How Do Free/Open Source Developers Pick Their Tools? A Delphi Study of the Debian Project

Martin F. Krafft
Debian Developer
Munich, Germany
mail@martin-krafft.net

Klaas-Jan Stol
Lero—the Irish Software Research Centre, University of Limerick, Ireland
klaas-jan.stol@lero.ie

Brian Fitzgerald
Lero—the Irish Software Research Centre, University of Limerick, Ireland
bf@lero.ie

ABSTRACT
Free and Open Source Software (FOSS) has come to play a critical role in the global software industry. Organizations are widely adopting FOSS and interacting with open source communities, and hence organizations have a considerable interest in seeing these communities flourishing. Little research has focused on the tools used to develop that software. Given the absence of formal mandate that would appear in traditional organizations, an open question is what influences a FOSS contributor's decision to adopt a tool and how workflows get established in FOSS teams. In this paper we report on a Delphi study conducted in the Debian Project, one of the largest FOSS projects. Drawing from data collected in three phases from a panel of 21 carefully selected and well-informed participants, we identified 15 factors that affect decisions to adopt tools and relate those to existing models of innovation and diffusion.

CCS Concepts
• **Software and its engineering** → *Software maintenance tools; Collaboration in software development*

Keywords
Free/open source software, tools, Delphi study, qualitative study

1. INTRODUCTION
Tools play an essential part in software development [15, 32], and research in this area has been extensive [18]. New tools and technologies are continuously emerging, which in turn affect the way software is developed. Much research on software tools and environments has focused on industrial software engineering development contexts [4, 20, 30]. However, a significant development has been the rise of Free and Open Source Software (FOSS), which has gained significant attention from both researchers and practitioners in the past two decades [13]. Since then, FOSS has been widely adopted in industry [19], and represents an important part of many software products. Therefore, a good understanding of how such projects work is essential. While there has been much research on FOSS, the use and selection of software development tools in FOSS has received very little attention [5], despite the fact that tools play a critical role in FOSS development.

One highly successful FOSS project is the Debian Project, founded in 1993, and run entirely as a development community comprising over 2,500 volunteers. As such, it is one of the largest FOSS projects [1]. The Debian project produces several operating systems, of which Debian GNU/Linux is the most popular, providing over 43,000 packages for ten different hardware architectures. Furthermore, the Debian System is the basis for around 150 derivative distributions, including the popular company-controlled Ubuntu distribution produced by Canonical. After more than 20 years in existence, some of the project's processes are still difficult to scale, which is needed to meet the tremendous growth the project has seen. Activities such as library transitions currently require dozens of contributors to work hand-in-hand, and often stall because of bottlenecks. Day-to-day tasks are often tedious and error-prone, relying on developers to maintain consistency: keeping track of patches, triaging bugs, following policy changes, and working with both 'upstream' projects (the original source projects included in a Debian distribution), and 'downstream' derivatives, to name just a few challenges. Looking at the way these processes are currently handled, it is surprising that contributors of a system as technically sound and universally applicable as Debian are still doing manually what a computer should be doing for them. Tasks such as those mentioned above could be streamlined and optimized to avoid redundancy and points of failure due to their brittle integration.

Improved tools and techniques are necessary to increase the efficiency of the Debian Project's contributors. Debian is a volunteer-controlled project—most of its contributors are not paid to work on the project, and have therefore limited time available as most of them have daytime jobs. Some sophisticated tools and techniques already exist and new technologies emerge frequently, but these are not readily adopted. Many contributors try to identify and communicate better approaches, but an in-depth understanding of individual adopters' behavior is lacking. As a result, new ideas only slowly rise to become competitors with existing approaches. Software technology transfer has been defined as a considerable concern [36, 38], but a deep understanding of the factors that influence tool adoption among voluntary FOSS developers is largely missing as of yet. Existing frameworks and theories such as the Technology Adoption Model (TAM) tend to focus exclusively on either individuals or non-volunteer (commercial or not-for-profit) organizations, and therefore are unsuitable to explain adoption in volunteer-based communities.

This research focuses on a challenge commonly found in volunteer-driven communities: a lack of authoritarian structures makes it impossible to mandate 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, a decision to change lies with each individual, and project-wide change thus depends on the entire community. Given the large size of the Debian Project, we chose to focus on one specific area that is of

ICSE '16 Companion, May 14 - 22, 2016, Austin, TX, USA

ACM 978-1-4503-4205-6/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889248

particular importance to Debian's success as a distribution containing tens of thousands of software packages: software packaging. Given the critical importance of tools in large FOSS projects and the lack of insight on how these tools are selected in volunteer-driven projects, we investigated the following question:

Research Question: What factors influence the Debian package maintainers' decision to adopt new tools or techniques?

This study focused specifically on the Debian Project as the primary author is a long-standing contributor to this project [23]. The paper proceeds with a background discussion on the Debian project and innovation in FOSS projects (Sec. 2). We then present the details of the Delphi study that was conducted (Sec. 3). This is followed by a presentation of the results of our study, namely a set of factors that affect the adoption of tools and techniques (Sec. 4). The paper continues with a discussion of the findings, the implications for research and practice, as well as the threats to validity of this study, followed by an outlook on future work (Sec. 5).

2. BACKGROUND AND RELATED WORK

2.1 Package Management in Debian

Debian as a FOSS project takes an extraordinary and somewhat radical approach, in promising that *"the Debian system and all its components will be free,"* and that the project *"will never make the system require the use of a non-free component"* [9]. Traditionally, the Debian Project distinguishes 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 emerged. In the Debian Project, not only developers 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 results in higher degrees of interdependencies between individuals. One contributor's decision in favor of or against a tool may have a significant effect on another contributor's decision to adopt it, and on this level, it matters little who is an official project member and who is not. The Debian Project is an organization driven entirely by volunteers. The project does not pay any of its developers, nor does it let its sponsors or its legal entity have any influence in the project's technical interests.

The packaging workflows used by Debian contributors are suboptimal; common methods are minimally integrated at best, and package maintainers lose time and energy on repetitive, error-prone tasks. This causes individual frustration and slows project progress. Ironically, improved tools featuring better integration, collaboration facilities, and greater degrees of automation do exist. In the two decades since Debian's foundation, a number of packaging automation tools have been widely adopted, so large-scale workflow improvements do happen. However, countless others never reached significant levels of use, and this begs the question as to what factors might be at play. Recently, the project has seen strong trends towards techniques supporting distributed development, which promise solutions to many of the (centralized) deadlocks and bottlenecks in the project. Yet, project-wide acceptance has been slow, for reasons that are not always obvious.

2.2 Explaining Adoption of New Tools

Tools, and tool integration specifically has gained sustained attention from researchers studying traditional organizational contexts. However, in the FOSS context, Crowston et al. [5] observed that *"surprisingly little research has examined the use of different software development tools."* Previously, Oezbek and Prechelt sought a suitable research method for studying process innovation in FOSS projects [37]. However, their focus was specifically on innovation by people (e.g., researchers) that are *not* members of a FOSS community. Shaikh and Cornford studied the adoption of a commercial version control system (BitKeeper) in the Linux kernel project in 2002 [45]. The key reason why CVS (the most popular version control system (VCS) at the time) was not adopted was technical, as CVS did not have all the features that Linus Torvalds required as benevolent dictator of the Linux kernel project. Torvalds subsequently started development on Git [46], a popular distributed VCS.

Redwine and Riddle discuss a number of factors that either inhibit or facilitate software engineering technology maturation [40]. Some critical factors they discuss are a clear recognition of need, tuneability, and management commitment, and inhibitors include high cost and contracting disincentives. However, Redwine and Riddle focus on the *maturity* of technologies (i.e., the *product*), rather than the *process* that influences adoption of new tools. Also, factors such as management support and contracting disincentives do not play a role in volunteer-driven FOSS projects.

Numerous frameworks have been proposed to understand and explain technology diffusion and adoption, which can roughly be divided into two groups: (1) frameworks and theories that consider adoption at the *individual* level (e.g. Rogers [41]), and (2) those that focus on the *organizational* level (e.g. Kwon and Zmud [25]).

Perhaps the best-known model in the first category (focusing on the individual) is the Technology Acceptance Model (TAM) [8], which has been referred to as *"the most influential and commonly employed theory for describing an individual's acceptance of information systems"* [27]. The model was highly revolutionary at the time of its conception, and has since been extended in several ways. However, it has also been criticized for its limitations [2], one of which is that it is based only on attitude and behavior.

Another model is the Task-Technology Fit (TTF) model [16], which states that the 'fit' between a task and technology is *"the matching of the functional capability of available software with the activity demands of the task"* [10], but this ignores human and social factors such as personal preferences of developers. Others focus on specific contexts; for example, both Rossi et al. [42], and Fitzgerald et al. [14] present frameworks to explain FOSS adoption in the public sector. Eckhardt et al. studied the impact of social influences, referring to the influences of colleagues and other departments in an organization [12].

To summarize, tool adoption in FOSS communities, consisting of independent volunteers, cannot be explained by existing theories and frameworks for a number of reasons:

- **Exclusive perspective on individuals or organizations**. Tool adoption in FOSS communities does not happen exclusively on an individual or organization level.
- **Unsuitable for volunteer-driven communities**. Existing frameworks focusing on organizational adoption are not suitable as volunteer-driven FOSS communities have no formal authoritarian leadership ('management') or business-focus [48]; there are no change agents [38].
- **Assuming independence of adopters**. Existing frameworks tend to assume independence among adopters, i.e. absence of network effects. FOSS communities, however, rely on close collaboration among its contributors and therefore imply a level of dependency among project members.
- **General focus on 'innovations.'** Many frameworks consider the diffusion and adoption of 'innovations' in a general sense, but not software tools specifically.

It is also worth noting that many of the frameworks proposed are based on observations of "historical accounts" and attempt to provide a generalized process of technology adoption (e.g. [40]); others are based on a set of factors that have been identified a priori before any feedback is solicited from experts (e.g. [42]). While these approaches are not invalid, they ignore perhaps the most important stakeholders, namely the volunteers adopting the technology.

3. RESEARCH DESIGN

This paper reports the results of a Delphi study modeled on the Policy Delphi approach. The Delphi method has seen very limited adoption in the software engineering discipline. We first present a brief discussion of the method. We then discuss how the Delphi panel was selected, and provide details of the Delphi process including data collection and analysis. A more detailed description is offered in the first author's dissertation [24].

3.1 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"* [6]. The original Delphi study sought to investigate the impact of technology on warfare and was exploratory in nature [7]. 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"* [28]. It is an instance of moderated communication: a facilitator serves a series of questions to the participants, who return their answers to the facilitator. The answers are anonymized, 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.

Since the Delphi approach was originally put forth by Dalkey and Helmer [6], several researchers have modified the method resulting in a number of variants. This study's design was based on a *Policy Delphi* approach [47]. Whereas the Delphi method traditionally aimed at gaining consensus, the policy Delphi aims *"to support decisions by structuring and discussing the diverse views of the 'preferred future' "* [22] and *"seeks to generate the strongest possible opposing views"* regarding an issue [47]—or what one might call 'dissensus' to differentiate from consensus-seeking Delphi studies [29]. Rather than a tool for decision making, the policy Delphi can be used to *"generate options and suggest alternative courses of action for consideration"* [34]. The aim of this study was to understand the process of tool adoption in a FOSS community; a Policy Delphi was deemed appropriate, as it would help to suggest 'courses of action' for FOSS developers.

3.2 Selection of Delphi Panelists

Selecting the right participants for a Delphi panel is key to a successful study. Okoli and Pawlowski called panelist selection *"perhaps the most important yet most neglected aspect of the Delphi method"* [33] and Judd claimed that deciding 'who is an expert' is *"the single most confounding factor in panel selection"* [21]. An important consideration in the study design was that the primary author is a long-time member of the Debian community, and it was important to include people that he was not closely acquainted with, and with whom the research issues had not been discussed prior to this study.

To select members for the Delphi panel, project members were identified who took part in team efforts, or otherwise cooperated with others in the project. These were identified through several channels, such as scanning the various mailing lists, IRC (Internet Relay Chat) logs, the package maintainer database, and notes from various meetings and discussions at Debian conferences. A total of 162 people were asked to nominate colleagues whom they deemed to have deep insights into the adoption behavior of Debian contributors, along with a short reason for nomination. Self-nominations were explicitly mentioned as an option. From this, 98 responses were received, with a total of 429 nominations. From the list of nominations, 48 people were identified who received three nominations; of those, 10 were excluded due to unavailability. From 50 other nominees who had two nominations each, five were manually selected whose nominations made them particularly interesting candidates, resulting in a group of 43 people. In order to determine the nature of the candidate's project work and collaboration within the project, candidates were asked to provide some information about their involvement, resulting in 36 responses. Based on these responses, candidates were organized on four dimensions, namely whether or not a candidate (1) was involved as a team player; (2) had a uniform set of tasks; (3) used uniform tools; and (4) was interested in workflow improvement. Stratified purposeful sampling [35] was employed to select candidates that represented maximum diversity on the dimensions in Fig. 1 [11], as is desirable for a Delphi. In total, 21 panelists were selected that had 'extreme' profiles on the four dimensions. This panel size lies well within the recommended range of 15-30 carefully selected participants [29].

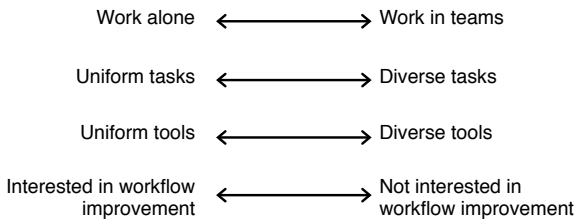

Figure 1. Four dimensions for Delphi panelist selection

3.3 Data Collection and Analysis

The Delphi study took several months to complete, and consisted of four phases (see Fig. 2), which are described next. Three rounds were carried out as part of the Delphi study, which was followed by a 'reduction' phase so as to identify a parsimonious set of factors.

Phase 1: Brainstorming. In the first phase, a brainstorming round was conducted to obtain a broad sense of the factors that shape package maintainers' decisions regarding the adoption or rejection of tools and techniques. The aim of this phase was to exhaustively seek factors. Participants were asked to identify at least six factors that influence the decision to adopt or reject new tools or techniques. In order to encourage participants to be as open and frank as possible, anonymity was assured.

The first phase resulted in responses totaling 3,500 lines. To make the discussion of these responses manageable, responses were organized in a number of categories—Schmidt suggests to use up to 20 categories [44]. A first round of analysis resulted in a set of 104 keywords. These were reduced to a set of 40 categories using a concept-mapping approach described by Novak and Cañas [31]. Through further analysis 14 categories were merged, resulting in 26 remaining categories. As the primary researcher deemed 26 categories too many for the next phase, three colleagues were invited to a card-sorting exercise [43]. This resulted in a further reduction to 15 categories; the 3,500 lines of responses were reduced to 1,300 lines, and organized into the 15 categories.

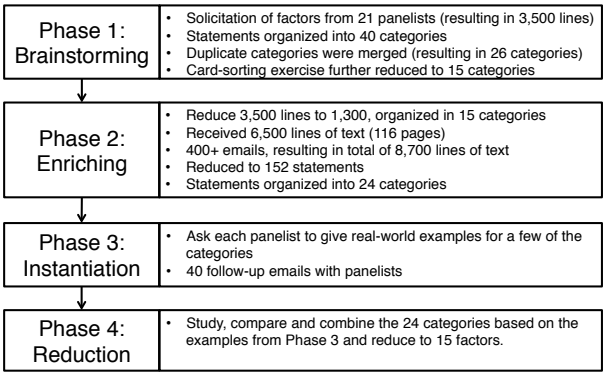

Figure 2. Phases of the Delphi study

Phase 2: Enrichment. The goal of the second phase of the study was to enrich the data by seeking further qualifications from the panel, identifying the statements that panelists commonly agreed on, and identifying any discrepancies between the panelists' judgments. To that end, the 15 categories of related statements resulting from the first phase were sent to the panel. Specifically, panelists were invited to refute factors they did not agree with, identify links between comments from other panelists, and provide additional information as they saw fit. The instructions to the panelists encouraged them to read critically as their agreement would be assumed by default. By only asking for qualifications and refutations, the intention was to *enrich* the data gathered thus far without excessively burdening the panelists.

The responses in this second round comprised approximately 6,500 lines of text (116 pages of text). Where several statements were related or contradicting, these statements were presented to the panelists in follow-up emails in order to seek further clarification. In total, almost 400 emails were exchanged (one email per issue). Relevant information retrieved from this process was inserted into the list of statements, resulting in a total of approximately 8,700 lines of text (156 pages of text).

This set of data was analyzed by identifying non-obvious and insightful statements by the panel. Long statements were shortened while paying specific attention to capturing the context and essence without losing any critical detail. This resulted in 281 statements. These were subsequently organized into groups of related statements, first by identifying redundant statements (resulting in 152 remaining statements), and subsequently by organizing them in categories of related statements. This resulted in 24 categories. For each category, a short descriptive paragraph was subsequently proof read by three colleagues.

Phase 3: Instantiation. The goal of the third phase was to identify the salient factors to adoption or rejection decisions among Debian contributors. Rather than seeking a ranking of factors or agreement among the panelists as would be typical for a traditional Delphi study, panelists were requested to provide *"stories from the trenches."* A ranking of factors would be 'weak' given the diversity among the panelists, as many categories would rank closely to each other. (If, on the other hand, the panel had not been as diverse, the ranking would not have been representative of the project).

Panelists were asked to select the three most important factors they had experienced in the context of their packaging work in the Debian project, and share details about how these factors had previously manifested and were expected to do so again in their immediate environments. Further clarification was sought through 40 follow-up emails with the panelists.

Phase 4: Reduction. In the fourth and final phase, we sought to achieve parsimony by combining factors that were similar in essence. For example, two factors that emerged from an earlier phase of the study were 'modularity' and 'transparency.' The former refers to the level of granularity (fine vs. coarse-grained), which affects the ability for a maintainer to follow the various steps. A tool that defines an interface at a high level of abstraction (i.e. a coarse-grained interface) exhibits a lower level of transparency because it is harder to follow the internal mechanisms of the tool. Because these two factors were so closely related as two sides of the same coin, these were combined into a joint factor.

4. RESULTS OF THE DELPHI STUDY

The Delphi study resulted in a set of 15 factors presented below. Each factor is summarized followed by an elaborated discussion.

Factor 1: Sedimentation

> New ideas can take time to gain widespread acceptance. People reject ideas until they understand the underlying problems, are able to formulate them succinctly, and identify the benefits of a solution.

For new technologies to be accepted, awareness of such technologies must grow and the benefits they offer must be clear, but this process can take a significant amount of time. One panelist mentioned the example of distributed version control systems (DVCS): *"DVCSs have been around for years, and it's only now (last 2 years) that we see a real growth in users."* Technologies may seem to be too revolutionary at first for the wider community to perceive them as 'ready' for adoption. New technological solutions may address problems that people may not have clearly formulated 'in their heads,' or 'seem irrelevant.' Through a process of 'sedimentation,' a new technology slowly gains recognition, and at some point people may become sufficiently comfortable to start using it. However, this process could take years.

Factor 2: Marketing

> Using appropriate channels and content, active promotion or marketing of a new tool or technique can feed excitement and exposure of the innovation, and can stimulate others to evaluate them.

One panelist explained: *"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."* However, it is important to use appropriate channels. While success stories and a positive attitude are stimulating, especially when needs are *met* instead of *created*, inappropriate corporate links (given the 'free' nature of Debian, and FOSS projects in general), premature promotion and TV-style marketing on the other hand, can have negative effects and ought to be avoided.

The Debian Project is not lacking any communication media—an abundant and diverse collection of communication channels is available which can facilitate the spread of information, including mailing lists, blogs, and IRC channels. This forces volunteers (with limited time) to select a subset to concentrate on, which potentially creates smaller, well connected 'cliques' or 'tribes' who may not interact with one another.

While mailing lists seem to be the first choice for spreading information, blogs can have a huge impact. One panelist recalled how the project's extensive adoption of Git was partly due to the 'buzz' on Planet Debian (an aggregate of blogs of Debian developers) on this topic, despite the fact that other systems such as Mercurial and Bazaar had a reputation for being easier to use. An appropriate marketing approach should consider a number of aspects, including frequency (repeated exposure), timing, the choice

of channels to use, and the content of marketing message. Finally, care should be taken not to 'overhype' so as to prevent disappointing potential adopters.

Factor 3: 'Peercolation'

> Information spreads through networks of peers, and information that flows between peers is often accorded a higher weight. Those with significant experience in an area and who can clearly explain a tool's benefits, get more respect. People tend to favor peers they trust.

People tend to favor peers they trust, or with whom they have overlaps in interest or heritage. The term 'peercolation' was coined by one of the panelists to describe the percolation of information (and particularly knowledge of innovations) through networks of trusted peers. While related to 'marketing' (discussed above), one panelist clearly distinguished the two concepts: *"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."*

One panelist argued that most Debian community members prefer to use 'standard' tools. There is a general perception that there are many good ways 'of doing things,' and that anything that is broadly used is likely good enough. In other words, tools that have a significant momentum and are widely adopted are likely to get more support from others. Another case where people depend on their peers is when there is little time for an individual evaluation of a tool, and they must then rely on trust to shortcut the evaluation. As one panelist illustrated: *"knowing what people you trust are using or interested in is a major factor."* Others still preferred to evaluate a new tool themselves. One factor at play here is the credibility, or status, of peers. Most panelists agreed that messages from respected peers weigh heavier than messages from others.

Finally, an innovation's pedigree may also affect its adoption, related to the question of why and by whom a tool is developed. For example, one panelist claimed that Bazaar (a DVCS) had *"a bad start in Debian,"* because it was developed by Canonical, the corporate entity sponsoring Ubuntu. This is of particular importance in a FOSS project such as Debian, given its core principle of independence. While anti-corporate feelings were not generally shared among the panelists, there are some within the Debian community that have some anti-corporate bias. Pedigree is not of exclusive importance to involve corporate involvement; FOSS tools, too, may be critically viewed. One panelist referred to Git as an example: *"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 user space tasks."*

Factor 4: First Impressions

> First impressions usually establish inertia for or against an innovation. A clearly defined mission statement that explains the rationale and principles of the innovation that does not require specialized domain knowledge is likely to positively affect first impressions of outsiders.

A user's first impression may prove to be an important factor in the decision to keep using a tool or technique. One panelist recalled a project in which a DVCS-style package management approach was attempted. However, the combination of the size of the packages, the specific DVCS selected, and the infrastructure that was used for hosting the repository, resulted in a system that was too slow and 'extremely painful' to be used effectively, as he described: *"this negative initial experience has made me very reluctant to use that system again, even though many people describe it as 'much faster now.' "* Another panelist commented that this reaction was curious, considering the *"release early, release often"* spirit commonly found in free software [39]. While a negative first impression may make a significant ripple through the community, positive first impressions have far less impact since fresh enthusiasm about a new tool is usually taken with a grain of salt.

A clearly defined purpose or mission statement will positively impact the forming of a good first impression. One panelist explained that, *"often, the designers of tools tend to assume that all future users will have their knowledge, skills and wisdom, which is a fallacy."*

Factor 5: Elegance

> Elegance is a subjective reward, but community members expose common preferences, including technical excellence, perfectionism and aesthetics.

Working on something that pleases can help increase one's efficiency. The design, quality of implementation, and technical correctness of a tool or technique can be important factors to some users, but users also have personal preferences that cannot be easily qualified and which may lead to irrational behavior. One panelist illustrated this: *"One of the most significant factors for tool adoption for me is a perception of 'cleanliness.' "* Another panelist added: *"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."*

The strive for technical excellence in the project is common, and one panelist claimed that *"the quality of implementation might sometimes be an important factor to decide about adopting a tool or not."* Others agreed with this strive for perfection among contributors as a core cultural trait of the Debian Project, where contributors are not told what to do, do not work to deadlines, and simply want to properly maintain their packages. Perceptions of 'elegance' are inherently subjective, and impressions can turn into 'religious beliefs' and the defense of tools against all forms of criticism without any facts to back up claims. This is common in cases where tools are more or less equivalent in features (e.g., the Vi vs. Emacs text editors).

Factor 6: Resistance

> Initial resistance to new ideas can help to separate good ideas from bad ones. Resistance can be met with conversion instructions, support, and patience.

Resistance to change is a common negative factor to adoption behavior, but not without positive aspects. One panelist explained an inherent resistance to changing the status quo: *"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."*

Reasons for resistance include a general time scarcity, a preference to get 'actual work' done, a lack of understanding of the new proposed concepts, and a categorical unwillingness to depart from existing approaches. On the other hand, resistance can help to filter out the good from the bad ideas. The latter are unlikely to withstand resistance for longer periods of time, and consequently the project does not lose time with tools that will not survive, and avoids having to recover from problems caused by mistaken adoption. Inadequately preparing for or supporting a change can cause a loss of interest, which makes overcoming others' resistance difficult. One panelist described how the disorganized state of the wiki page tracking the discussion on machine-readable copyright files made the process so inaccessible that potential supporters

turned away. On the other hand, advocates who took care to maintain available information and actively managed the discussion had more success in having their proposals adopted in the project.

Finally, some changes might affect maintainers of large numbers of packages more than the majority of the project members. They might raise resistance in order to defend themselves against an increased workload due to a proposed change. Related to this is the case where an improvement over previous approaches may not result in greater efficiency for the individual, but only at the project level. One panelist explained: *"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."*

Potential adopters will weigh adoption cost against a tool/technique's benefit. Debian contributors will without a doubt consider tools or techniques that automate manual labor and reduce maintenance costs. However, they are also aware of the costs of adopting a new tool, as one panelist explained: *"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.' "* The cost-benefit trade-off is influenced by a number of factors, such as the time investment needed, pragmatism ('good enough'), and 'doing the right thing,' that is, finding the right tool for a given problem, even if this takes more time than a manual approach.

Factor 7: Sustainability

> Confidence in the development direction and future of a tool or technique makes it a sustainable choice. Maintainers should incorporate feedback and enable users to influence the direction of development.

User faith in the development direction and future sustainability of a tool or technique was also found to be a factor. Developers want a certain level of confidence that a tool develops in the 'right' direction, and that its maintainers incorporate feedback, allowing users to influence that direction. A lack of such confidence increases the risk of a waste of invested time. Debian contributors tend to seek tools and techniques that will not disappear or become neglected. While predicting which tools will 'survive' is impossible, two key considerations are how well a tool is maintained, and the community that has formed around it to maintain the tool. As one panelist explained: *"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."*

Maintainers of tools can play an active role in addressing potential adopters' needs and perceptions of sustainability. One panelist recalled his analysis of the entire source archive to identify the number of packages using debhelper (the tool he was developing) versus other tools. He explained that, *"Improving market share was mostly a matter of figuring out why people were not using it and adding the features they needed, and responding to bugs and feature requests quickly."*

Factor 8: Quality Documentation and Examples

> Well-maintained documentation and clear examples are needed for widespread adoption. Early adopters seek background information including rationale for the innovation; later adopters seek tutorials. Examples provide practical starting points.

The availability of quality documentation and examples was also found to play a significant role in an adoption/rejection decision. Documentation is a necessity, especially when a tool/technique diverges far from the current processes, as one participant explained: *"The importance of documentation is directly proportional to the amount of divergence from similar tools or existing workflow patterns."* Mailing list archives and source code are not sufficient for widespread adoption—documentation needs to be maintained and must cater for different types of users; early adopters want background information and care about motivation, while late adopters tend to seek tutorial-style documentation. High quality documentation is also a sign of the maturity and stability of a tool, as one participant explained: *"Documenting ideas can be seen as a sign that they are serious and get stable."*

Factor 9: Trialability and Scalability

> New tools and techniques are evaluated in the context of individuals' own use-cases to determine their worth. Trying out a new tool should be as easy as possible, since that is one of the best ways to form an impression.

The importance of the ease with which a new tool or technique can be tried was succinctly illustrated by one panelist: *"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?"* Tools that require complex configuration or infrastructure to be set up in order to run have lower trialability; one panelist gave a comparative example: *"lintian [a package checker] has 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."* Directly related to trialability is a tool's scalability. The idea of scaled use is that one can put a tool or technique to use with ease for basic tasks, and still continue using the same approach as the complexity in usage scenarios increases. Some tools may be easy to use due to simplifications that would inhibit more complex use-cases.

Over time, people will have established workflows, and adapted tools to fit those. They may want to improve and evolve those workflows, rather than replace or revolutionize them. Volunteer free time is limited and fragmented (e.g., a few hours per day), and adopting new tools or techniques often requires significant chunks of time, and thus presents a potential inhibitor to adopting such new tools or techniques. One participant illustrated this as follows: *"Large monolithic changes to processes tend to take a lot of time, require a lot of debugging and can be disruptive to a general goal of getting things done. [...] It's 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."* In other words, the level to which a tool facilitates a gradual or evolutionary adoption may be more appealing than one that would cause a disruption (revolution) to the existing workflow.

Factor 10: Compatibility and Genericity

> Compatibility means that less time will be required for a new tool. The ability to reuse tools in other contexts will also positively affect adoption. There is a delicate balance between flexibility and usability.

Learning a new tool requires a time investment from developers, and with limited time available to work on a project, they will be very selective regarding how they spend their time. Tools that can be easily learned, or which automate tasks that you are already doing can be readily adopted; tools that build on known concepts or are 'finger-compatible' are easier to learn—this was thought to have played a role in Subversion's adoption rate (replacing CVS). One panelist emphasized that adopting a new tool will have a negative, temporary impact on productivity: *"Developers will build up their own arsenal of tools and their accompanying work-*

flows, and changing [tools] will cost productivity, so it is important that the impact of the switch be limited." Another panelist agreed, arguing that if a tool is too distinctive that it distorts normal workflow patterns or requires adjustments to long-established patterns, the perceived 'quality' of the tool will be diminished.

Compatibility is important on the conceptual level as well. One panelist criticized Git for its vocabulary that is incompatible with the terminology used in existing version control systems (VCS), rather than using compatible language shared with existing VCSs.

Related to compatibility is *genericity*, which refers to the preference for tools that are usable in different contexts. Being able to streamline work by reusing the same (or similar) tool/technique in different scenarios can play a decisive role in an adopt/reject decision. Once adopted, generic solutions tend to be reused, as one participant explained: *"People usually have their favorite packaging helpers, patch systems, etc., and when creating a new package will often reach for the last similar one."* Another panelist also argued that, *"reusability of tools is a major factor in their adoption in Debian, often very much at the expense of elegance."*

Factor 11: Modularity and Transparency

A very fine-grained solution may require code duplication due to the need to repeat similar sequences of instructions, but has the advantage that it offers more transparency and understanding of the various steps. Monolithic solutions on the other hand are less flexible and transparent. Such a higher level of abstraction leads to loss of control and makes a tool more difficult to understand.

Different tasks and communities need different levels of abstraction. We illustrate this point using an example of two widespread build utilities: debhelper and CDBS. Debhelper is a collection of scripts in the Unix spirit, each with a well-defined task and a consistent interface [39]. CDBS, which uses debhelper internally, presents a more abstract interface to the user, and exposes a large number of options to configure the build process. There was little agreement among developers about which was better as both tools have benefits and drawbacks. Participants emphasized a number of benefits in an abstraction layer such as CDBS—less code duplication, for example. Another benefit was that CDBS encourages maintainers only to specify the ways in which their packages deviate from the default behavior. Panelists responsible for large numbers of packages seemed in favor of higher levels of abstraction. However, others argued that, *"sometimes it's better and clearer to explicitly have 'repeated' code."* Also, one participant thought defects within CDBS were more difficult to fix, and claimed that, *"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."* Striving for the 'right' level of abstraction involves a compromise between individual flexibility and regularity across the project.

Factor 12: Maturity

Tools must exhibit a sufficient level of maturity, i.e., they must provide a reliable base before people will trust and depend on them. This implies that it should not change in ways that would require users to re-learn or change their scripts.

Tools or techniques must provide a reliable base before people will depend on them. A tool/technique must be usable to attract and sustain followers; it should not change continuously and require users to relearn or change their scripts, as one panelist explained: *"Implementation maturity is important, i.e., stable interfaces and relative freedom from serious bugs over time."* While the maturity of a tool is important, this does not suggest that tools should only be introduced when the software is 'finished,' as one participant explained: *"Release early as long as the software gets a simple, yet meaningful, job done: that all gives people a reason to start using it, and hopefully contribute to its growth."*

The importance of maturity depends on the 'reach' of impact that a tool has within the project. For changes such as those to the Debian source package format (which would have a very far-reaching impact), *"you need to consider every single aspect."*

Tools that seek to replace existing approaches will be scrutinized and compared to the current ones, and any problems will become barriers to adoption, as people can just stick with the status quo. On the other hand, new approaches that fill niches, or solve problems that were hitherto not addressed or unknown, instill lower expectations and requirements. The tolerance to new problems decreases as tools age, but project members also seem to build up a group tolerance to existing problems over time, which become 'well known' and get documented. Expectations of tools proposed to address such problems seem to grow, and tools that are not complete solutions are easily rejected.

Factor 13: Network Effects

Working in teams affects others in their choice of tools. People tend to be conservative in adopting new tools, which requires high quality alternatives. Changing tools or workflows affects everyone in a team, and therefore the larger the team, the more the inertia.

Network effects have become increasingly noticeable within the Debian project, as the project has shifted more towards team collaboration to alleviate the bottlenecks due to the voluntary nature of package maintainers and increasing package count. It is crucial to make changes harmoniously with collaborators, and as such, network effects may slow down adoption of new tools, but also help to ensure that the quality of tools that are in use remains high. The need to collaborate can restrict individual developers in their choice of tools. A community member is free to choose any text editor, for example, but when it comes to build dependencies (such as a patch management system), the tool used has to be compatible with those used by others. This effect slows down adoption of new tools within teams, even if a new tool constitutes an improvement. Making the use of a certain tool mandatory could be possible under certain circumstances (e.g. a small team), but doing so may alienate other collaborators that may be unwilling to follow suit. One panelist elaborated: *"I think the reason for team conservatism 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."*

The choice of a tool may also be affected by accessibility factors; potentially, tools may be adopted which are not 'the best' but which are more accessible to contributors. A conservative adoption strategy as a result of group inertia towards adopting new tools may result in a selection of 'better' tools, as 'bad' ones may not be accepted by a team of collaborators.

Factor 14: Consensus

Achieving consensus is a necessary process in a volunteer-driven project. Decisions that are made without consulting the community can be considered cabalistic and their authority questioned.

Pioneering work is necessary, but concrete solutions need to follow from that. It is important to build consensus among experienced people. Too much discussion can, however, cause loss of focus and hinder change. Debian contributors cannot be forced to use a particular tool, although tools may be mandated through a process of standardization (discussed below). Contributors who use incompatible and non-standard tools will effectively be forced

to bear the cost of a migration in that case. The process of building consensus is important to prevent increased resistance (discussed above). There seems to be an expectation for a minimum level of discussion; if a decision is made without giving everyone a chance to participate in the lead-up discussion, it may be considered 'cabalistic' and a decision's authority may be questioned. Debian prides itself on its openness, and non-public discussions are frowned upon. However, private discussions (among a small group of people) may be useful to anticipate disagreement when preparing controversial proposals (i.e. a new tool or technique that will have a significant impact on the status quo). The ease with which consensus can be achieved depends on the level of controversy that a new tool may introduce; non-controversial changes may be adopted readily, but the amount of discussion increases for topics that will have broad implications for the project.

Factor 15: Standards and Uniformity

> Standards evolve from practice and should define interfaces, not processes or tools. Uniformity reduces complexity across the project, which enables progress as approaches are streamlined.

A number of sources of standards exist in the Debian project in both explicit and implicit form. The most important explicit sources are the Debian Policy and the Developer's Reference [3]. The Debian Policy is a binding document describing rules to which packages must abide to be included in the Debian System. The Developer's Reference is a collection of responsibilities and best practices for developers. Furthermore, there are several unwritten rules or best practices; for example, the use of certain outdated tools (yada, dbs) should be avoided. Many best practices have not been formalized, either because consensus (discussed above) has not been reached, or nobody has taken initiative towards that end. One panelist clarified that, *"Debian has a strong culture around the idea [that] '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."* Another panelist added that, *"dictating through the [Debian] policy is a very good way to make people mad at it."*

Standards and uniformity should be sought at the righ level, i.e., at the level of *interfaces*, rather than specific tools. This allows people to use different tools, while adhering to a standard interface. One panelist used the following example: *"Having all packages of a team in the same Subversion repository doesn't mean I must use the SVN tool myself. Yay for git-svn."* (Git-svn is a tool that allows bridging between SVN and Git repositories.) Another panelist argued: *"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."*

Consistency across the project as a whole can motivate change. A good reason for adoption of a tool can be the desire to re-align outlying factions, i.e., teams who are doing things differently. One panelist commented that, *"While diversity is good to let many techniques compete, there's a time when they have matured where that diversity hurts more than anything else."*

Uniformity may also be triggered by critical events, such as the 'OpenSSL debacle,' which refers to a Debian-only patch to the OpenSSL package causing its cryptographic key to be predictable [26]. While this incident cannot be reduced to a lack of uniformity, some argued that the problem could have been prevented if there had been a uniform, canonical resource to track divergence between Debian and upstream software. The discussion that followed resulted in improved guidelines for patch management as well as a project-wide patch tracker.

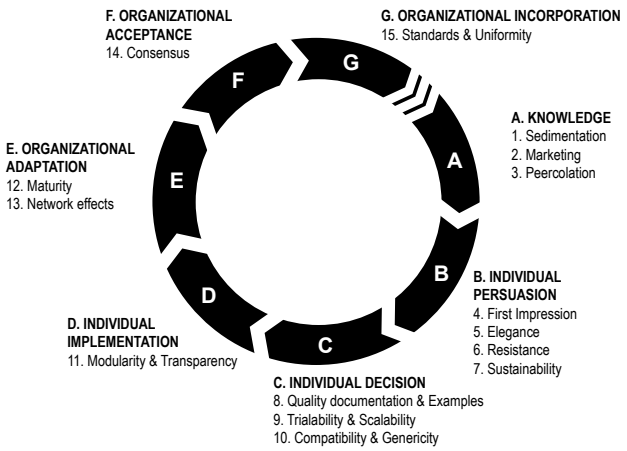

Figure 3. Multi-stage model for innovations in open source

5. DISCUSSION AND CONCLUSION

5.1 A Model for Innovation in FOSS Projects

The 15 factors that resulted from the Delphi study each play a role in the "cycle of innovation" in open source projects. Based on well known and widely used models of innovation adoption (e.g. [25, 41]), we suggest a multi-stage model for innovations in FOSS projects. The model is shown in Fig. 3 and consists of seven stages numbered A to G, which are discussed in more detail below. It is important to note that the factors' influence is not restricted to any particular stage—rather, each stage has a primary focus in which certain factors are more prevalent than others.

5.1.1 Stage A. Knowledge

Stage A, "Knowledge," refers to the idea that knowledge about new tools needs to spread before these tools can be adopted. Diffusion of knowledge happens through a number of channels, and several factors affect the spreading of knowledge. The first three factors, **sedimentation** (representing the time factor) **marketing** (pro-active advertising) and **'peercolation'** (opinions of respected peers) are key in this first stage.

5.1.2 Stage B. Individual Persuasion

In the second stage, potential individual adopters form an opinion about an innovation before they decide to adopt—this separation between persuasion and decision follows Rogers' innovation-decision process [41]. Several factors play a role in this stage. Making a good **first impression** is important, as well as an individual's opinion regarding a new tool's **elegance**. **Resistance** must be overcome, and finally an individual must be convinced of the new tool's **sustainability**. This stage is concerned with forming a favorable (or unfavorable) attitude.

5.1.3 Stage C. Individual Decision

If a potential individual adopter has formed a favorable opinion about a new tool to the extent that it has become a candidate for adoption, some practical factors come into play. Based on **quality documentation and examples**, an individual may start to gauge how the new tool can be used. The ease with which the tool can be tried out (**trialability**) and scaled up (**scalability**) will affect this decision. Furthermore, an important consideration in the decision stage is also whether or not the tool is compatible with the current workflow, and whether or not the tool is sufficiently **generic** that it can be used in other contexts—i.e. the time investment made for converting may be well worth it.

5.1.4 Stage D. Individual Implementation

Once a decision is made, an individual adopter may start implementation. It is important to note that this stage may still be aborted—implementation here does not imply successful adoption, but merely that efforts are made to start using the tool in practice. Of particular importance in this stage are the **modularity and transparency** of the tool, as these directly affect an adopter's understanding of the level of precision that can be achieved to automate the task that the tool aims to enable.

5.1.5 Stage E. Organizational Adaptation

Organizational adaptation is the next stage. In the case of a FOSS project the organization should be interpreted as the community. This stage starts after a considerable number of individuals have adopted an innovation. Knowledge of the innovation will have spread through the community and individual adoptions will converge, and the community will extend, re-invent or combine (*adapt*) innovations in a shape that works best for the community as a whole. In this stage, the **maturity** of an innovation becomes important—individuals may have different thresholds for accepting flaws, but for a community to accept an innovation it must exhibit a sufficient level of maturity. Also, in this stage **network effects** come into play, as a successful adoption of an innovation depends on community-wide acceptance.

5.1.6 Stage F. Organizational Acceptance

After organizational adaptation, the next stage is organizational acceptance. Once **consensus** has been achieved regarding the use of an innovation, this stage is completed. However, in the context of innovation adoption theory, acceptance is not the conclusive stage; it merely confirms that organizational members (or in our case, community members) are induced to commit to the innovation's usage. A further stage is necessary.

5.1.7 Stage G. Organizational Incorporation

Kwon and Zmud [25] stated that *"the innovation becomes embedded within an organization's routine and when the innovation is being applied to its full potential within an organization."* Thus, Incorporation is achieved when routinization occurs, that is usage of the technology application is encouraged as a normal activity, and also when infusion has been reached—increased organizational effectiveness is obtained by using the innovation. Explicit routinization happens through defining a policy or best practice in a standards document, but *de facto* standards are often sufficient to be considered routine without such definition. The stage organizational incorporation is considered to be achieved when an innovation has been promoted as a **standard**. **Uniformity** is a final factor at play in the decision to adopt a certain tool or technique. Uniformity reduces complexity and increases consistency across the project.

5.2 Threats to Validity

Several researchers have argued that trustworthiness is a more appropriate way to judge the validity of qualitative research such as this study. We adopt Guba's criteria [17] to evaluate naturalistic inquiries to differentiate them from quantitative studies which typically consider validity types such as internal and external validity. These criteria are credibility, transferability, dependability, and confirmability.

Credibility. We believe the identified factors are all plausible, and our confidence is strengthened by the fact that all factors were identified through a longitudinal process of several months involving 21 experts. This means that the factors have been discussed at great length; none of the panelists indicated that any of the factors should not be included. Furthermore, the Delphi study included a specific phase in which the expert panel was asked for specific instances, thus bringing the factors to life. Thus, we believe the Delphi process itself, having taken several iterations, has contributed to the credibility of the findings.

Transferability. This study focused specifically on the Debian Project, one of the largest FOSS projects comprising tens of thousands of packages. Some of the factors might be of less importance in smaller projects. Most FOSS projects are significantly smaller, even when excluding those projects with only a single contributor. Nevertheless, even smaller projects should consider network effects and consensus, and technical considerations such as elegance are always desirable characteristics. We observe that none of the factors are tied specifically to the Debian Project, and as such we believe these factors can apply to all volunteer-driven projects. We argue that those projects with significant company involvement, and thus with stakeholders that have significant influence to 'push' changes, can also benefit from being cognizant of these factors.

Dependability. In our study, the Delphi panel consisted of 21 carefully selected participants through a stratified purposeful sampling strategy. We identified panelists across a number of 'dimensions' so as to include people with a wide variety of insights and opinions, as is desirable for Delphi studies. Furthermore, the research process itself is completely recorded, thus establishing an *audit trail* of intermediate research artifacts. This facilitates full traceability of findings back to the original input from panelists.

Confirmability. In selecting the panelists, we took great care in selecting members with whom we had no prior interaction, which was of particular importance given the lead author's role within the Debian community. Another tactic is that of *member checking*, which is inherently built into the multi-phased Delphi process. As insights and opinions were recorded, they were analyzed, rephrased and summarized and presented back to the panelists.

5.3 Conclusion

We observed a tension between, on the one hand, the availability of efficient tools and techniques that could help large projects such as Debian scale better, and the slow adoption of these tools and techniques on the other hand. The underpinning challenge lies in the voluntary nature of FOSS projects and the lack of authoritarian decision-making structures to enforce those changes.

This study investigated which factors influence the Debian package maintainers' decision to adopt new tools or techniques. Using a policy Delphi study conducted over the course of several months involving a panel of 21 carefully selected participants, we distilled 15 factors that affect the decision to use tools and techniques in an FOSS context. These were subsequently organized in a seven-stage model for innovation in open source projects.

The contribution of this paper is twofold. The first contribution is insight into the various factors that affect decisions to adopt novel tools and techniques by FOSS developers in the Debian project. While there have been several studies of the Debian project, to the best of our knowledge this is the first study investigating the adoption of tools and techniques used in the Debian project specifically, and in FOSS projects more generally. As pointed out in Sec. 1, very few studies have addressed this issue.

The second contribution is methodological, through its demonstration of the viability and use of the policy Delphi method to study a contemporary phenomenon in software engineering research in general, and FOSS in particular. The Delphi method has seen very little use in the software engineering discipline, but it offers a very

rigorous approach to conducting field research which has built-in mechanisms such as member checking which help to assess the validity of the findings. Very little research has focused on adoption and diffusion *within* FOSS communities (as opposed to research on adoption of FOSS products by end-users and organizations). Therefore, we believe this qualitative study focusing on a FOSS project contributes an alternative approach to this area.

6. ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 to Lero; the Irish Research Council New Foundations 2014; and Enterprise Ireland IR/2013/0021 to SCALARE.

7. REFERENCES

[1] Amor-Iglesias, J.-J., Gonzalez-Barahona, J.M., Robles, G. and Herraiz, I. 2005. Measuring libre software using Debian 3.1 (Sarge) as a case study. *UPGRADE*, 6, 3.

[2] 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.

[3] Barth, A., Carlo, A.D., Hertzog, R., Nussbaum, L., Schwarz, C. and Jackson, I. 2014. *Debian Developer's Ref. v3.4.14*.

[4] Bruckhaus, T., Madhavji, N.H., Janssen, I. and Henshaw, J. 1996. The Impact of Tools on Software Productivity. *IEEE Software*, 13, 5.

[5] Crowston, K., Wei, K., Howison, J. and Wiggins, A. 2012. Free/Libre Open Source Software Development: What We Know and What We Do Not Know. *ACM Comput. Surv.*, 44.

[6] Dalkey, N. and Helmer, O. 1963. An experimental application of the Delphi method to the use of experts. *Management Science*, 9, 3.

[7] Dalkey, N.C. 1972. *The Delphi method: an experimental study of group opinion*. Lexington Books.

[8] 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.

[9] Debian Community. 2004. *Debian Social Contract v. 1.1*.

[10] Dishaw, M.T. and Strong, D.M. 1998. Supporting software maintenance with software engineering tools: A Computed task-technology fit analysis. *J. Sys. Soft.*, 44, 2.

[11] Dunn, W.N. 1994. *Public Policy Analysis: An Introduction*. Prentice Hall, Englewood Cliffs, NJ.

[12] Eckhardt, A., Laumer, S. and Weitzel, T. 2009. Who influences whom? Analyzing workplace referents' social influence on IT adoption and non-adoption. *J Inf Tech 24, 1*

[13] Fitzgerald, B. 2006. The Transformation of Open Source Software. *MIS Quart.*, 30, 3.

[14] Fitzgerald, B., Kesan, J.P., Russo, B., Shaikh, M. and Succi, G. 2011. *Adopting Open Source Software.* MIT Press.

[15] Fraser, S., Cooper, K., Coplien, J., Lennon, R., Ravichandar, R., Spinellis, D. and Succi, G. 2012. Software Tools Research: A Matter of Scale and Scope - or Commoditization? In *Proc. 3rd Conf. SPLASH*. ACM.

[16] Goodhue, D.L. and Thompson, R.L. 1995. Task-technology fit and individual performance. *MIS Quart.*, 19, 2.

[17] Guba, E.G. 1981. Criteria for assessing the trustworthiness of naturalistic inquiries. *Edu. Commun. Technol.*, 29, 2.

[18] Harrison, W., Ossher, H. and Tarr, P. 2000. Software Engineering Tools and Environments: A Roadmap. In *Proc. Future of Software Engineering*.

[19] Hauge, Ø., Ayala, C. and Conradi, R. 2010. Adoption of open source software in software-intensive organizations-A systematic literature review. *Inf. Softw. Technol.*, 52, 11.

[20] Iivari, J. 1996. Why are CASE Tools not used? *CACM*, 39.

[21] Judd, R.C. 1972. Use of Delphi methods in higher education. *Technological Forecasting and Social Change*, 4, 2.

[22] Keeney, S., Hasson, F. and McKenna, H. 2011. *The Delphi Technique in Nursing and Health Research*. Wiley.

[23] Krafft, M.F. 2006. *The Debian System: Concepts and Techniques*. No Starch Press.

[24] Krafft, M.F. 2010. *A Delphi study of the influences on innovation adoption and process evolution in a large open-source project: The case of Debian*. University of Limerick.

[25] Kwon, T.H. and Zmud, R.W. 1987. *Unifying the fragmented models of information systems implementation*. John Wiley.

[26] Laurie, B. 2008. Debian and OpenSSL: The Aftermath. `http://www.links.org/?p=328.`

[27] Lee, Y., Kozar, K.A. and Larsen, K.R. 2003. The Technology Acceptance Model: Past, Present, and Future. *Comm. Assoc. Inform. Syst.*, 12.

[28] Linstone, H.A. and Turoff, M. 2002. *The Delphi Method: Techniques and Applications*. Information Systems Department, New Jersey Institute of Technology.

[29] Loo, R. 2002. The Delphi method: a powerful tool for strategic management. *Policing: An International Journal of Police Strategies & Management*, 25, 4.

[30] Mosley, V. 1992. How to Assess Tools Efficiently and Quantitatively. *IEEE Softw.*, 9, 3.

[31] Novak, J.D. and Cañas, A.J. 2006. *The theory underlying concept maps and how to construct and use them*. Florida Institute for Human and Machine Cognition.

[32] O'Sullivan, B. 2009. Making Sense of Revision-Control Systems. *Commun. ACM*, 52, 9.

[33] Okoli, C. and Pawlowski, S.D. 2004. The Delphi method as a research tool: an example, design considerations and applications. *Information & Management*, 42, 1.

[34] Paraskevas, A. and Saunders, M.N.K. 2012. Beyond consensus: an alternative use of Delphi enquiry in hospitality research. *Int'l J. Contemporary Hospitality Manage*, 24, 6.

[35] Patton, M.Q. 2001. *Qualitative Evaluation and Research Methods*. Sage Publications, Newbury Park, CA, USA.

[36] Pfleeger, S.L. and Menezes, W. 2000. Marketing Technology to Software Practitioners. *IEEE Softw.*, 17, 1.

[37] Prechelt, L. and Oezbek, C. 2011. The search for a research method for studying OSS process innovation. *Empir Software Eng*, 16, 4.

[38] Raghavan, S.A. and Chand, D.R. 1989. Diffusing software-engineering methods. *IEEE Software*, 6, 4.

[39] Raymond, E.S. 2001. *The Cathedral & the Bazaar*. O'Reilly.

[40] Redwine, S.T. and Riddle, W.E. 1985. Software technology maturation. In *Proc. ICSE*.

[41] Rogers, E.M. 2003. *Diffusion of Innovations*. Free Press.

[42] Rossi, B., Russo, B. and Succi, G. 2012. Adoption of free/libre open source software in public organizations: factors of impact. *Information Technology & People*, 25, 2.

[43] Rugg, G. and McGeorge, P. 2005. The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, 22, 3.

[44] Schmidt, R.C. 1997. Managing Delphi Surveys Using Nonparametric Statistical Techniques. *Decision Sciences*, 28.

[45] Shaikh, M. and Cornford, T. 2003. Version Management Tools: CVS to BK in the Linux Kernel. In *Proc. 3rd Workshop Open Source Softw. Eng.*

[46] Spinellis, D. 2012. Git. *IEEE Softw.*, 29, 3.

[47] Turoff, M. 1970. The Design of a Policy Delphi. *Technological Forecasting & Social Change*, 2, 2.

[48] Wicks, M.N. and Dewar, R.G. 2007. A new research agenda for tool integration. *J. Syst. Softw.*, 80, 9.