know?
User-Agent: madduck's sub-zero not-quite-mass-mailer

Salve self,

You are me, and I am you!
Martin
```

This is no rocket science, and only the use of `$rfc2231_utf8_fullname` is worth a mention: `822merge` allows field names to be prefixed with `rfc2231_`*`encoding`*`_`, which causes the contents to be encoded according to RFC 2231, using the given encoding, which allows me to use non-ASCII characters in the headers and hence properly address the e-mails to participants with special letters in their names. Field names could similarly be prefixed with `html_` to escape them for use in an HTML source.

The latter is motivated by my choice for HTML as the source format, and the generation of the plain-text e-mails from HTML with the dump-functionality of w3m, because it would nicely reflow the text, format lists and rules nicely, and allow me to hide comments in the source that would not appear in the final result.

The last component of my personalised e-mail creation tool is a `Makefile`, which turns the plain-text output by `822merge` into a full-fletched MIME-formatted e-mail message (using mime-construct), with the ability to attach per-recipient as well as common attachments. This made it possible to include previous (individual) correspondence with e-mails for easier reference by the recipients, but also to distribute common information, such as the background and approach overview sections of my research webpage, which I attached to the initial e-mail message so that people could obtain more information about my endeavour without leaving their e-mail programmes. Finally, the `Makefile` added the functionality to sign each message with GNU Privacy Guard (GPG) to remove further mass-e-mail character.

Both scripts are available under the terms of the Artistic Licence 2.0 and may be downloaded from `http://phd.martin-krafft.net/tools/`.

Appendix D

Acronyms & abbreviations

AC	adoptive capacity
APT	Advanced Package Tool
ABI	Application Binary Interface
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BSP	Bug Squashing Party
BTS	Bug Tracking System (for Debian: `http://bugs.debian.org`)
CDBS	Common Debian Build System
COSS	Commercial Open-Source Software
CSCW	Computer Supported Cooperative Work
DC	Debian Contributor
DD	Debian Developer
DEB	The Debian package file format
DEP	Debian Enhancement Proposals (`http://dep.debian.net`)
DFSG	Debian Free Software Guidelines
DM	Debian Maintainer
DPL	Debian Project Leader

DVCS distributed version control system

FOSS Free/Open-Source Software

FLOSS Free/Libre/Open-Source Software

FSF Free Software Foundation (`http://www.fsf.org`)

GNU GNU is not Unix (a recursive acronym)

GPG GNU Privacy Guard (`http://en.wikipedia.org/wiki/Gpg` [19 Jun 2008])

GPL GNU Public Licence

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

INGRID innovation grid [Kearns, 1992]

IRC Internet relay chat

IS information systems

IT information technology

KISS "keep it simple, stupid" (`http://en.wikipedia.org/wiki/KISS_principle` [7 Aug 2009])

KRNW knowledge resource nomination worksheet

MIME Multipurpose Internet Mail Extensions

NIH "Not-Invented Here" [Katz and Allen, 1982]

NM New Maintainer

NMU non-maintainer upload

OSI Open Source Initiative (`http://www.opensource.org`)

OSS Open-Source Software

PGP Pretty Good Privacy (`http://en.wikipedia.org/wiki/Pretty_Good_Privacy` [19 Jun 2008])

PTS Package Tracking System

RC release-critical

RFC Request For Comments (*de facto* standards of the Internet)

SPI Software in the Public Interest (`http://spi-inc.org`)

TAM Technology Acceptance Model

TRA	Theory of Reasoned Action
TPB	Theory of Planned Behavior
URL	Uniform Resource Locator
UTAUT	Unified Theory of Acceptance and Use of Technology
VoIP	Voice-over-IP
VCS	version control system
XML	Extensible Markup Language