

# **Quality Assurance under the Open Source Development Model**

Luyin Zhao  
Philips Research USA  
Phillips Corporation  
luyin.zhao@philips.com

Sebastian Elbaum  
Computer Science and Engineering  
University of Nebraska-Lincoln  
elbaum@cse.unl.edu

## **Abstract**

The open source development model has defied traditional software development practices by generating widely accepted products (e.g., Linux, Apache, Perl) while following unconventional principles such as the distribution of free source code and massive user participation. Those achievements have initiated and supported many declarations about the potential of the open source model to accelerate the development of reliable software. However, the pronouncements in favor or against this model have been usually argumentative, lacking of empirical evidence to support either position. Our work uses a comprehensive survey to overcome those limitations. The study explores how software quality assurance is performed under the open source model, how it differs from more traditional software development models, and whether those differences could translate into practical advantages. The findings indicate that open source has certainly introduced a new dimension in large-scale distributed software development. However, we also discovered that the potential capabilities of open source could not be exploited under all scenarios. Furthermore, we found that many of the open source quality assurance activities are still evolving.

## **1. Introduction**

Our interest in development a survey on this subject originated in the popularity gained by the open source model in the last few years through the delivery of successful products such as Linux, Apache, Perl and sendmail. This software development model seemed to be yielding products in rapid succession and with high quality, without following traditional quality practices of accepted software development models [Raymond, 1999]. Furthermore, it appears that the

emergence of open source might be able to challenge certain established quality assurance approaches, claiming to be successful through techniques and principles that defy some of the current and standard software development practices.

Under a traditional software development model, software quality assurance constitutes the set of systematic activities providing evidence of the ability of the software process to generate a software product that is fit to use [Schulmeyer, 1999]. The activities that a traditional quality assurance group carries out, and the quantification mechanisms for those activities, have been the focus of considerable research. For example, the effectiveness of inspections and reviews, the efficiency of testing techniques, and the impact of certain development processes have all been extensively investigated [Fagan, 1986; Porter, 1995; Frankl, 1998; Rothermel, 1996; Perry, 1994;].

However, the quantification of such activities for open source is not abundant, which makes it difficult to support or deny the model's claimed advantages [Glass, 2001]. In a preliminary survey, we confirmed some of the uniqueness of this model [Zhao, 2000]. For example, we found that the most popular open source projects encouraged (and leveraged) user participation to levels not observed in more traditional software development environments. However, we also discovered that some of the open source claims could not be substantiated with the existing survey and data. In this paper, we present a more comprehensive survey study that attempts to further uncover how software quality assurance is performed on the open source model, how it differs from more traditional models, and whether those differences could translate into practical advantages.

In the next section we present the related work in this area and provide a concise introduction to the open source development model and its major claims. Section 3 presents the survey methodology. Section 4 introduces the major survey results. Last, Section 5 summarizes the findings of this effort.

## **2. Related Work**

### **2.1. "Open source model"**

The open source initiative and its followers propose a software development model that promotes free distribution and

complete access to source code [OpenSource, 2002; Wu, 2001]. This model has been labeled the “open source development model”, “the open source model”, or just plainly “open source”. Its origins can be traced back to the “hacker culture” that created Unix, Linux, and parts of the Internet infrastructure [Raymond, 1999; Wu, 2001]. However, recent success stories of many products developed under this model have given it enormous momentum and visibility, making it an interesting alternative for large software development companies [Behlendorf, 1999]. Netscape pioneered this movement by making its browser publicly available in 1998. Others companies such as IBM, Apple, and SGI soon started to explore this path [OpenSource, 2002]. This growth sparked a need to capture the attributes that make open source products successful [Wang, 2001], and our work fits that profile, within the area of quality assurance.

The argument behind the open source model is that source code availability allows faster software evolution. The idea is that multiple contributors can be writing, testing, or debugging the product in parallel, which supposed to accelerate software evolution. Raymond repeatedly observes that more people looking at the code, will results in more “bugs” found, which is likely to accelerate software improvements [Raymond, 1999]. User participation is then a major foundation in this model, and the motivators of that commitment have been discussed in a recent survey [Har, 2001]. The model also claims that this rapid evolution produces better software than the traditional closed model because in the later “only a very few programmers can see the source and everybody else must blindly use an opaque block of bits” [OpenSource, 2002]. In this paper, we will be trying to quantify and analyze these statements from the quality assurance perspective.

## **2.2. Quality assurance under the open source model**

Despite the number of heated informal discussions on and about the open source development model, empirical studies regarding open source quality assurance activities and quality claims are rare [Glass, 2001]. One of the exceptions is the study Mockus performed on the popular Apache web server [Mockus, 2000]. This study gives a fairly comprehensive analysis of developer participation, core team size, code ownership, productivity, defect density, and problem resolution

intervals by comparing the Apache server with other five commercial products.

[Wu, 2001] and [Cubranic, 1999] focus on cooperative work and configuration management to support distributed development. [Cubranic, 1999] discusses major issues of coordinating open source development projects, including collaborative communication mediums and configuration management tools. Criticisms about the lacking of high-level coordination approaches in open source community are raised, emphasizing that the most successful open source projects have a centralized control structure (e.g. Linux, Apache server). [Wu, 2001] concentrates in the study of three version control systems – diff and patch for Unix, RCS and CVS – that support the coordination effort required by open source development.

Another aspect that has been the target of some attention is the reliability of open source systems. The “fuzz papers” [Miller, 1995] studied the reliability of several Unix products. Among those products was the Linux operating system, one of the open source flagship products. The results indicated that the reliability of those open source products was superior to those of commercial vendors, with the caveat that the compared products did not provide exactly the same services [Glass, 2001].

Our preliminary survey on open source [Zhao, 2000] explored the quality assurance activities in open source. The results indicated that testing takes a significant portion of the software life cycle, planning is not regularly done, and user participation usually did not include looking at the source code. The findings and limitations of this study served as the motivators for the current work.

### **3. Survey Methodology**

#### **3.1. Goals**

We established three main goals for this study. First, we want to capture the quality assurance techniques used by open source developers and their perceptions about software quality. We would like to understand developers’ expectations regarding the quality of their product, and whether those are met through the techniques they employ. Second, we want to

quantify how much the software user really contributes to the software evolution. Quantifying this aspect will provide unique evidence on the level and type of user participation, which constitutes one of the major claimed reasons behind the open source model success. Last, we want to determine if attributes such as project size, maturity, and programming language have an impact on how quality assurance is carried. Answering this last question will provide a valid context for the generalized statements made about the open source model potential.

### **3.2. Data collection**

The first task was to identify the sample universe of open source products to initiate our study. We were familiar with many web sites hosting open source projects over the world. We decided to limit our universe to [www.sourceforge.net](http://www.sourceforge.net) and [www.freshmeat.net](http://www.freshmeat.net) (henceforth referred to as sourceforge and freshmeat) because they are well known and respected, they host a large number of open source projects<sup>1</sup>, they include a large variety of projects, and they have the support and participation of several leaders of the open source community. Then, in order to reflect projects that are active and evolving, we reduced the pool of subjects based on posting dates, considering all projects posted on freshmeat from Dec 4<sup>th</sup>, 2001 to Jan 4<sup>th</sup>, 2002 (1532 postings), and all projects on sourceforge from Dec 28<sup>th</sup>, 2001 to Jan 25<sup>th</sup>, 2002 (1549 postings).

The next step was to stratify the refined universe. Our experience in the pilot survey [Zhao, 2000] clearly indicated that certain factors such as programming language could have a large impact on the quality assurance activities. Hence, we defined three attributes to control the influence of these factors, and obtain a sample with an even number of observations per attribute combination. The attributes are programming language (java, c, scripting), environment (X11, windows32, web, daemon, console), and topic (communications, databases, games, internet, desktop, software development, system). We then proceeded to randomly select a project for each combination group until we had eight

---

<sup>1</sup> In December 2001, there were over 30,000 open source projects registered on sourceforge and approximately 9,000 projects on freshmeat (with some overlapping.)

observations in each, or there were not more possible observations for that group. With this procedure, the maximum number of surveyed projects turns out to be 840 (7 topics \* 5 environments \* 3 languages \* 8 on each combination). Based on the fact that sourceforge hosts 3 times more projects as freshmeat, we assigned 630 to sourceforge and 210 to freshmeat. Although a higher number of observations per combination would be desirable, the number eight was selected in an attempt to ensure that all combinations are balanced in terms of available observations. However, and in spite of the large universe, some combinations did not have even eight observations. We found that some combinations had very few projects (e.g., system applications are rarely programmed in Java) and many projects did not provide all the needed attributes. In spite of that, our conservative scheme let us target 474 open source projects, with an unbiased distribution across programming languages, environments, and topics. While planning the sampling scheme, we started the preparation of the questionnaire with 22 items organized into four groups: project characterization, respondent characterization, process, testing, and user participation and feedback. To maximize the accuracy of the answers, and minimize the load on the respondents, the questionnaire followed a multiple-choice format. In addition, an explanation describing the purpose of this survey was attached to the distributed survey. The questionnaire can be found in the appendix at the end of the paper.

#### **4. Results**

The data collection process was automated whenever possible. Several scripts were developed to retrieve information from web sites and contact respondents. We received a total of 232 responses, which corresponds to a respond rate of 48.5%. We did not perceived any pattern among the respondents and non-respondents that could represent any bias in the collected data. Out of the total responses, 229 were used for data analysis (three observations were not used due to corrupted response file or lack of responses). We then proceeded to interpret the collected data, using Microsoft Excel and Statistica to assist us in the analysis. The findings are presented in the following sections.

#### 4.1. General descriptive findings

The questions grouped under project characterization were used to provide a general appreciation of the project profile, including project size, staffing, number of users, release frequency, and time in the market. In our sampling, almost half projects fall into Medium size category (1,000 – 10,000 lines of code). Other projects are distributed in the remaining size categories as follows: Small – 10% (less than a 100 lines of code), Large – 31% (10,000 – 100,000 lines of code), and Huge - 6% (more than 100,000 lines of code). Although we are not certain of the methods employed to compute the lines of code, this estimate helps us to quantify the impact of product size on different quality attributes.

The vast majority of projects are developed by small groups of less than 5 members (51% projects have 1 developer, 43% have 1 to 5 developers). Only 5% projects have more than 5 developers. However, over half (59%) of the projects said to have user groups with more than 50 people, 15% projects have 10 to 50 users, 13% have 5 to 10 users, and 10% have less than 5 users. Again, these numbers constitute developers estimates.

We also found that for 43% of the products, a new version is released every month. The average release intervals with other projects are: 29% - Every Quarter, 11% - Every Week. Very few projects (7%) have new versions every 6 months, while 10% projects answered “Other” or stated that their new versions were released very irregularly. An interesting finding was that only 10% of projects have been in the market for more than 3 years, while 39% projects have been in the market for less than 6 months. Most of the projects start in the small and medium category, and tend to grow steady as they reach maturity, with over 50% of the projects being Huge by their third year in the market. There are some exceptions to this tendency within the Large and Huge new projects; generally, these are existing projects that have recently moved to open source. Figure 1 provides more detail about the relationship between size and project maturity.

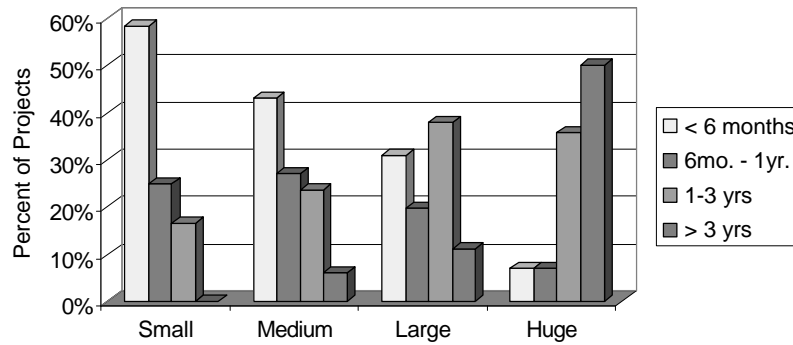


Figure 1 – Time in market by project size

We also attempted to characterize the respondents to determine how it could impact the other responses. Although the questions leave some latitude regarding the quality of the “years of experience”, it is interesting to note that most responses came from developers with several years of experience. For example, 61% of the respondents had more than 5 years development experience, while 33% had 1 to 5 years experience. It was also very interesting to observe that 77% of the respondents performed open source development in their personal time (12% were partially supported by employer, and only 5% respondents are dedicated to open source full-time). This actually seems to reflect the “giving spirit” described by the open source promoters [Hars, 2001].

## 4.2. Process

From the pilot survey [Zhao, 2000] we learned that open source projects largely stay in the “ad-hoc” phase in terms of traditional software process engineering as formulated by models like the Capability Maturity Model [Humphrey, 1989] and ISO 9000 [Baker, 2001]. On the other hand, many of the “key process areas” defined by these process frameworks may not be applicable to open source software, making it unfair to assess them from this perspective. Therefore, our questions try to collect certain process-related data that are more likely to be adopted by the open source community. The questions consider project purpose, changes between releases, configuration management, fault tracking systems<sup>2</sup>, and documentation.



First, we validated the anecdotal evidence presented by Raymond [Raymond, 2000] regarding the origin of the open source projects. Close to 60% of the projects started to meet personal needs (categorized as “external rewards motivation” by Raymond and also in [Hars, 2001]), while 28% products were initiated with the software community in mind, and 24% for company needs<sup>3</sup>. However, as projects mature and grow in size to fit the needs of more users, the tendency becomes less obvious as exposed by Figure 2. It was also interesting to see that while 50% of Web applications were started to meet company needs, less than 30% of the other type of applications had that objective, indicating a greater likelihood of open source development in a traditional business environment if the target application fits the web domain.

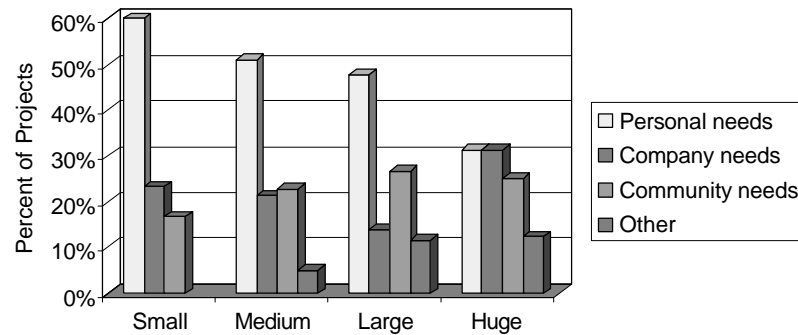


Figure 2 – Project startup purpose by project size

A second aspect we evaluated was the use of software configuration management processes and tools. Since open source is based on the premise of extensive (and likely distributed) collaboration, this aspect becomes even more critical (as addressed in [Cubranic, 1999]). In our survey, approximately 75% of the respondents use configuration management tools. Within these projects, 89% percent use the CVS tool [Bar, 2001], which was provided by default for the hosted projects. Interestingly enough, only 2% answered “not sure” to this question, the rest of the respondents were aware of the availability of these tools and their purpose. The percentages are similar for bug tracking tools. Over 61% of the projects also employ bug tracking tools, and a majority of projects use bug tracking tools provided by the host web sites.

As expected, larger projects made more extensive use of the configuration management and fault tracking

<sup>2</sup> Problem tracking and fault tracking systems are generally referred to as “bug” tracking systems within the open source community.

<sup>3</sup> Note that some projects have multiple purposes.

capabilities. However, it is noticeable that almost 50% of the small and most recent products make use of these facilities.

Figure 3 and 4 provides more details about the use of these supporting tools.

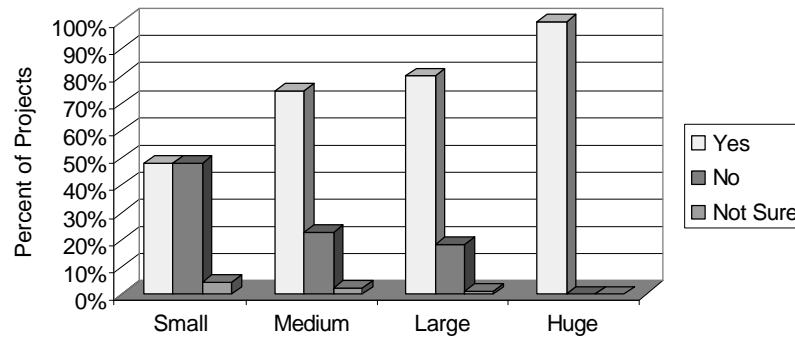


Figure 3 – SCM Usage by project size

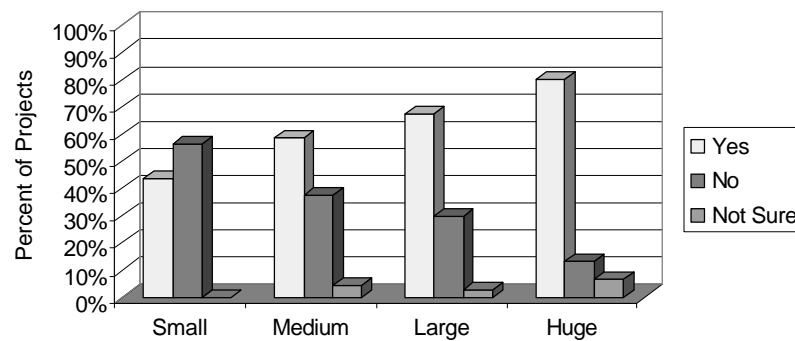


Figure 4 – Bug tracking tools usage by project size

Documentation did not play such a dominant role. Over 84% of the respondents prepare a “TODO” list (including list of pending features and open bugs). 62% build installation and building guidelines, 32% projects have design documents, and 20% have documents to plan releases (including dates and content). As evident from Figure 5, these numbers do not vary much across the type of application type, nor did they change due to application size or time in the market.

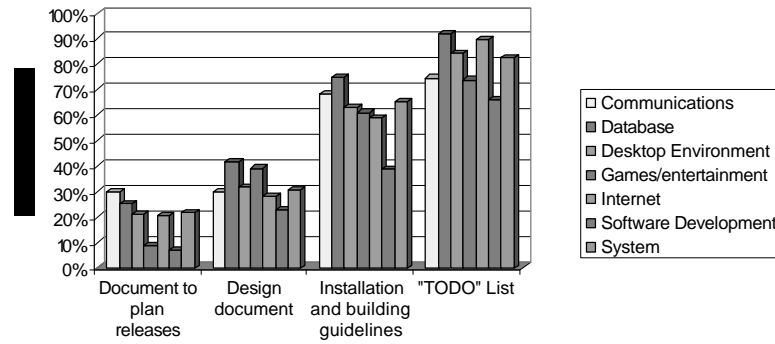


Figure 5 – Documentation by project topic

### 4.3. Testing

Traditionally, testing constitutes the last validation stage to ensure that a product meets the user requirements and quality specifications. This activity seems to receive less importance in open source, where some of the validation responsibilities are (supposedly) transferred to the user [Vixie, 1999]. In the survey we found that 58% of the projects spent more than 20% of the time on testing, while more than 15% of the projects spent more than 40% of their time in testing. This confirms the existence of a testing activity that consumes significant resources, even though some of the testing responsibility is shifted to the user. Figure 5 provides more insights about the testing time discriminated by project size. It seems that larger projects tend to spend less time in their testing phase compared with smaller projects.

Our findings also indicate that 68% of the respondents “provide inputs trying to imitate user behavior”, 25% “provide extreme values as inputs”, 25% “use assertions (assert, Junit, others.)”, and 26% people adopt other validation methodologies. The respondents were familiar with the notion of coverage, but only 5% of them employed tools to assess it accurately. Furthermore, almost 30% of the projects had an estimated coverage of less than 30%, independently of project size or maturity. Given the large percentage of time spent at the testing stage, this lack of attention to basic, accepted, and mature testing techniques was surprising. However, as projects get larger, the validation techniques tend to become a bit more mature. For example, almost 35% of the Huge projects use some kind of assertions. Figure 7 presents more details about the validation activities across program size.

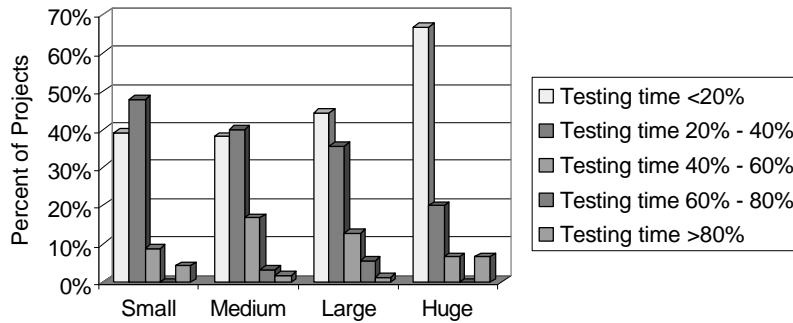


Figure 6 – Testing time by project size

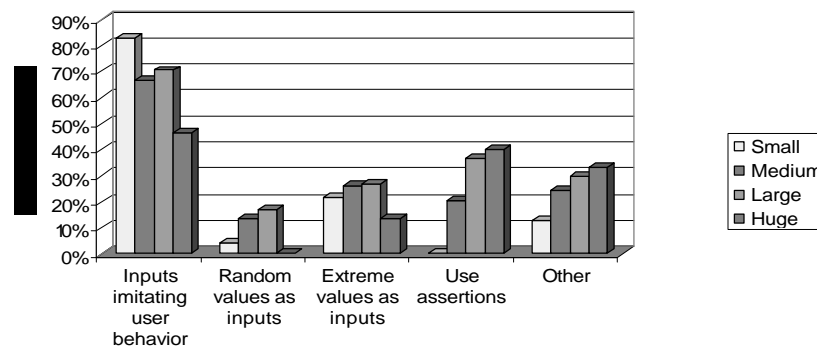


Figure 7 – Validation by project size

Another interesting fact is that only 48% of the projects use a baseline test suite to support testing. This is surprising because the lack of a baseline indicates the likely absence of regression testing, which could jeopardize the ability to generate multiple “reliable” releases in a short time frame (see Section 4.1). Even among the Huge systems, only 53% had a regression test suite. When categorizing by programming languages we found that over 69% of the Java projects used baseline test suites, while for C/C++ it was 41%. The availability of more open source tools to support the testing of software developed in java might explain this variation among programming languages.

#### 4.4. User participation and feedback

Open source approaches clearly aim to put more emphasis on field-testing and user reviews, taking advantage of the user’s willingness to experiment with an “unpolished”, but free product [Vixie, 1999]. This user participation and

feedback constitutes one of the backbones of the open source model [Raymond, 1999], but until now there were no clear indications about the effectiveness or efficiency of that process. We found that user suggestions generate over 20% of the changes on almost 50% of the projects. We also found that in almost 20% percent of the projects, the users discovered 20% to 40% of the bugs, and 44% of the respondents thought that users found “hard” bugs (not likely to be found by the developers).

Although not as extreme, respondents thought user suggestions to code change are usually “reasonable”, and only 14% people thought the users “don’t help too much”. Figures 8 and 9, and Tables 1 and 2 provides more details about the user participation in the open source model.

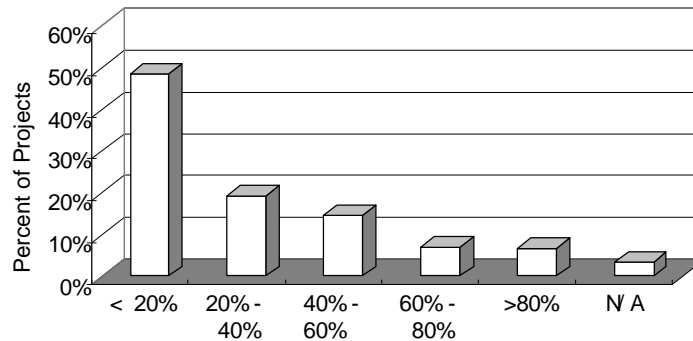


Figure 8 – Percent of “bugs” found by users

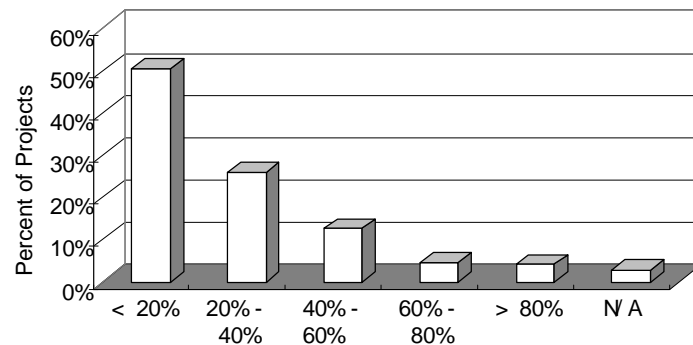


Figure 9 – Code changed by user suggestions

<b>Bug locating effectiveness</b>	<b>% of Projects</b>
They found "hard" bugs that could have taken us a long time to find	44%
Given some more time, I would have found most of them	30%
They don't help too much	14%
Other	15%

Table 1 – Bug locating effectiveness

<b>User suggestion usefulness</b>	<b>Percent of Projects</b>
Very creative	25%
Reasonable	55%
Useful but not so necessary	11%
Not fitting into my application design	8%
Other	16%

Table 2 – User suggestion usefulness

When categorizing the influence of project size on user feedback, we found that for the choice “Users found hard bugs”, the percentage is higher in Large and Huge projects than in Medium and Small projects. For example, in Huge projects users find 80% of the “hard” bugs, while in the Small and Medium projects users find an average of 40% of the “hard” faults. Also, for the question on user change suggestions the choice “Not fitting into my design” was selected in approximately 20% of huge projects, while this response is almost absent in small projects, which indicates a higher flexibility in smaller projects. It is also interesting to see that user participation is reflected in a shorter feedback loop. This is more evident in large products where, given the large number of users, the feedback is received sometimes in hours as evident in Figure 10.

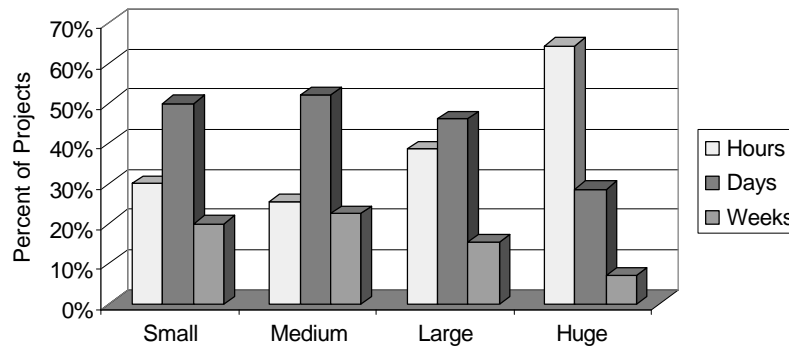


Figure 10 – How soon to hear user feedback by project size

As shown in Figure 11, “Users found hard bugs” for almost 60% of web applications, while only 30% for Win32 applications. This could be explained by the nature of web application, which are distributed among many heterogeneous components and platforms that make them particularly hard to validate for an individual developer. However, more regular problems were found for Win32 applications than other categories. For the choice “They don’t help much”, Daemon applications take the highest percentage while X11 is in the lowest percentage. Last, we found that user contributions in terms of percentage of faults found increased for more mature products. This was expected, as products that have been in the market for longer periods of time are likely to have a greater user base. This also implies that even in the presence of a larger market and a mature product, user participation remained consistently high. Last, in Figure 12, we observe the relationship between the time invested in testing and the percentage of faults found by the users. As expected, it is clear that users find less number of faults in projects that spend more time in testing.

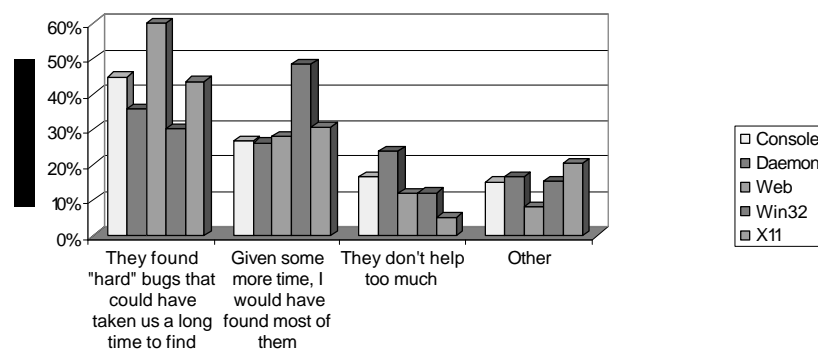


Figure 11 – Bug locating effectiveness by project environment

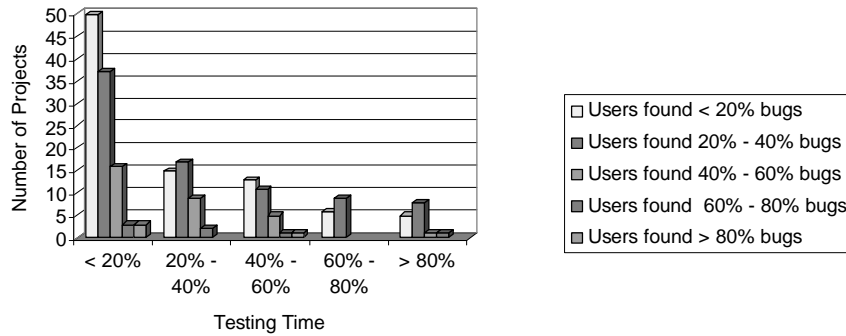


Figure 12 – Testing time vs. the percentage of faults found by users

## 5. Conclusions and final remarks

Through this study we have gained a greater appreciation of the quality assurance activities employed in the open source model. Perhaps more important is that we were also able to collect evidence and quantify certain open source activities to more objectively assess the virtues of this model. We now proceed to summarize our findings, their implications, and how they can adjust the expectations on this software development model.

First, we found that the level of user participation in open source projects was notoriously high, generating up to 20% of the changes for almost 50% of the projects, and discovering 20% to 40% of the faults in 20% of the projects. This substantiates one of the potential advantages of the model in terms of having the resources available to parallelize, for example, debugging tasks. Note, however, that the activities carried by those users were significantly different depending on project size. In general, smaller projects cannot expect much contribution from the users except for feature suggestions. On the other hand, larger and more matured applications can expect users that will contribute to fault finding, but individual user's feature suggestions are likely to be discarded.

Second, we found evidence that supports the life cycle of open source projects described by Raymond [Raymond, 1999]. Almost 60% of the projects were started to meet the developer's personal needs, later migrating to the community (if that need was common to many users) and growing in size while trying to accommodate an increasing number of features. This is a commonality with non-open source projects: successful projects grow as part of their evolution [Lehman, 1985]. However, we also found an increasing number of large projects that have been moved to the open source



model, which might indicate a new tendency among the latest open source projects.

Third, the use of configuration and bug tracking tools to support collaborative and distributed software development reached approximately 75% of the projects. The open source processes and tools for change management employed by some of the projects definitely seemed to be at the cutting edge of large-scale collaborative software development [Mozilla, 2002]. However, we also realize that these numbers might be confounded by the fact that the host sites for all these projects provided a supporting tool and infrastructure to manage change (e.g., CVS repositories are provided by default by sourceforge). Still, it was very interesting to find that these types of host sites can have such a degree of impact, educating the developers and effectively shaping the open source model.

Fourth, documentation was not a high priority for most projects. A simple "TODO" list and installation guidelines were the most common documents. Although this was expected from the often loosely controlled evolution of the open source projects, it was striking that less than 20% of the projects have planned release dates. It was not clear that even mature and large projects had more formal planning, which might be a bit unsettling for companies depending on these programs.

Last, open source projects supposed to take advantage of the users to validate the software. With that premise in mind, most developers seemed to invest no time in utilizing any testing techniques or even tools. For example, most developers just attempt to informally imitate user behavior to test their software, and only 5% employ any tools to compute any type of test coverage. Another example is the lack of regression testing in spite of the high release rate observed across most projects. The contradiction is that over half of the projects spend over 20% of the time testing their system. Since in-house testing still takes a considerable amount of time under the open source model, embracing known and mature testing techniques would seem recommendable.

Overall there is still much to be learned about the open source model. With this effort we have started to quantify this model and its potential. However, further studies are necessary to provide additional empirical evidence to support the

claims of open source promoters, to put those claims into the proper context, or to just contradict those claims. Our future work will head in that direction.

## **Acknowledgments**

We are very thankful to all the open source developers that took the time to respond to our survey. We also want to thank the J. D. Edwards Honor Program for supporting Luyin Zhao while he was attending at University of Nebraska-Lincoln, and Philips Research USA for understanding the value of this work.

## **References**

- [Baker, 2001] Baker, E. 2001. Which Way, SQA? IEEE Software, Vol. 18, No. 1, 16-18.
- [Bar, 2001] Bar, M. and Fogel, K. 2001. Open Source Development with CVS, 2nd Edition, Coriolis Group, Scottsdale, Arizona.
- [Behlendorf, 1999] Behlendorf, B. 1999. Open source as a business strategy, Open Sources: Voices from the Open Source Revolution, 1<sup>st</sup> Edition, O'Reilly, 149-170.
- [Cubranic, 1999] Cubranic, D., and Booth, K. 1999. Coordinating Open-Source Software Development, Proceedings of IEEE 8th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 61-69.
- [Fagan, 1986] Fagan, M. 1986. Advances in Software Inspections, IEEE Transactions in Software Engineering, Vol. 12, No. 7, 744-751.
- [Frankl, 1998] Frankl, P., Hamlet, R., Littlewood, B., and Strigini, L. 1998. Evaluating Testing Methods by Delivered Reliability, IEEE Transactions on Software Engineering, Vol. 24, No. 8, 586-601.
- [Glass, 2001] Glass, R. 2001. Is open source software more reliable? An elusive answer, The Software Practitioner, Vol. 11, No. 6.
- [Hars, 2001] Hars, A., and Ou, S. 2001. Working for Free? - Motivations of Participating in Open Source Projects,

Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2284 - 2292.

[Humphrey, 1989] Humphrey W. 1989. Managing the Software Process, The SEI Series in Software Engineering, Addison Wesley.

[Lehman, 1985] Lehman, M., and Belady, L. 1985. Program Evolution – Processes of Software Change, Academic Press, London, UK.

[Miller, 1995] Miller, B., Koski, D., Lee, C., Maganty, V., Murthy, R., Natarajan, A., and Steidl, J. 1995. Fuzz Revisited: A re-examination of the reliability of UNIX utilities and services, <http://www.cs.wisc.edu/~bart/fuzz/fuzz.html>.

[Mockus, 2000] Mockus, A., Fielding, R.T., and Herbsleb, J. 2000. A Case Study of Open Source Software Development: The Apache Server, The 22<sup>nd</sup> International Conference on Software Engineering. 263-272.

[Mozilla, 2002] Mozilla. 2002. Mozilla tinderbox framework, <http://tinderbox.mozilla.org>.

[OpenSource, 2002] Open Source Initiative. 2002. <http://www.opensource.org/>.

[Perry, 1994] Perry, D., Staudenmayer P., and Votta, L. 1994. People, organizations, and process improvement. IEEE Software, volume 11, number 4, 36-45.

[Porter, 1995] Porter, A., and Votta, L.. 1995. Comparing Detection Methods For Software Requirements Inspections: A Replication Using Professional Subjects, Empirical Software Engineering: An International Journal, Vol. 3, No. 4, 355-379.

[Raymond, 1999] Raymond, E.S. 1999. Linux and Open-Source Success, IEEE Software, Vol. 16, No. 1, 85 - 89.

[Rothermel, 1996] Rothermel, G. and Harrold, M. 1996. Analyzing Regression Test Selection Techniques, IEEE Transaction on Software Engineering, Vol. 22, No. 8, 529-551.

[Schulmeyer, 1999] Schulmeyer, G, and McManus, J. 1999. Handbook of Software Quality Assurance. Prentice Hall.

[Vixie, 1999] Vixie, P., 1999. Software Engineering, Open Sources: Voices from the Open Source Revolution, 1<sup>st</sup> Edition, O'Reilly, 91-100.

[Wang, 2001] Wang, H., and Wang, C. 2001. Open Source Software Adoption: A Status Report, IEEE Software, Vol. 18,

No. 2, 90-95.

[Wu, 2001] Wu, M.W., and Lin, Y.D. 2001. Open Source Software Development: An Overview, Computer, Vol. 34, No. 6, 33-38.

[Zhao, 2000] Zhao, L., and Elbaum, S. 2000. A Survey on Software Quality Related Activities in Open Source, Software Engineering Notes, ACM, 54-57.

## **Appendix: Survey Questionnaire**

### **Part A: Project characterization**

1. What is the estimated number of lines of code of the project?
  - A. < 1,000
  - B. 1,000 - 10,000
  - C. 10,000 - 100,000
  - D. > 100,000 Lines of code
  
2. How many software developers are actively involved in this project?
  - A. 1
  - B. 1-5
  - C. 5-20
  - D. +20
  
3. What is the estimated current number of users of this product?
  - A. 1-5
  - B. 5-10
  - C. 10-50
  - D. +50
  
4. How often are the product releases (on average)?
  - A. Every week
  - B. Every month
  - C. Every quarter
  - D. Every 6 months
  - E. Other
  
5. How long has the product been available in the market?
  - A. Less than 6 months
  - B. Between 6 months and a year
  - C. Between 1 and 3 years
  - D. More than 3 years

**Part B: Respondent characterization**

6. Software development experience

- A. <1 year
- B. 1-5 years
- C. +5 years

7. What level of participation do you have in the project?

- A. Dedicated full-time
- B. Part-time, supported by employer
- C. Part-time, personal time
- D. Other

**Part C: Process**

8. Did the project start to satisfy:

- A. Personal needs
- B. Company needs
- C. Community needs
- D. Other

9. What percentage of your product changes from release to release (major releases)?

- A. < 20 %
- B. 20% - 40%
- C. 40% - 60%
- D. 60% - 80%
- E. > 80%

10. Do you use software configuration management tools (version control tools) ?

- A. Yes (Name      )
- B. No
- C. Not sure

11. Do you use any "bug" tracking tool?

- A. Yes (Name      )
- B. No
- C. Not sure

12. Which of the following documents is used to support the project?

- A. Document to plan releases (dates and content)
- B. Design document
- C. Installation and building guidelines
- D. "TODO" List (including list of pending features and open bugs)

**Part D: Testing**

13. How do you validate your product before release?

- A. Provide inputs trying to imitate user behavior (ad-hoc)
- B. Use script to provide random values as inputs

- C. Provide extreme values as inputs
- D. Use assertions (assert, Junit, others)
- E. Other

14. What percentage of your time and effort is spent on testing?

- A. < 20 %
- B. 20% - 40%
- C. 40% - 60%
- D. 60% - 80%
- E. > 80%

15. Do you have a "baseline" test suite that you re-run on your software before every release?

- A. Yes
- B. No

16. What percentage of source code is covered by the testing activity?

- A. < 20 %
- B. 20% - 40%
- C. 40% - 60%
- D. 60% - 80%
- E. > 80%

17. The previous coverage information was based on:

- A. Reports by coverage tool (Name it: \_\_\_\_\_)
- B. Personal estimation

#### **Part E: Users Participation and Feedback**

18. How soon after release do you hear back from users?

- A. Hours
- B. Days
- C. Weeks

19. What percentage of "bugs" did users find?

- A. < 20%
- B. 20% - 40%
- C. 40% - 60%
- D. 60 - 80%
- E. >80%

20. What percentage of code has changed in response to users suggestions?

- A. < 20%
- B. 20% - 40%
- C. 40% - 60%
- D. 60% - 80%
- E. > 80%

21. How do you evaluate the "bug" locating effectiveness of external users?

- A. They found "hard" bugs that could have taken us a long time to find
- B. Given some more time, I would have found most of them
- C. They don't help too much
- D. Other

22. The modifications suggested by users, are:

- A. Very creative
- B. Reasonable
- C. Useful but not so necessary
- D. Not fitting into my application design
- E. Other