# Software Process Maturity and the Success of Free Software Projects

Martin MICHLMAYR
*Department of Computer Science and Software Engineering*
*University of Melbourne*
*Victoria, 3010, Australia*
*e-mail: martin@michlmayr.org* [1]

**Abstract.** The success of free software and open source projects has increased interest in utilizing the open source model for mature software development. However, the ad hoc nature of open source development may result in poor quality software or failures for a number of volunteer projects. In this paper, projects from SourceForge are assessed to test the hypothesis that there is a relationship between process maturity and the success of free software and open source projects. This study addresses the question of whether the maturity of particular software processes differs in successful and unsuccessful projects. Processes are identified that are key factors in successful free software projects. The insights gained from this study can be applied as to improve the software process used by free software projects.

**Keywords.** Process maturity, quality improvement, free software, open source

## Introduction

The development model employed in free software and open source projects is unique in many respects. Free software and open source projects are traditionally characterized by two factors that have an impact on quality and project management [8], [7]. The first factor is that they are performed by volunteers, the second that the participants are globally distributed. In addition to these two important characteristics, the development model is also crucially determined by the nature of free software and open source itself; all source code is available and shared with others. Raymond suggests that this open exchange leads to a high level of peer review and user involvement [11], [13]. It has been shown in software engineering research that peer review contributes to the quality of software [1]. This may offer an explanation for the high levels of quality found in some successful free software and open source projects, such as Linux and Apache.

While there has been an increase in research carried out on free software and open source, only a small amount of attention has been given to investigating which traditional software engineering insights can be mapped to the open source development model [6]. Since volunteers working on free software and open source are often driven by different motivations to those of software companies developing proprietary software [3], it is not clear whether all strategies postulated in software

[1] Present address: Centre for Technology Management, University of Cambridge, UK

engineering can be applied to open source projects. However, this issue is of high relevance if techniques are to be found that further improve the success of the open source development model.

This paper investigates whether the maturity of the processes employed by distributed, volunteer projects is connected to their success. While most software engineering research on open source has concentrated on successful projects (for example Apache [9], Debian [2], [14], GNOME [5] and Linux [15]), the observation can be made that there is a large number of unsuccessful open source projects that have not been studied in detail to date [4]. Our hypothesis is that there is a link between process maturity and the success of a project, especially where processes related to coordination and communication are concerned. During the investigation of this hypothesis, key areas of the software process linked to the success of free software and open source projects are identified. These factors give an indication of the importance of various processes and will contribute to the creation of more successful free software and open source projects.

## 1. Methodology

This paper is based on an empirical case study involving 80 projects hosted on SourceForge. This is currently the largest hosting site for free software and open source projects with over 95,000 projects and more than one million registered developers. The maturity of these randomly chosen projects were evaluated using a simple assessment designed for this purpose. One central goal during the development of this assessment was to create a general mechanism for evaluating process maturity rather than one specific to the present research question. The assessment form can therefore be used by other researchers and practitioners to judge the maturity of any open source project; it is included in Appendix A. Due to the generalizable nature of the assessment it is described before the actual case study.

### *1.1. Project Assessment*

An assessment has been developed for this study that can be used to evaluate some crucial aspects of the maturity of free software and open source projects. In particular, the assessment focuses on the maturity of processes related to coordination and communication since these are important components in distributed projects [7]. The assessment makes use of empirical data typically available in open source projects. The validity of the assessment is assumed to be high since most questions can be answered objectively and only some require personal judgement. While neither the inter-rater nor the re-test reliability have been statistically verified, both showed reliable results in this case study.

The assessment incorporates insights from both software engineering and open source. The assessment is based on quality models which postulate the importance of high quality and disciplined processes. In particular, emphasis was put on processes which might be important in open source projects. Specifically, the distributed and volunteer nature of free software and open source projects was taken into account. Since free software and open source benefit from a strong community surrounding a project [11], [16], processes to attract volunteers and integrate them into the project are vital.

The assessment is grouped into five categories taking different aspects of free software projects into account. The majority of questions can be addressed with a simple "yes" or "no" answer since they concern the presence or availability of a process or piece of infrastructure. For some questions, a finer grained measure is beneficial and zero, one or two points are given for these questions (more points indicating higher maturity). The whole assessment with a description of the questions is available in Appendix A

### *1.1.1. Version Control*

The first question relates to version control and configuration management in the project. The use of version control tools, such as CVS [18], allows multiple volunteers to work on the same code with little coordination. Furthermore, the wide availability of the code repository encourages volunteers to review code, remove defects or add new functionality. Therefore, this question distinguishes whether a version control system is used at all, as well as whether it is publicly available or not.

### *1.1.2. Mailing Lists*

Mailing lists are also important in free software projects because they create an effective and fast medium for communication and coordination. Furthermore, mailing lists are often used to answer questions posed by users and developers and hence encourage more people to get involved and contribute to the project. This question differentiates between the presence of no mailing list at all, one mailing list, or multiple specialized lists. Dedicated mailing lists for different topics help to keep the focus. There can be lists for users, developers and announcements. There is also a question about the availability of mailing list archives. Mailing list archives give users and developers the opportunity to follow previous discussions and to find answers which have previously been given. This reduces the time spent by developers answering questions that have already been addressed, and it lowers the barrier of entry for prospective developers.

### *1.1.3. Documentation*

The third category is about documentation, both for users and developers. There might be an important link between user documentation and the success of a project since it is possible that users will not be able to use the software if it is not well documented. Dedicated developer documentation might encourage new volunteers to contribute by reducing the learning curve associated with getting involved in a project. It could also increase the maintainability and quality of the source code if everyone adheres to the same coding standards outlined in the developer documentation.

### *1.1.4. Systematic Testing*

The next three questions relate to systematic testing and the availability of important infrastructure and processes. The first question asks whether the project publishes release candidates before issuing stable releases. Release candidates are an indication that a project has a clear release plan. The next question concerns the availability of an automatic test suite comprising regression or unit tests, and the third one checks the presence of an defect or issue tracking system. Such a system might encourage users to report defects and hence enables developers to remove them and improve the quality of

the software. It may also be a good mechanism to capture feedback and ensure that valuable input is not lost.

#### *1.1.5. Portability*

Finally, the last question is about the portability of the software. Software that can be ported to a wide variety of platforms attracts more users who might contribute to the project. Additionally, porting software to a new platform often highlights defects which only occur under special circumstances. This question differentiates between three levels of portability. A piece of software might be aimed specifically at one platform, can support a set of hard-coded platforms, or supports multiple platforms through a system which automatically identifies the characteristics of a system.

### *1.2. Case Study*

The previously described assessment was employed to evaluate the maturity of 80 projects from SourceForge. SourceForge was chosen because it is a very popular hosting site for free software and open source projects. At the time of writing, SourceForge hosts more than 95,000 projects and has over one million registered developers. It provides various types of infrastructure for projects, such as CVS, mailing lists and issue tracking systems. While this hosting site was chosen for the data collection, the findings of this research are intended to also apply to open source projects not hosted on SourceForge. Since the assessment form used for the case study are described in detail, the findings here can be replicated by other researchers.

Since it is the aim of this study to investigate whether the maturity of the employed processes is linked to the success of a project, 40 successful and 40 unsuccessful projects were randomly selected for this investigation. SourceForge provides elaborate statistics about projects and their download data was taken to define success. While downloads do not equate or necessarily imply quality or even success [4], it is a fairly good measure of fitness for purpose because users of open source software must actively download this software rather than use pre-installed applications on their computer systems. Additionally, the advantage of using downloads as a measure is that it is objective and independent – the users determine this measure rather than the researcher.

As a first step, 40 successful projects were selected from one of SourceForge's statistics page which lists the 100 top downloads. Since comparability is a vital criterion in this case study, projects were excluded which do not create a piece of software but rather assemble a large collection of software created at other places. Additionally, projects for which no source code was available were not taken into account since this would not allow a complete assessment. Following these criteria, 33 projects out of the top 100 were excluded. From those remaining, 40 projects were chosen randomly.

In order to select unsuccessful projects, the statistics on downloads was employed. Specifically, the percentile value for a given project from September 2003 was taken because this indicates how a project is placed in relation to others. Our randomly selected successful projects had a mean percentile of 97.66% ($\sigma$ = 2.78). Taking this figure into account, an upper limit of 75% was arbitrarily chosen as the criterion of an unsuccessful project as this is considerable less than the average placement of successful projects. During the selection of unsuccessful projects it became evident that

they were on average much smaller in size than the successful ones. While this observation is interesting and should be pursued in greater detail in the future, it is problematic for this case study since the two groups had to be of similar characteristics and *only* differ in their success. If the groups differed in other ways it could not be clearly argued that the findings of this study are related only to project success. Hence, unsuccessful projects were selected randomly until the group of unsuccessful projects had similar characteristics as the one consisting of successful projects. In particular, the factors of age, lines of code (LOC) and development status were taken into account (see Table 1). Development status is based on SourceForge's categorization of the project; lines of code of a project was determined with the popular program SLOCCount by David A. Wheeler [20], [2]. At the end, the groups were adequately comparable, with neither age ($t(78) = 1.466$; $p = 0.147$), lines of code ($t(72) = 1.47$; $p = 0.146$) or development status ($W = 921$; $p = 0.089$) showing a significant difference. However, the groups highly differed in their success ($t(40) = 16.04$; $p < 0.001$).

**Table 1.** Means and standard deviation of successful and unsuccessful projects

| | | Age | LOC | Status | Ranking |
|---|---|---|---|---|---|
| Mean | successful | 1163 | 126500 | 4.43 | 97.66% |
| | unsuccessful | 1066 | 78140 | 4.11 | 41.18% |
| Standard deviation | successful | 298 | 167995 | 1.06 | 2.78% |
| | unsuccessful | 291 | 123109 | 0.98 | 22.09% |

All 80 projects were assessed by two graduate students, each focusing on 40 random projects. The assessment was performed on a blind basis, i.e. the students were not aware whether a project was deemed successful or unsuccessful. While they were graduate students of computer science with a solid background in the general field, they had little experience with free software and open source projects, so reducing the likelihood that their assessment would be biased by the recognition of successful and popular open source projects. Since the assessment has been designed in a way to ensure high validity even if little familiarity with open source is present, the selection of these assessors strengthens the methodological foundations of this study.

The assessment was carried out in the first week of October 2003. After the data gathering and assessment was completed and verified, various statistical tests were performed. The chi-square test was used for nominal data while the Wilcoxon test was employed for ordinal data. All tests were performed with the statistics package GNU R, version 1.8.0, and were carried out with $\alpha = 0.05$ (the probability of rejecting the statistical hypothesis tested when, in fact, that hypothesis is true). The null hypothesis for all tests was that there is no difference between successful and unsuccessful projects, and all tests were two-sided.

## 2. Results and Findings

The data for each question of the assessment form was analyzed, and the results will be presented individually in the following section, followed by a discussion and a prospective of further research.

*2.1. Results*

In total, nine statistical tests were carried out, corresponding to the questions which comprise the assessment.[2] As expected, given the number of tests, some tests revealed significant differences in the two groups of successful and unsuccessful projects while others did not. Version control is one aspect where the groups differ significantly ($W$ = 1046.5, $p$ = 0.004). While the successful group shows 1.75 points on average ($\sigma$ = 0.59) out of 2, the unsuccessful projects only score 1.275 points ($\sigma$ = 0.85). Similarly, the availability of mailing lists differ between the two groups ($W$ = 1113.5, $p < 0.001$). Not only do successful projects make better use of mailing lists, gaining 1.55 points on average ($\sigma$ = 0.78) out of 2 as compared to 0.925 points ($\sigma$ = 0.86) of unsuccessful projects, they also provide archives of their discussions in more cases. A total of 80% of successful projects offer mailing archives while only half of the unsuccessful projects do so.

**Table 2.** Availability of documentation

| | | **User** | **Developer** |
|---|---|---|---|
| Successful | available | 23 (57.5%) | 10 (25.0%) |
| | not available | 17 (42.5%) | 30 (75.0%) |
| Unsuccessful | available | 24 (60.0%) | 5 (12.5%) |
| | not available | 16 (40.0%) | 35 (87.5%) |

The third category of the assessment is related to documentation in the project (Table 2). The groups do not differ significantly either in the availability of user documentation ($\chi^2$ = 0; $p$ = 1) or of documentation aimed at developers ($\chi^2$ = 1.31; $p$ = 0.25). While about 60% of projects of projects offer user documentation, only 25% of successful and 12.5% of unsuccessful projects offer developer documentation. The fourth category, testing, reveals mixed results. Significantly more successful projects offer release candidates ($\chi^2$ = 5.16; $p$ = 0.023), and make use of a defect tracking system ($\chi^2$ = 24.96; $p < 0.001$). However, the groups do not differ significantly in the use of regression or unit test suites ($\chi^2$ = 0.37; $p$ = 0.54). Table 3 shows exactly how much the different facilities are employed by the two groups. In summary, most successful projects use Source-Forge's defect tracking system, more than half of the successful projects provide release candidates but few projects use test suites. On the hand, only about 30% of unsuccessful projects use the defect tracking system and make release candidates available.

**Table 3.** Presence of testing facilities

| | | **Release Candidates** | **Automatic Test Suite** | **Defect Tracking** |
|---|---|---|---|---|
| Successful | used | 22 (55.0%) | 5 (12.5%) | 35 (87.5%) |
| | not used | 18 (45.0%) | 35 (87.5%) | 5 (12.5%) |
| Unsuccessful | used | 11 (27.5%) | 8 (20.0%) | 12 (30.0%) |
| | not used | 29 (72.5%) | 32 (80.0%) | 28 (70.0%) |

[2] The results are given in standard statistical notations. $W$ refers to the Wilcoxon test used for ordinal data whereas $\chi^2$ denotes the chi-square test used for nominal data. $p$ is the probability; if it is lower than $\alpha$, the null hypothesis is rejected.

Finally, there is no major difference between the groups with regards to portability ($W = 793$, $p = 0.95$). The two groups score almost equally, with successful projects gaining an average of 1.325 points ($\sigma = 0.76$) and unsuccessful projects scoring 1.375 points ($\sigma = 0.62$). These results will be discussed in more detail in the following section.

### *2.2. Discussion*

A statistical analysis of the data gained through the evaluation of 40 successful and 40 unsuccessful projects shows that 5 out of 9 tests show significant differences between the two groups. In those cases where major differences can be found, the successful projects have been found to employ a more mature process than unsuccessful project. Due to the high number of tests which show differences between the groups, it can be concluded that the maturity of the processes employed by a project is linked to its success.

This paper has therefore shown the existence of a relationship between the processes used by a project and its success. However, the present study did not investigate the nature of this relationship. In particular, no conclusions on a possible causality between process maturity and success can be drawn from the present study. Further in-depth work on the qualitative aspects of the relationship between process maturity and success have to be carried out.

Finally, although this paper clearly shows that the maturity of the processes play an important role, there may be other factors which have not been considered in the present study.

#### *2.2.1. Version Control*

Version control tools, such as CVS in the case of projects hosted on SourceForge, are used more widely in successful than in unsuccessful projects. Furthermore, a higher percentage of version control repositories are available publicly. In free software projects, CVS and similar tools play important roles related to coordination. Having a private version control repository limits the access of prospective contributors. On the other hand, a publicly available CVS repository allows contributors to monitor exactly what other people are working on. This allows multiple people to work together efficiently in a distributed way. In a similar way to defect tracking systems, public version control systems may attract more volunteers since users see what needs to be done and how they can get involved.

#### *2.2.2. Mailing Lists*

Similarly to version control tools, mailing lists have an important function for coordination in a project. Archived mailing lists also replace documentation to some degree since users can find answers to questions which have been asked before. Prospective volunteers can follow previous discussions about the development and can so easily learn how to take part in the project. In both, version control tools and mailing lists, it is not clear from the present findings whether a successful project requires these types of infrastructure to flourish, or whether the implementation of the infrastructure has attracted more volunteers and so led to more success. Our assumption is that there is no clear causality and that both affect each other. However, it is clear that mailing lists and version control tools are key factors in successful open source projects.

### *2.2.3. Documentation*

The presence of user and development documentation is not a good indication for the success of a project. While more than half of the projects in this case study offer documentation for users, a much smaller number provides developer documentation. This difference suggests that the findings must be interpreted differently for user and developer documentation. Since a larger number of projects provide documentation that do not, it can be argued that user documentation is an important factor in free software projects. However, this factor alone does not determine success. Developer documentation, on the other hand, is not very common in free software projects. A reason for this might be that prospective developers find most answers to their questions from other sources, such as mailing list archives or the source code. They can derive the coding style used in the project directly from the source code if there is no guide about the coding style, and can follow the development process on the mailing lists. They can effectively learn the rules by observing them in action [19]. This suggests that written documentation is not necessary because it is available in indirect form already. Competent developers will be able to find all information from other sources and by directly observing the process. While this explanation seems very plausible, there is one remaining question. It would be interesting to study the code quality of different free software and open source projects to investigate whether the availability of documentation for developers leads to higher code quality. It is also possible that the availability of good developer documentation will attract more potential developers who will contribute to a project.

### *2.2.4. Systematic Testing*

The analysis of processes involving testing shows that successful projects make more use of release candidates and defect tracking systems than unsuccessful projects. There is no difference in the presence of automated test suites. The availability of release candidates has been taken as an indication of a well defined release plan. While it would be revealing to qualitatively study and compare the release strategies of different projects, the present findings suggest that a clear release strategy coupled with testing contributes to the success of a project. Furthermore, defect tracking systems play a crucial role. While the successful and unsuccessful groups in this case study do not differ significantly in their size, successful projects are likely to receive more feedback from users. This feedback has to be captured properly so it can later be prioritized and analyzed. Neither successful nor unsuccessful projects make much use of automated test suites. One reason for this may be the nature of open source itself. Raymond argues that open source projects benefit from a greater community surrounding them that provides defect reports [11]. Once a regression occurs it is very likely that one of the users will notice it quickly and report it. This might be a reason why little time is spent on implementing automated tests. However, it remains to be seen whether the use of test suites could lead to even higher quality. The use of automated tests would allow volunteer to spend less time on tracking down regressions and they could devote their time to the removal of other defects or the implementation of new features.

### *2.2.5. Portability*

This last assessed factor is very high in both groups. A likely reason for this is that the Unix philosophy has always promoted portability [12]. Connected to this is the

availability of software, such as autoconf [17], [10], which makes it relatively easy for a piece of software to automatically determine the characteristics of a platform and to adapt to it. Another factor which could contribute to portability is the diverse nature of volunteers participating in free software projects.

#### *2.2.6. Summary*

Several factors have been identified which are linked to the success of a project. Projects which have incorporated version control, effective communication mechanism such as mailing lists and various testing strategies into their software development process are more successful than other projects. These findings clearly support the hypothesis that free software and open source projects profit from the use of mature processes. While this study has focused on processes connected to coordination and communication, two factors that are crucial in distributed projects, the open source process might also benefit from other processes used in traditional software engineering projects. This finding is in contrast to the suggestion of some open source developers that their development model is unique and will not benefit from process maturity or traditional insights [6].

### *2.3. Further Work*

The present paper clearly shows that free software and open source projects benefit from a highly mature process. This study supports the notion that some software engineering insights can be applied to the open source development model. The results of this paper are of substantial value and demonstrate the importance of further research in this area to address some of the questions raised in this paper. While this study showed that developer documentation does not contribute to the success of a project, this paper brings forward the suggestion that solid developer documentation can lead to higher quality source code. A study which compares the code quality of projects with and without good developer documentation can shed more light on this question. Furthermore, while the use of automated test suites in free software projects is not currently common, they may offer certain benefits to the participants of a volunteer project. Another question related to testing is whether projects using defect tracking systems have a higher defect removal rate than projects using less reliable techniques to track defects. Finally, a qualitative approach comparing different release strategies could lead to specific guidelines being developed tat promote successful strategies for delivering tested and stable open source software for end-users.

## 3. Conclusions

This paper started with the observation that free software and open source projects are different to traditional software engineering projects in many respects. This is mainly because free software and open source projects are typically performed by volunteers who are distributed all over the world. This important difference to other software engineering projects, such as those deployed by companies, raises the question whether traditional insights can be applied to open source projects. It has been proposed that some fundamental insights from software quality models, such as the demand for mature processes, can be applied to open source projects. Some processes praised by

software engineering, such as the requirement for requirements specifications and other formal guides, could be incompatible with the motivation of volunteer participants. However, others can be unified with the open source development model and contribute to success and quality. In particular, processes connected to coordination and communication appear to be vital for free software projects.

In order to investigate this hypothesis, a case study has been conducted involving randomly chosen projects from SourceForge. In the case study, 40 successful and 40 unsuccessful projects were assessed regarding their maturity in several aspects using an assessment developed in this paper. The results of various statistical analyses clearly showed that the maturity of some processes are linked to the success of a project. This study identified the importance of the use of version control tools, effective communication through the deployment of mailing lists, and found several effective strategies related to testing. The identification of processes employed by successful free software projects is of substantial value to practitioners since they give an indication of which areas deserve attention.

The findings of this paper emphasize the importance of applying software engineering insights to the open source development model. Further research has been suggested to explore some of the results of this paper in more detail, such as a thorough comparison of code quality and a qualitative evaluation of release strategies. The identification of important processes, together with more research in this area, will contribute to higher quality and more successful free software and open source projects.

**Acknowledgements**

Thanks are given to Deepika Trehan and S Deepak Viswanathan, two graduate students at the University of Melbourne, for assessing the maturity of the 80 projects in this case study. Thanks are also extended to Mike Ciavarella and Paul Gruba for their continuous support throughout this case study and to the anonymous reviewers for their extensive comments. This work has been funded in part by Fotango.

## A. Assessment Questions

### *A.1. Version Control*

1. Does the project use version control?
   - No, the project does not use any version control (0 points)
   - Yes, it is used but the repository is private (1 point)
   - Yes, it is used and the repository available to the public (2 points)

### *A.2. Mailing Lists*

1. Are mailing lists available?
   - No, the project has no mailing list at all (0 points)
   - Yes, there is one mailing list (1 point)
   - Yes, there are multiple, specialized mailing lists (2 points)
2. Are mailing list archives available?

*A.3. Documentation*

1. Does the project have documentation for users?
2. Does the project have documentation for developers?

*A.4. Systematic Testing*

1. Are release candidates available?
2. Does the source code contain an automatic suite?
3. Does the project use a defect tracking system?

*A.5. Portability*

1. Is the software portable?
   - No, it has been written specifically for a single platform (0 points)
   - Yes, it supports a limited number of platforms (1 point)
   - Yes, it automatically determines the properties of a platform (2 points)

## References


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

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

[3] Alexander Hars and Shaosong Ou. Why is open source software viable? A study of intrinsic motivation, personal needs, and future returns. In *Proceedings of the Sixth Americas Conference on Information Systems (AMCIS 2000)*, pages 486–490, Long Beach, California, 2000.

[4] James Howison and Kevin Crowston. The perils and pitfalls of mining Source-Forge. In *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*, pages 7–11, Edinburgh, UK, 2004.

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

[6] Bart Massey. Why OSS folks think SE folks are clue-impaired. In *3rd Workshop on Open Source Software Engineering*, pages 91–97. ICSE, 2003.

[7] Martin Michlmayr. Managing volunteer activity in free software projects. In *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, pages 93–102, Boston, USA, 2004.

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

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

[10] Andy Oram and Mike Loukides. *Programming With GNU Software*. O'Reilly & Associates, 1996.

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

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

[13] Eric S. Raymond and W. Craig Trader. Linux and open-source success. *IEEE Software*, 16(1):85–89, 1999.

[14] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Martin Michlmayr. Evolution of volunteer participation in libre software projects: Evidence from Debian. In Marco Scotto and Giancarlo Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 100–107, Genova, Italy, 2005.

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

[16] Anthony Senyard and Martin Michlmayr. How to have a successful free software project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pages 84–91, Busan, Korea, 2004. IEEE Computer Society.
[17] Gary V. Vaughan, Ben Elliston, Tom Tromey, and Ian Lance Taylor. *GNU Autoconf, Automake, and Libtool*. SAMS, 2000.
[18] Jennifer Vesperman. *Essential CVS*. O'Reilly & Associates, 2003.
[19] Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.
[20] David A. Wheeler. More than a gigabuck: Estimating GNU/Linux's size, 2001. `http://www.dwheeler.com/sloc`