Assessing Free/Open Source Software Quality

Ioannis Samoladas, Ioannis Stamelos

Department of Informatics

Aristotle University of Thessaloniki

54124, Thessaloniki, Greece

email: {ioansam, stamelos}@csd.auth.gr

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Abstract

According to its proponents, one of the most acclaimed advantages of Free/Open Source Software (F/OSS) is its superior quality. However, this suggestion is an open issue, since there is little concrete evidence to justify whether F/OSS quality is indeed better or worse than that of proprietary software products. The general perspective of this article is to discuss the current status of F/OSS quality and to assess its performance in various aspects of quality, based on existing literature. Specifically, this article will provide some answers to various questions raised by the assertion concerning the quality of F/OSS. In this regard issues addressed in this article include the quality framework, through which F/OSS quality should be investigated and the performance of F/OSS in various quality factors within this quality framework. Answers to these issues are given by providing evidence from various research papers, empirical studies and reports based on experience about the quality of F/OSS products. The overall results seem to indicate that F/OSS has achieved an acceptable level of quality, although there is more to be done in order to outperform proprietary software.

Keywords: Free Software, Open Source Software, Software Quality

1. Free/Open Source Software

Free/Open Source Software (F/OSS) is relatively a new, alternative, idea of software

development in the area of software engineering. The term "open source software" was first coined in 1998 at a meeting of the F/OSS movement pioneers at Palo Alto (Feller & Fitzgerald, 2000). A piece of software (i.e. a software product) is F/OSS when its distribution license fulfills the "four freedoms" of F/OSS. To be more precisely, a software can be said that it is F/OSS when the receiver of the software can:

- use it at wish
- copy and redistribute it
- modify it (which imposes the distribution of the source code along with the binary version of the software. The binary version of the software and the source code can be separately, but freely distributed)distribute the modified versions

  Any software, which its distribution license satisfies the above, can be called F/OSS. There are mainly two major promoters of F/OSS: The Free Software Foundation, which is responsible for the famous GNU Project, and the Open Source Initiative which keeps a repository with "Open Source" compliant licenses[1].

The majority of F/OSS projects, probably most of them, use development practices, models and methods which very far away from those recommended by the "classical" software engineering, for example software development life-cycle models like waterfall and the spiral. The main idea behind the development model of F/OSS is simple. A single person or a group of people have an idea about a software product or want a particular software to address a particular need, so they write a first release of that software to satisfy their needs, their "personal itch" (Raymond, 2002). They put that software somewhere on-line, along with its source code, and ask for other people to contribute sending either bug fixes or functional improvement for their software. We have to note here that the distribution license of that software has to fulfill the "four freedoms" mentioned above. When people start making contributions, then F/OSS development has began. At some point these contributions

[1] Although there exist a controversy between these two groups about the definition of F/OSS, the "four freedoms" mentioned above is maybe the most widely accepted

are incorporated into the source code of the product and the next test version of the software is released. After some time of testing and new code submissions, the test version of the software becomes the next stable release. Then new contributions are made and this process of release, code contribution (bug fixing and/or functional improvement), code integration into the current one, next release continues in a circular manner. The evolution of the product is done by a single or a group of coordinators, who are responsible for deciding which piece of new code will be incorporated into the current one. The coordinators are usually the initial creators of the software and in most of the cases they have a strong impact on the evolution and even the success of the product. The contributors are usually spread worldwide, contributing over the Internet, and they don't know each other.

2. Software Quality

As McConnell (1999) pointed out, successful F/OSS projects should not be compared with the average proprietary (closed[2]) source project. They should be compared with the software development effectiveness achieved by leading edge organizations that use a combination of practices to produce better quality software and keep costs and schedules down. This assertion suggests that the quality of the delivered product should be compared with the quality levels pursued by the modern software industry. But first it is necessary to introduce the notion of software quality.

Just as any other product, for example a building, a car or a commercial electronic device, has a level of quality, software products have some kind of quality as well. By quality we mean whether or not the product conforms to a set of standards posed by someone, either by the manufacturer or by the customer. For example: Does the software do what the user wants it to do? Or is it well-designed, well coded, well tested, error free, so that it will function properly? For a more detailed discussion on "What is software quality?" see Kitchenham & Pfleeger (1996). In order to assess software quality, it is usual to construct models which combine different aspects of quality to produce a final result about the level of a software product quality. There exist several models of

[2] Throughout the article the terms closed source software and proprietary software are used interchangeably

software quality, which suggest various ways to bring together different quality attributes. Each one of these models tries to aggregate these several attributes of quality in order to give an overall view of the quality of a software. One of such models is the ISO 9126 model (International Organization for Standardization, 1991).

The ISO 9126 quality model was suggested in 1991 in order to combine the several views of software quality models that existed during that time. ISO 9126 is a hierarchical model consisting of six major attributes contributing to software quality. The attributes and what they represent are:

- Functionality
- Reliability
- Usability
- Maintainability
- Portability
- Efficiency

As with other models, these attributes do not necessarily lead to direct measurement, but to other indirect measurements, which can be further divided into other quality factors. Some of the above attributes, which can be measured using direct measurement, are usually expressed as some kind of equation, which has as inputs various source code metrics such as the McCabe's cyclomatic complexity (McCabe, 1976) or Halstead Effort Metrics (Halstead, 1975). An example of such equation is the Maintainability Index proposed by Carnegie Mellon's Software Engineering Institute (Software Engineering Institute, 2002). An extensive discussion and reference to software metrics can be found in Fenton & Pfleeger (1997).

It is worth mentioning here that software quality models and quality in general, is a field of a continuous debate on the correctness, for example, of the models or the attributes/factors/metrics they use to measure quality. In this article we preferred to include security in the reliability section. These quality factors have been examined in varying degree in the F/OSS literature. The rest of the article is structured according to the quality factors mentioned above. Hereto, we have to mention

here that it be ideal to use the same standard on some commercial systems in order to compare their quality versus the quality of F/OSS projects. However, it is difficult to find such assessments and quantitative studies, since many of them are hard to find or even confidential.

3. Functionality

The first quality factor we examine in this article is functionality. According to the ISO 9126 standard, functionality is "A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs" (International Organization for Standardization, 1991).The standard further noted that functionality is a set of attributes and "This set of attributes characterizes what the software does to fulfill needs, whereas the other sets mainly characterizes when and how it does so" (International Organization for Standardization, 1991). The rest of the section is based on the latter concept and explores the extend F/OSS fulfills this prerequisite.

F/OSS has produced products that in certain cases are a kind of "monopoly" in their application domain. Such a "category killer" is BIND (Berkeley Internet Name Domain) Server, a critical application for Internet's infrastructure. The BIND project was initiated as an academic project and soon became the dominant Domain Name System (DNS). Now it is being maintained by the Internet Software Consortium and its dominance and long time usage has proved that it implements the functionality of a DNS very well. Furthermore, the well-known Apache Web Server is used in 62.7% of the cases (Netcraft, 2003), implying that the functionality offered satisfies the majority of the web server administrators and hosting companies. There are numerous other examples of F/OSS products, which are being used by developers and users instead of closed source software, which performs the same functionality. Such examples are relational database management systems, programming tools and much more. Table 1 shows a summary of the major F/OSS products.

Application Type	***Projects***
Operating Systems	• *GNU/Linux* • *Free/Net/OpenBSD*
Graphical User Interfaces	• *GNOME* • *KDE* • *Ximian*
Office Suites, Graphics	• *OpenOffice.org* • *Koffice* • *GNOME Office* • *Gimp*
Network Applications	• *Apache* • *BIND* • *Sendmail* • *Mozilla* • *Samba*
Programming Languages/Compilers	• *GCC* • *Perl* • *Python* • *Tcl/Tk* • *php*

Table 1. Major F/OSS products

The majority of the F/OSS applications lie in the area of Internet or system applications. This is something that has been identified by some of the pioneers of F/OSS like Brian Behlendorf and Eric Raymond. They identified that open source projects was involved mainly in projects that had to do with networks or operating systems applications (Behlendorf, 1999 and Raymond, 1999). The majority of the successful[3] cases of F/OSS are coming from the areas mentioned above and not from the desktop or GUI applications category. Figure 1 shows the number of the projects of the top categories, hosted at Sourceforge.net (a major host provider for F/OSS projects). Behlendorf

[3] By the term successful projects, we mean projects that have an extensive installation base and/or their download numbers are significantly higher than that of others . The question "what is success in F/OSS" was one of the main that were discussed during a workshop about F/OSS at the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (see http://www.xp2003.org/workshop/parnas.html)

suggests that F/OSS development tends to be more effective in projects where incremental change is rewarded and he states that this kind of development applies to back-end systems rather than front ends. An answer to the question why developers spend more time on application from the network and system software category can be given if someone takes into account the "egoboo" argument by Raymond (2002). Raymond states that one of the main reasons that someone voluntarily contributes to an F/OSS project is the "ego-boosting" or the enhancement of his/her reputation among others. Network and system applications are software that everyone uses, so in order to gain reputation a contributor to an F/OSS project prefers this kind of projects.

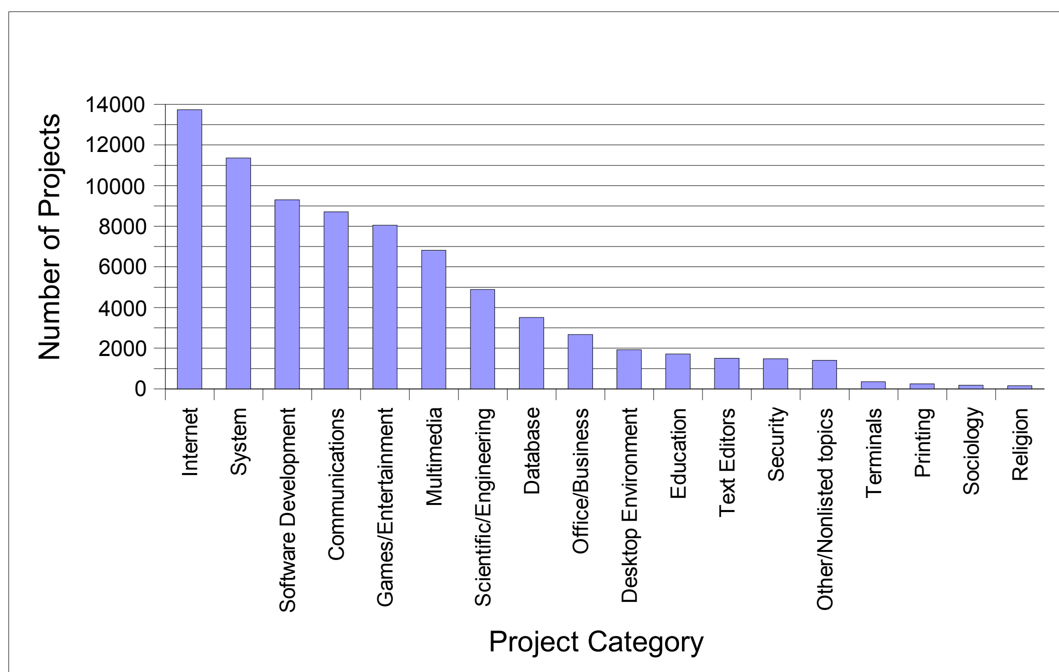


Figure 1. Number of projects per topic hosted at Sourceforge.net

Moreover, the existence of a large variety of F/OSS in the areas mentioned (network and system applications, programming tools and operating systems), reveals that this kind of development is also suitable (suitability is another factor of a software functionality) for applications with well defined (pre-defined) requirements. It can be said that F/OSS seems better suited to horizontal requirements[4], addressing the needs of broad user categories (McConnell, 1999). McConnell

[4] By the term "horizontal requirements" we mean requirements that address the needs of a broad set of users (e.g. a database management system). "vertical requirements" refer to systems that address specific needs of specific users

explained this lack of requirements by saying that “By the time Linux came around, requirements and architecture defects had already been flushed out during the development of many previous generations of UNIX”. Whether F/OSS is appropriate to vertical requirements, such as the Enterprise Resource Planning and/or Customer Relationship Management (ERP/CRM) systems that require a lot of customization remains to be seen. However, such systems exist and one good example is Compiere [http://www.compiere.org] with more than 460,000 downloads and frequently found in the top ten downloads list of Sourceforge.net hosting service [http://sourceforge.net]. It still remains an issue whether this project will reach the success level of the commercial ERP systems.

In addition to the above, F/OSS tries to be compliant with many standards that promote functionality and interoperability. The F/OSS movement sets as one of its first priorities compliance to open standards. By open standards we do not mean only standards that where developed in a democratic way, with anyone contributing and suggesting things. On the contrary, open standards mean that someone who wants to implement such standards does not have to pay for any patent or to sign any license that poses various limitations on the usage and the implementation of the standard. These limitations, that sometimes software companies pose, lead to incompatibility between different software products and raise interoperability issues. For example, UNISYS has posed some patent matters about the GIF image format which implements the LZW algorithm for image compression. This algorithm was patented after UNISYS and many products that implement this algorithm now have legal problems with UNISYS[5]. In fact that was the reason for the creation of, well known in the UNIX world, the GNU gzip: the Free Software Foundation wanted an alternative for the compress utility, which also implemented the LZW algorithm by Unisys. An interesting article about open standards and patents can be found at Coverpages, a resource center by OASIS (Coverpages, 2003).

In addition to how difficult is to implement some standards because of copyright reasons, some

(e.g. an engineering application).

[5] We have to note here that the GIF patent has expired in the U.S. and it will soon expire in other countries as well. See http://www.gnu.org/philosophy/gif.html

standards are hard to implement because they are not available to the public in an easy form and workable format, like on a web site. As Freericks (2001) suggests, open access to a standard sometimes is not enough. Standards should be available to the public in different workable formats, in order to facilitate their usage by the developers' community with the support of the appropriate tools and methodology. The openness of a standard also offers more intensive participation and feedback of the users, which at the end will force the standard to be more flexible and optimized. By this way, the open source approach might lead to a faster diffusion of a standard.

As mentioned above, the existence of open standards is very important for the interoperability of an application and a lot of organizations are supporting open standardship. Interoperability is a critical quality attribute related to functionality. Such examples are the World Wide Web Consortium, the Internet Engineering Task Force, OMG, the OASIS Group and the Open Group. F/OSS tries to stick to these standards when it implements an application that takes advantage of them. Indeed, F/OSS developers try to stick as much as possible to standards, open standards, and in many cases this has been the ultimate goal. Behlendorf (1999) recounted an incident, which happened in the early days of the Apache server, in which the America On Line's (AOL) proxy server responded with an error message under certain circumstances and when functioning with the Apache. The problem was tracked back to an anomaly in the implementation of the HTTP/1.1 protocol on the side of the AOL's product. The Apache group explained that their software implemented the protocol properly and the problem was a misimplementation of the AOL's proxy. AOL fixed the problem and everything went fine. Similar problems of interoperability are numerous in the case of the implementation of the HTML definition, where many web browsers and servers implement their own "customized" version of the HTML and do not perform properly with other products[6].

It is worth mentioning here that, for the Linux[7] kernel case interoperability is achieved with the

[6] A good example in favor of interoperability in F/OSS is the Gecko rendering machine of the Mozilla project. Gecko is maybe the only one that uses the latest standards from the W3C and many vendors (including a recent rumor about AOL) are supporting it.

[7] Linux is only the kernel of an operating system. Here for the sake of simplicity, with the term Linux we mean a

help of various modules (which mainly facilitate porting as mentioned below) that can be loaded into the kernel in order to achieve communication with other systems. An important F/OSS product that can offer invisible file and printing sharing among various platforms is samba [http://www.samba.org]. Samba makes it possible for machines with different file systems and network protocols to share files and printing services. This makes working in a multi-operating system environment easy and seamless.

4. Reliability

The current status of the global IT market makes reliability one of the most important factors of software quality. The ISO 9126 (International Organization for Standardization, 1991) model defines reliability as "A set of attributes that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time". With the presence of critical applications and applications that run over the Internet for a long period of time, it is crucial for organizations to have software that ensures all around the clock reliability. Examples of such large applications are on line shops, monitoring applications that use the Internet as a mean of communication, banking applications and so on. Proponents of F/OSS claim that F/OSS satisfies the condition that the ISO 9126 model poses and there is enough evidence to support that F/OSS is more reliable than proprietary software.

An important factor that directly affects reliability is the frequency of bug discovering. In this case, the status of F/OSS in bug discovering is described by the famous Eric Raymond (2002) postulation: "Given enough eyeballs, all bugs are shallow" (otherwise known as "Linus Law"). Indeed, feedback about bugs and misimplementations in F/OSS is direct and immediate: Once a version of software is released, it is a matter of days, hours or even minutes before the first bug is reported and the official fix is announced (Schmidt & Porter, 2001). An example of such immediate response is the case of the Teardrop attack described by Hissam, Plakosh & Weinstock (2002). In

---

distribution, the whole system, unless mentioned otherwise

1997 a denial-of-service (DoS) attack against the Linux operating system was reported. A flaw in the Linux kernel's IP stack caused the system to crash, when a special IP packet was received. What made this attack more interesting is that it was also affecting another well known closed source operating system, making Teardrop a case of a heterogeneous attack. For the Linux case, the vulnerability was fixed within few hours. Moreover, the patch which fixed the bug, contained a fix for another bug, which developers observed, but attacker didn't. The patch, which was originally aimed at the first bug, immunized the system against the second bug as well. This rendered Linux resisting future attacks of not only Teardrop itself, but of other invariants of it as well. The patch for the other closed source operating system took a lot longer to be published and successive patches were needed for the other kind of attacks to be resolved. Another fact that makes F/OSS more recoverable is that these patches are immediately put on line and are included in the code stored in the CVS[8] repository of the product. It is not strange to see a new version of software, one day after the previous one was released. In fact, during the early days of Linux, this was the case. In F/OSS there is no need for someone to wait for the company to publish a fix. In fact, even if a particular F/OSS project is abandoned by its original coordinators, there is always the possibility for someone else to take over, allowing people to continue using the software[9].

The strength of F/OSS comes to a large extend out of the principle mentioned above, namely fast bug fixing. But for this to become a reality, F/OSS depends heavily on the Internet and tools that operate over it, in order to make communication between developers fast and effective (Feller & Fitzgerald, 2002). Effective communication is needed not only between developers, but also between developers and users and between users themselves. It is important for the day-to-day users to have an organized way, a well organized set of tools for communicating with the coordinators of a project in order to submit bug reports and several other problems associated with the project . As Feller and Fitzgerald (2002) noted, “these tools are backed up with an ethos which recognizes quality

[8] CVS stands for Concurrent Versions System. It is the most widely used version control system in the F/OSS community.

[9] Microsoft has recently announced that it will abandon NT support. In the case of F/OSS, as mentioned, there is always the possibility for someone else to take over. A good example for that is GIMP.

contributions from any source and treats users as co-developers". These two points operate as a motivation factor for users and developers to submit either new functionality, bug reports and/or bug fixes. In the same book it is mentioned that feedback about the software is rather different from that of proprietary software, where the tight schedules and low budgets often limit testing and bug discovery. F/OSS users and developers, (in many cases a user is also a developer) constitute a large pool of beta testers, who try the software, report bugs and fix them, not only because it is their job to do so (which is the case in proprietary software), but because they use the software for their own needs.

Mockus, Fielding & Herbsleb (2002) conducted an extensive empirical study on the development of two open source projects, the renowned Apache Web Server and the Mozilla browser. They processed the mail archives, the CVS repository, the BUGDB and other sources in a period of three years in order to test a group of questions and formulate a group of hypotheses about F/OSS. In that study, the authors reported that in the Apache case, 458 people reported 591 problems that resulted in code change and 182 people did 695 fixes. For the Mozilla case, things were different with 113 people reporting 50% of the bugs. However 46 of these 113 people did not contribute any code for the fix.

A good study that present errors found in the Linux kernel by automatic, static, compiler analysis, is presented by Chou et al. (2001). The authors in this study examined the bugs found in 21 versions of the Linux kernel and compared the results against a version of the OpenBSD. They found that the part of the kernel were most of the bugs are concentrated, is the device drivers section, which has error rates up to three to seven times higher than the rest parts of the kernel. Furthermore they showed that the bug distribution among source code files follows a logarithmic distribution and the clustering of the errors pointed out that for the most heavily clustered error type, less than 10% of the files contained those errors. What is more is that, the average lifespan for certain types of the bugs across the 21 versions of the Linux kernel was found to be about 1.8 years. The comparison of the error rates of the OpenBSD against the Linux kernel showed that the OpenBSD has a higher

error rate, from 1.2 to 6 times higher.

In F/OSS, the time to resolve bugs is also short. In the study by Mockus et al. mentioned before, authors questioned the time it took to resolve problems in these projects, whether high priority problems were resolved faster than low priority problems and whether the resolution interval had increased over time. Results on the Apache project showed a cumulative 50% of the problems were fixed within one day, 75% within an interval of 42 and 90% within 140 days. Concerning priority, there was prioritization of the problems according to their nature. For example, problems that fell into the "Core" and the "Most sites" category were resolved as quickly as possible. Regarding resolution interval, it was found that there was indeed a reduction and the interval was significantly shorter in the most recent periods of the software as studied. As the authors point out, "this indicates that important aspect of customer support improved over time, despite dramatic increase in the number of users". For the Mozilla project the median resolution interval is much longer than for Apache. This happens mainly because in Mozilla the interval between the fix submission and fix verification by the project coordinators is much longer.

One additional feature, which plays a critical role in software product quality, is defect density. Again Mockus et al. present an extensive study on defect density for the Apache and the Mozilla projects, and compare it against four other commercial products. In order to measure the defect density they used, as a metric, the defects per thousand lines of code added (KLOCA). They found that "Although the user-perceived defect density of the Apache project is inferior to that of the commercial products, the defect density of the code before system test is much lower". This means that the development process used for the Apache assures that fewer bugs are inserted into the source code. In addition, inspections and tests conducted by the group of the developers are more efficient in finding bugs. The same results were found to be true for the Mozilla project.

Reasoning ©, a company that provides automated software inspection services for many IT companies, has made some recent reports that compare the defect density between F/OSS products and closed source software publicly available. The first one of these studies (Reasoning, 2003),

analyzed the code of the Linux kernel component that handles the TCP/IP networking and compared it against a sample of 160 projects (22 million lines of code). The second study measured Apache2.1-dev, which was compared against a sample of 200 projects (35 million lines of code). Results for both studies are presented in Table 2. The last study conducted by Reasoning© was on Tomcat and it was found to have a defect density of 0.24 defects/KSLC (Kilo Source Lines of Code). All the above results indicate these, industrial sized, F/OSS applications achieve a low defect density.

Application Name	***Application Average (defects/KLSC)***	***Industry Average***		
		Best Third	***Middle Third***	***Worst Third***
Linux 2.4.19 Networking Sample	0.10	< 0.15	0.15 – 0.35	> 0.35
Apache 2.1-dev	0.53	< 0.36	0.36 – 0.71	> 0.71

Table 2. Defect results as presented by Reasoning©

From the above, it can be conjectured that one of the strongest points of F/OSS is indeed its effectiveness in bug fixing. One of the main reasons for this effectiveness is the rapid release of the product along with its source code. This helps people to inspect and debug the code at the time they discover a bug, if they are capable of doing so, and, if they are lucky, to fix the software and send the correct version to the coordinator and from there to the whole community. Along with the availability of the source code, debugging is made effective because of the lack of any formal debugging/testing process and most of it is done in operational mode of the system. As Vixie (1999) points out, developers are friendlier when they are doing something that they are not paid for or not responsible for. The lack of pressure makes them more willing to read and fix the software and share that fix with the rest of the community. Additionally the effectiveness of debugging in F/OSS is achieved through the true peer review of the code. Without having to meet each other, the developers do not have any reason not to support the peer review process, thus making it more reliable (Feller & Fitzgerald, 2002) and more effective.

One important question raised is whether F/OSS is more secure than proprietary software. This topic is one of most frequent subjects on debates about F/OSS. Indeed many people support the idea of code openness, which gives people the opportunity to inspect the code for potential flaws (Wheeler, 2003). It is not only the algorithm that has to be open (especially cryptographic algorithms), but the code, too. It is not difficult for someone to assert that it is not only the developer that has access to the code, but the potential attacker, too. So what is the benefit? Witten Landwehr & Caloyannides (2001) suggested that having closed source software, someone cannot be sure about the content of it. It is not only the source code itself, but the compiler that was used to make it executable. It is possible for the compiler to insert various malicious parts in the software for later use. Open source software let the users review their code and investigate the existence of such back doors and other kinds of flaws in software. The “Fuzz” report of the University of Wisconsin (Miller et al., 1995) showed that the reliability of the GNU and Linux software was better than that of commercial UNIX products. From all the above, the conclusion is that it is not certain in all cases that F/OSS is more secure than proprietary software, but the openness of the code is definitely a positive aspect.

The question of whether or not F/OSS is reliable was the cause for an interesting project on dependability, in the area of F/OSS. The project was conducted by DIRC (Interdisciplinary Research Collaboration in Dependability in Computer-Based Systems) in the United Kingdom. A definition of dependability exists on their web site: “Dependability is a systems (or "weakest-link") issue since virtually any aspect of a system, and of the means by which it was specified and designed, can cause a computer-based system to malfunction” (DIRC, 2002). The group that was appointed with the task of studying F/OSS organized an International Workshop on Open Source Development in order to gain better understanding of the F/OSS phenomenon. The outcomes of that study were that “ there is much variation among F/OSS products just as there is among proprietary software” and “that the general issues of F/OSS worth no further research, but the F/OSS development methodologies in

constructing software should not be avoided" (Arief et al., 2002).

In the workshop mentioned above, Bosio et al. (2002) presented a first attempt to develop a model for the reliability improvement in F/OSS projects. In that study, they initially used a simple probabilistic model to understand better the relationship between software processes and software dependability. The model takes into account the probability of a user to report a bug and to fix it at the same time. Although the model looks simple (its authors admit so), it is a first step towards understanding the process of bug reporting and fixing in F/OSS.

5. Usability

Usability as a computer term means the ease with which a user can learn to operate a machine. It represents the ease with which a user can input data to a machine (a desktop computer for example), operate the machine in order to process the data and understand the output of the machine to obtain some useful result. One of the most respectful researchers in usability, Nielsen (1993), describes usability as the ease of learning, efficiency of use, memorability, error frequency and severity. In the ISO 9126 (International Organization for Standardization, 1991) model Usability is defined as "A set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users".

Usability regarding F/OSS is another research issue and has been a matter of interest not only for the academic community, but also for F/OSS movement itself. Many people within the F/OSS community realize that usability must be a core issue for F/OSS if it wants to attract a critical mass. Raymond (1999) makes it clear that in the desktop market, what counts is the ergonomic design of the product and the user interface, crucial aspects for the average desktop user. In the same vein, another F/OSS pioneer raises usability as one of the important issues in F/OSS (Behlendorf, 1999). Feller & Fitzgerald (2000) states that the non-developers will focus not only on the availability of the source code, but mainly on the quality and the support of the product.

F/OSS usability seems to be no better than that of the proprietary software and suffers from the

same problems that the commercial software does. These problems stem from the inherent difference between developers and users. The community of F/OSS does not pay so much attention to the usability issues that they themselves do not experience (Nichols, Thomson & Yeates, 2001). Indeed there is substantial evidence that many of the developers that drive F/OSS development are in the upper class of the programming profession (Feller & Fitzgerald, 2002). These people cannot imagine how a system looks like for a novice user and how easy it is to operate it. Behlendorf (1999) identifies that end user applications which involve GUIs, like a desktop environment, are hard to write. He states that the main causes for that are the constant changes of a program concerning GUIs, its software is rather complex and most notably, because most programmers are not so good GUIs designers.

Moreover, the tailored nature of F/OSS development is an obstacle to taking usability into account. . Indeed F/OSS projects start as “a developers' personal itch” [10] (Raymond, 2002), i.e. developers make software to meet their own needs and other users’ needs. For example, at the time people wanted a web server and Apache was made. Developers and users in that case are essentially the same and the software depicts their own needs. Usually parties involved care for the software to meet their own demands of usability and do not take into consideration other users. This leads to a sophisticated product with a concrete and specific user interface, which is suitable for people with their own special skills. For example, many utilities concerning the Linux system do not have a modern user interface, they are command line tools, with many command line parameters to determine functionality or have a text/console based front end. Nowadays many graphical front ends have made their appearance, but for fine-tuning purposes the use of the command line in the Linux case is arduous but inevitable.

The two major Linux desktop environments, GNOME and KDE, have identified the problem of usability. Both have launched “Usability Projects” in order to investigate and improve the usability of

[10] Something that was rather true during the early days of F/OSS. Nowadays many projects have a good starting point and a framework. Moreover many companies (including IBM, SUN, SGI, etc) turn a portion of their software to open source.

their products. The GNOME Usability Project [http://developer.gnome.org/projects/gup/] has come up with a study (Smith et al., 2001), which was conducted in March 2001, by the usability experts of SUN's Human Computer Interaction staff. The study involved twelve people with experience on the use of computers, but with no computer science background. They used the GNOME 1.2.2 in order to perform various day-to-day user tasks, such as logging in, exploring the desktop, file management tasks, customization tasks and logging out, according to a scenario. The group of experts came up with various design recommendations to the GNOME people about their project. Additionally, recognizing the importance of the user interface of the applications that use the GNOME environment, GNOME has issued a full text of guidelines that the applications in the GNOME environment should follow (Benson et al., 2002). KDE and OpenOffice.org have also launched usability projects, with similar goals to those of the GNOME.

Ecklund, Feldman & Tromblay (2001) conducted another comparative usability study about F/OSS and proprietary office application. . The study compared the usability of Sun StarOffice's Calc[11] with Microsoft Office's 2000 Excel. The study was conducted for a course at Berkeley University. Their findings revealed that even though Excel outperformed Calc -with small differences, often statistically insignificant- 92% of the people expressed their willingness to switch to Calc, taking into account the fact that it is free. Everitt & Lederer (2001) conducted a similar study for the same course at Berkeley, but with Microsoft Word and StarOffice's Writer. Again, the results were comparative for both of the products, but the users found Word easier to use than Writer. Both of these studies are only a small evidence of assessment of the usability of F/OSS since, as mentioned, StarOffice has been developed by Sun as closed source software.

Another recent usability study, which directly compares F/OSS GUI and Windows XP, was conducted by a commercial company, relevantive AG (2003). In that study 60 test participants performed typical tasks, regarding desktop usage, on a Linux system equipped with the KDE 3.1.2

[11] We have to mention here that StarOffice was initially a closed source project by Sun and shifted recently to an F/OSS project under the name OpenOffice.org, continuing at the same time as proprietary software (free for educational use, not open source).

desktop. Another group of 20 participants performed identical tasks on a Windows XP system. The main result of this study was that the usability of Linux as a desktop system has been evaluated by users as nearly equal to Windows XP. A difficulty comparison between the Windows XP and the KDE is presented in Figure 2.

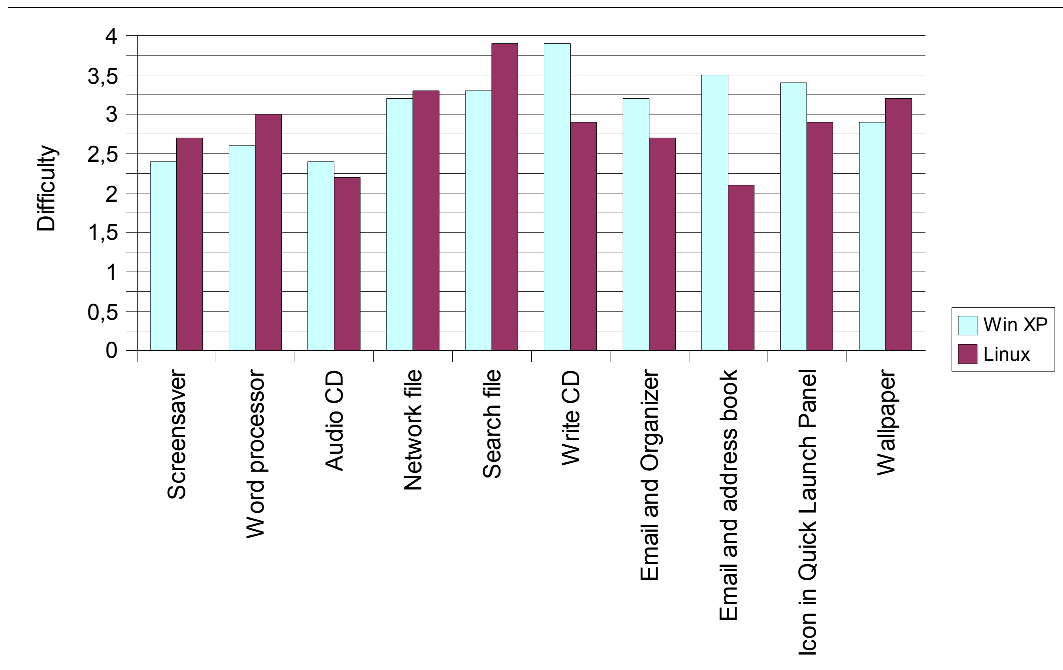
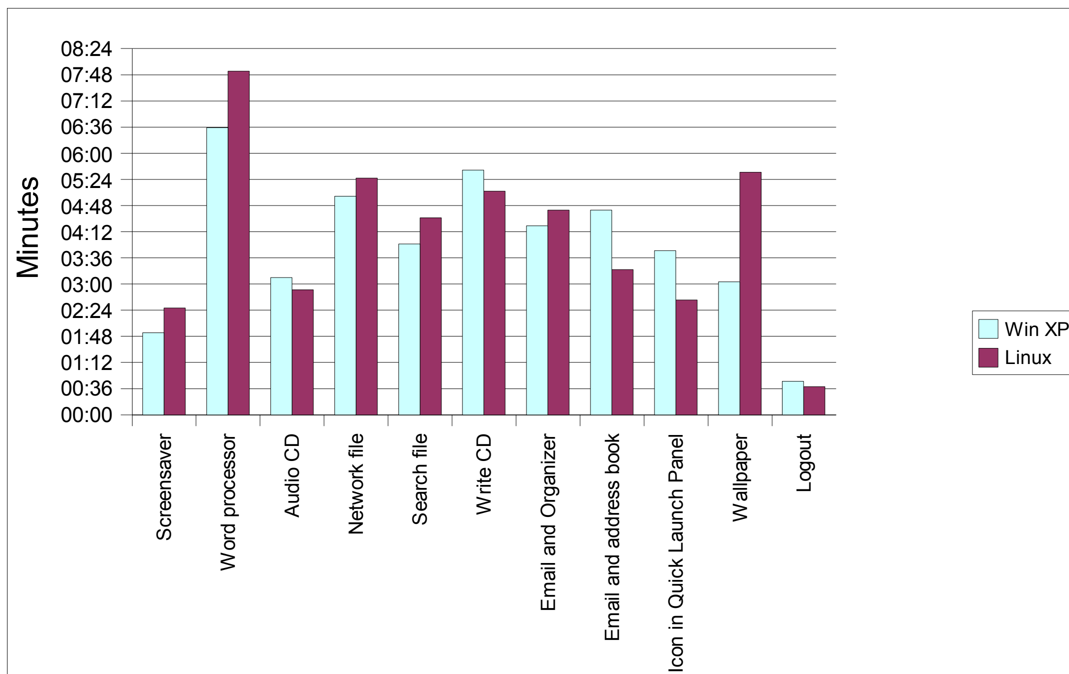


Figure 2. Average difficulty of tasks between Windows XP and KDE

The average time for a user to complete a specific task was in average only little behind Windows XP. The time needed by the users to complete a task in each one of the two systems is presented in Figure 3. The study also describes certain significant problems of the Linux system, such as partly missing clarity and structure of the desktop interface. However, the majority of the participants quite liked the system and needed a maximum of one week to acquire their previous level of competence on this system, thus making a transition to Linux look easier.

Figure 3. Average time needed per task



A good survey on usability of F/OSS is that of Nichols & Twindale (2003). In addition to the F/OSS usability findings already discussed, the authors try to find the causes of the problems in F/OSS usability. They argued that it is not only the F/OSS developers nature, which makes them to ignore usability, but it is also the lack of usability experts in the F/OSS development process, a fact that inhibits F/OSS from having a better usability than closed source software. It seems a foregone conclusion that the nature of F/OSS development is not suitable for designing for usability. Usability takes place before any actual coding starts and certainly this is not the case in F/OSS. Usability is a quality characteristic that needs to be planned from the first stages of the software development cycle (Ferre et al., 2001), (Scholtz & Shneiderman, 1999). In particular (Ferre et al., 2001) "Usability Engineering defines the target usability level in advance and ensures that the software developed reaches that level". High usability levels may be pursued through various usability engineering techniques, e.g. by including a user analysis task in the requirements analysis phase, by allowing for user-centered design or by building prototypes. There exists an obvious cause-effect relationship between usability-oriented techniques and perceived usability after product development. Usually, as mentioned earlier, in the case of the proprietary software, a prototype is made in order to explore its

usability and the main functionality is added later. This aspect of software development is not within the realms of F/OSS, a fact which leads F/OSS to follow ideas derived from proprietary software, regardless of whether they are the best or not. In addition, as it is underlined by Nichols, usability problems are harder to specify and distribute than functionality problems. Users rarely post usability problems and it is even more difficult for someone to take over these problems, for the coordinators to assign such problems to someone and even to modularize them.

At this point it is interesting to point out that Microsoft uses a development model that at some points resembles that of the F/OSS case. As Cusumano & Selby (1997) described, Microsoft uses, as the authors call it, the “sync-and-stabilize” development method. In the ‘Microsoft model’, instead of having a large team working together in a sequential manner (like the “waterfall” development model), developers split into many small parallel teams (three to eight developers each). These small teams work together in order to build large products, while they still have the freedom to evolve their designs and operate nearly autonomously. In addition, they continually synchronize what they are doing as autonomous teams and periodically stabilize the product in increments (milestones) rather that once at the end. These teams start with an initial set of features which they change during the development cycles. They also test features of the product as they build them, including bringing customers to try prototypes in the usability lab. The team managers monitor the whole process prioritizing features to be implemented during the next iteration, according to marketing needs, user satisfaction and the time it remains to deliver the final product.

Heretofore, this style of development resembles that of F/OSS. It could as well be argued that this “sync-and-stabilize” approach is kind of an agile method, but there are significant differences. One obvious difference is that Microsoft has certain schedules presupposing that the incremental versions of the product will be stabilized. For F/OSS this is not the case, since there are no strict release schedules. Furthermore, Microsoft uses its own usability labs; something that F/OSS has not the ease to do. Furthermore, usability is something that demands careful design before coding, which is rather difficult for F/OSS development.

One of the most important aspects of usability is localization or internationalization. The purpose of localization is to make software accessible to users who are not fluent in the language that the system uses for interaction. Software systems in most cases use English as their interaction language, discouraging not English fluent users from using that software. This part of usability is rather important for desktop application like GNOME, KDE or Mozilla. These projects have raised localization as one of their major issues and many people are working on the translation of their user interfaces and documentation. The localization of the application is an important factor for the dissemination of the F/OSS, a fact that has been acknowledged by many government bodies, which promote the localization of many F/OSS applications (for example the Greek General Secretariat for Research and Technology).

Another important factor of usability in F/OSS is the difficulty of installing some F/OSS products. Indeed many users find it difficult to migrate to Linux and the whole process of installation is sometimes rather non-user friendly (Nimh, 2003). Many Linux distributions expect the user to answer many technical questions and to pass strange parameters to the system. However, we have to mention that many companies have made huge efforts to simplify the whole installation process and they are continuously trying to make their product better. The lack of a standard installation process of the various F/OSS products is another problem and sometimes the existence of various packaging systems is confusing for novice users. Luckily, while typically there is no formal process, there exist enough documentation on the Internet about F/OSS products and there are hundreds of mailing lists, in which the participants try to help each other.

6. Maintainability

Maintainability has to do with the easiness of code modification. According to the ISO 9126 (International Organization for Standardization, 1991) model, maintainability is "A set of attributes that bear on the effort needed to make specified modifications (which may include corrections, improvements, or adoptions of software to environmental changes and changes in the requirements

and functional specifications)". Maintainability of F/OSS projects is a factor that was one of the first to be investigated by the F/OSS literature. This was done mainly because F/OSS development emphasizes on the maintainability of the software released. Making software source code available over the Internet allows developers from all over the world to contribute code, adding new functionality (parallel development) or improving present one and submitting bug fixes to the present release (parallel debugging). A part of these contributions are incorporated into the next release and the loop of release, code submission/bug fixing, incorporation of the submitted code into the current and new release is continued. This circular manner of F/OSS development implies essentially a series of frequent maintenance efforts for debugging existing functionality and adding new one to the system. These two forms of maintenance are known as corrective and perfective maintenance respectively. From the above comes the fact that maintainability is a crucial part of F/OSS development and thus has become a research topic in various studies.

One of these studies (Stamelos et al, 2002) measured 100 C programs – a total of 606095 lines of code- found in a Linux distribution, using a commercial code analysis tool, Logiscope® by Telelogic Tau. The tool examines the code according to various attributes affecting maintainability, such as testability, simplicity, readability and self-descriptiveness. For each one of these factors, a number is computed taking account of several metrics such as lines of code, McCabe's Cyclomatic Complexity (McCabe, 1976) and Halstead metrics (Halstead, 1975). The results are shown in Table 3. These results sound like the proverb of the half full, half empty glass of water. The fact that half of the modules were acceptable shows the high maintainability of the F/OSS. Conversely, the other 50 percent might show the bad maintainability aspect of F/OSS. Taken into account the massive parallel development and the peer review of the F/OSS development, these findings implies that further inspection is needed for these modules.

Modules characterization	***Mean***	***Standard Deviation***	***Median***
acceptable as is	50.18%	18.65%	48.37%
require Comments	30.95%	14.09%	31.83%
require further inspection	8.55%	8.50%	7.65%
require further testing	4.31%	4.14%	3.55%
completely rewritten	5.57%	10.73%	3.20%

Table 3. Component allocation (in %) in modules characterization

Schach et al. (2002) studied 365 versions of the Linux kernel. For every version of the kernel they measured the number of instances of common (global) coupling. The term coupling refers to the situation when two mutually dependent software modules interact with each other. Fewer interactions and dependences between the modules of the software can reduce the risk of a single module's fault badly affecting the others. Although there are various types of coupling, the authors decided to investigate global coupling, which happens when two modules share reference to the same global variable. As the authors underlined, it has been shown that coupling is related to fault proneness but has not been explicitly shown to be related to maintainability. However, they argue that, if a module is fault-prone then it will facilitate maintainability actions, which directly affect software maintainability. While their measurement shown that the number of the total lines of the Linux kernel grows linearly with the kernel's version number, the number of instances of common coupling grows exponentially with version number. The latter made the authors to come to the conclusion that if this situation continues, there will be a heavy dependency among Linux kernel's modules, and therefore it will be difficult to maintain it. If this is the case with other F/OSS products and F/OSS depends heavily on common coupling, it will prove to be a negative aspect of F/OSS.

In another study, the maintainability of the source of an open source product is compared directly with a closed source one (Samoladas et al., 2003). In that study, the Maintainability Index (MI) proposed by Carnegie Mellon's SEI (Software Engineering Institute, 2002) was used in order to gain insight into the evolution of the maintainability of five F/OSS systems. The reason behind the use of

the MI stem from concurrence by both the authors and McConnell's that F/OSS should conform to such standards and metrics. The results of the study show that F/OSS maintainability is no worse than that of closed source software. In direct comparison, F/OSS was found to be better than closed source software. A summary of the results of this study are shown in Figure 4.

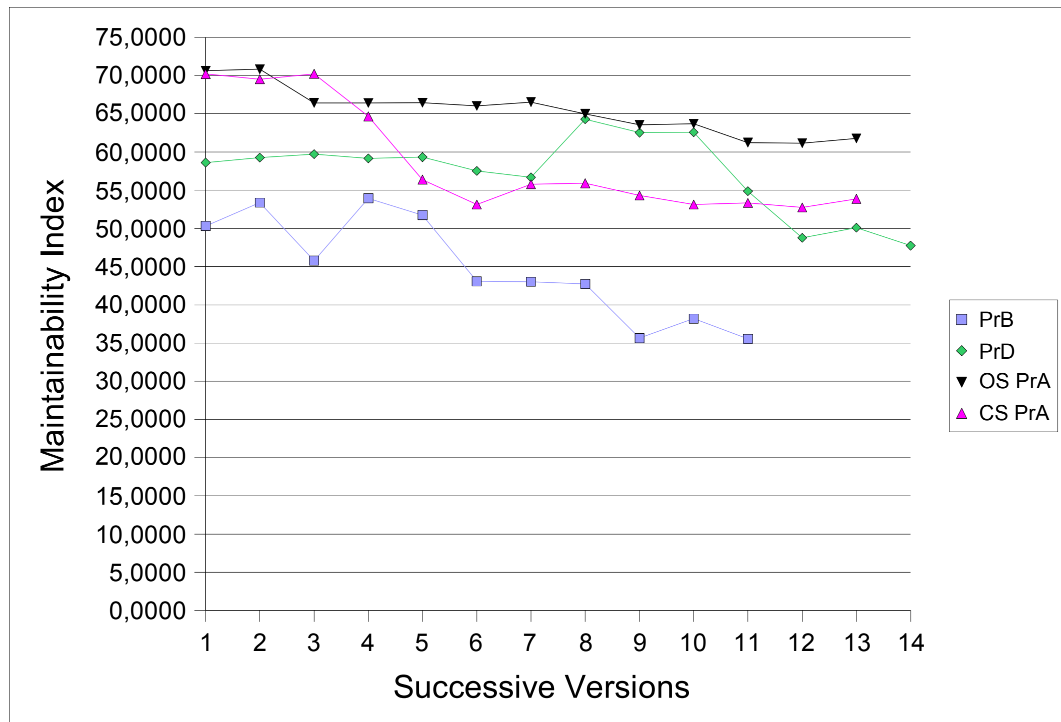


Figure 4. MI figures for three F/OSS products (PrB, PrC and OS PrA) and one closed source (CS PrA). Note that OS PrA and CS PrB perform the same functionality and they are directly compared

The same authors made another study of the maintainability of an open source ERP/CRM system, using a commercial tool (Samoladas & Stamelos, 2003). This system was written in Java and includes large portions of code from other F/OSS systems. The results, according to the quality standard denoted by the commercial tool (Logiscope®), are shown in Table 4. The table depicts the percentile of the application classes . It is obvious that these results suggest that the quality of the product is higher than the average. To a large extent, it is worth mentioning here that the maintainability or the code quality of the product itself is not a sufficient reason to suggest that the

specific ERP/CRM system outperforms others, because functionality, which is a critical parameter in such systems, was not considered in the design of the study.

Factor	*Excellent*	*Good*	*Fair*	*Poor*
Maintainability	13%	51%	35%	1%
Realizability	75%	25%	0%	0%
Changeability	47%	21%	0%	32%
Stability	80%	17%	0%	4%
Testability	17%	47%	32%	3%

Table 4. Compiere class results

7. Portability

Portability in software is an attribute that has to do mainly with platforms and machine dependence. It expresses the ease of transferring an existing system, running on a specific machine, to another machine with a different configuration. ISO/IEC 9126 (International Organization for Standardization, 1991) describes portability as "A set of attributes that bear on the ability of software to be transferred from one environment to another (including the organizational, hardware, or software environment)". From its early days, portability has been a central issue in F/OSS development. Various F/OSS systems have as first priority the ability of their software to be used on platforms with different architectures. Here, we have to stress one important fact, which originates from the nature of F/OSS, and helps portability, namely the availability of the source code of the destination software. If the source code is available, then it is possible for the potential developer to port an existing F/OSS application to a different platform than the one it was originally designed for.

Perhaps the most famous F/OSS, the Linux kernel, has been ported to various CPU architectures other than its original one, the x86. Linux kernel based systems exist for PowerPCs, Alphas, SPARCs along with some other, sometimes "exotic" architectures (Kernel.Org Organization, Inc, 2003). What helps Linux to be so portable is mainly the fact that, like many other UNIX like systems, it is written in C, a third generation language, and not in a machine specific machine language. This means that, with some effort, Linux is portable to those systems where a C compiler exists. The compiler that Linux uses is the GNU gcc. This leads to the conclusion that for every

machine there exists a port of the gcc compiler, Linux is portable (Torvalds, 1999). This portability feature is stated on the main kernel site (http://www.kernel.org) as “Linux is easily portable to most general-purpose 32- or 64-bit architectures, as long as they have a paged memory management unit (PMMU) and a port of the GNU C compiler (gcc)”. Moreover, the C compiler that the kernel uses, belongs to the gcc family, the GNU Compiler Collection. This collection, which is free software, contains front ends for numerous languages, including C, C++, Java, and Fortran and can be ported to different architectures. F/OSS applications that utilize this compiler collection can be ported to different platforms with little or no effort.

Another factor that made Linux portable is its own design. The creator/creators of Linux have tried to make its design as clean as possible in order to make it much portable. . This was achieved with the usage of loadable kernel modules. The existence of loadable kernel modules allows much of hardware specific functionality to be separate of the main kernel making the system more modular. These modules make it possible for the system to be installed on a huge variety of machines, with different hardware configuration, as long as a module for that hardware exists. For example, if someone tries to compile the Linux kernel, he will have to configure the kernel modules for the specific hardware that the compilation is going to take place in. In the configuration, he finds a vast selection of modules for every kind of hardware, file systems, code pages and so on. For example, he may configure which file systems modules he wants to be accessed by his system (there are modules for EXT2, FAT32, NTFS, etc). Two factors about the Linux kernel portability, namely, the existence of a port of the gcc and the ability to load kernel modules resulted in the presence of Linux based machines on different platforms. For example, the Debian GNU/Linux is released for ten different ports, along with other ports without the Linux kernel.

Another aspect that shows the importance of portability in the F/OSS world is the adherence of many F/OSS software to well known standards related to portability. One of the most popular standards of that kind is the IEEE POSIX (Portable Operating System Interface) standard. Indeed, Linux kernel tries to stick as much as possible to that standard in order to allow other applications

that utilize it to be portable[12]. Developing an application, following the POSIX API standard, it is possible to compile its source code, and to run it, on different platforms[13]. In addition, the Linux kernel includes much of the functionality of the Single UNIX Specification, Version 2 (The Open Group, 2002). The Single UNIX Specification is an attempt to integrate various standards, including the IEEE POSIX and ISO C, in order to provide a unified application programming interface. UNIX, from which Linux semantics was derived, was designed to be portable (Ritchie & Thompson, 1974) and this fact helps Linux itself to be portable. It is obvious that Linux tries to follow this specification in favor of the portability of the applications written for its system. One other important category of free operating systems, namely the BSD family (Free/Open/NetBSD), also follows the majority of the standards that Linux uses, thus making porting of the applications among Linux and the BSD family easy. The portability among the various Linux distributions is also another aspect of portability. Unfortunately many distributions add some features that prevent sometimes applications statically compiled for one distribution to run on another. This lead to the creation of the Linux Standard Base [http://www.linuxbase.org], an initiative that has as a goal to promote portability among the various Linux distributions.

8. Efficiency

Efficiency is a matter that has not been investigated in the area of F/OSS. There are no academic studies[14] which explore the efficiency in F/OSS. ISO/IEC 9126 identifies efficiency as "A set of attributes that bear on the relationship between the software's performance and the amount of resources used under stated conditions". In fact the term efficiency refers to the time and resource behavior of the software. There are, of course, some benchmarks from various popular magazines or some web sites, showing benchmark data about the performance of Linux based or Windows based servers, but sometimes the rationale of the benchmarking test is in most times, quite subjective. This

[12] Windows NT is also POSIX compliant at the systems API level.

[13] We have to mention here that POSIX compliance may not be sufficient to enable portability of applications with GUI dependencies between UNIX (or NT) systems

[14] At least not published on well known journals

leads to the fact that various tests favor Linux as the most efficient solution, in terms of time and resources behavior, while others favor Windows. Expediently, new research studies are needed in which, objectively and scientifically, efficiency of F/OSS is studied. This lack of research has been identified by a previous European Commission's workshop about F/OSS and has posed various research questions about efficiency in the context of F/OSS (Ghosh, 2002).

9. Conclusions – Further Research

The overall results from the facts stated above seem to indicate an acceptable level of F/OSS quality, although there is more to be done. Although in certain areas, F/OSS seems to do better than proprietary software, there are things to be improved and further expanded, so that we avoid typical problems that arise from the software business as we know it. Regarding the various quality factors mentioned above there are certain areas and issues to be underlined and further investigated.

As far as functionality is concerned, F/OSS has proved its suitability and applicability. It is obvious that in areas like Internet applications, operating system tools and programming languages, including languages and compilers for them, F/OSS is widely used and in certain spheres has outperformed proprietary software. On the contrary, F/OSS has not shown, until now, significant results in the area of the systems where vertical requirements are needed. The successful applications mentioned above implement widely accepted requirements that do not demand a tailored and more specific requirement analysis for their development. Yet still, great volumes of IT applications are developed to address specific and not broad user requirements. Further investigation is needed in order to explore how F/OSS can be effective and suitable for this kind of development. It is rather important to answer whether or not F/OSS is a viable solution for such types of application. That is to say: Is F/OSS the “Holy Grail” only for systems with horizontal requirements?

In the context of the second quality factor, reliability, again F/OSS seems to do a good job, but there is terrain for improvement. The “many eyeballs” rule has to be further explored and clarified. Quantitative studies in this area are certainly a good step towards better understandability of the

process through which F/OSS achieves reliability. Some efforts have already been done, for modeling and predicting reliability in F/OSS, but certainly more is needed (Bosio et al., 2002). Reliability in conjunction with productivity is another subject that worth further research. Productivity, alone is already studied (Koch & Schneider, 2000), but certainly it has to be related with various aspects of quality of the F/OSS development. Security as another reliability factor also calls for further exploration, taking into account its high importance nowadays. It is questioned of whether or not peer review and openness are in favor of security, although results seem to prove that eventually they lead to more secure software.

The next quality factor, usability, is maybe the most important for the mainstream success of F/OSS. For F/OSS to be widely used, usability should be the central focus of future development. The efforts must begin from the beginning of an F/OSS product, i.e. the stage when the initial idea is conceived. The installation process has to be further simplified in order to facilitate a novice user to install and use an F/OSS product, especially an F/OSS operating system. A "universal installer" may sound like a good step for the solution of the problem, but it has to be subjected to rigorous test and evaluation procedures. Nichols & Twindale (2003) provides some useful suggestions and directions for improving F/OSS usability. These suggestions include commercial along with technological approaches in evaluation of usability of F/OSS and in some cases in the development of user interfaces. Other approaches could involve end users in order to perform usability tests and then send the results back to the developers of the software. Furthermore, Nichols & Twindale (2003) suggests involvement of academia in F/OSS development, fragmentation of usability analysis and design, involvement of experts and convincing of F/OSS developers for the need of usability.

Maintainability is a quality factor that has been investigated to some extend. The results seem to indicate an acceptable level of maintainability, without showing large differences with respect to similar studies from the proprietary software area. These efforts are again only the beginning and further empirical research is needed. Effort and productivity have to be related to and studied along with maintainability. The evolution of maintainability of F/OSS has to be investigated in order for

prediction and estimation models to be produced (e.g. by identifying error prone modules). Statistical methods like discriminant analysis are a tool towards the achievement of such models. Moreover, a comparison of the maintainability of proprietary software versus F/OSS is a good indicator of how F/OSS maintainability performs. Furthermore, the trend towards object oriented programming and methodologies make it important for researchers to exploit object-oriented metrics. These measurements should include not only conventional methods, but it is also important to consider heuristics (Riel, 1996). Reusability, another acclaimed characteristic of F/OSS, has to be also further investigated.

The next quality factor we have studied, portability, seems to do well in the case of F/OSS, since it is one of the first priorities of F/OSS. The conformance of the majority of F/OSS applications to a standard (like POSIX) related to portability has resulted in high portability of F/OSS. The F/OSS most well known story, Linux kernel, seems to be highly portable mainly due to the fact that its main compiler, gcc, is itself portable. Additionally, the loadable module facility of the Linux kernel offers high installability in various environments as well as platforms. Further development of standards for F/OSS is one directive for future research in F/OSS.

The last quality factor, according to ISO/IEC 9126, efficiency, has not been studied as mentioned earlier and the main research directive should include comparison of F/OSS and proprietary software application efficiency and benchmarking in a way that conforms to international widely accepted standards, so as to produce objective results.

As a final conclusion, Free/Open Source Software quality is an open issue and Free/Open Source Software should continue striving for even better quality levels in order to outperform traditional, closed source development.

11. References

Arief, B., Bosio, D., Gacek, C., Rouncefield, M. (2002). Dependability issues in open source software DIRC Project activity 5 final report. (Technical Report CS-TR-760). Newcastle upon

Tyne: Department of Computing Science, University of Newcastle upon Tyne

Behlendorf, B. (1999). Open source as a business strategy. In Chris DiBona, Sam Ockman, & Mark Stone (Eds). Open Sources: Voices from the Open Source Revolution. Cambridge, MA: O'Reilly and Associates

Benson, C., Elman, A., Nickell, S., & Robertson, C.Z. (2002). GNOME Human Interface Guidelines (1.0), Retrieved March 18, 2003 from: http://developer.gnome.org/projects/gup/hig/1.0/

Bosio, D., Littlewood, B., Strigini, L. & Newby, M.J., (2002). Advantages of open source process for reliability: clarifying the issues. In C. Gacek & B. Arief (Eds.) Proceedings of the open source software development workshop (pp 40-47). Newcastle upon Tyne, UK

Chou, A., Yang, J., Chelf, B., Hallem, S. & Engler, D. (2001). An empirical study of operating system errors. Proceedings of the eighteenth ACM symposium on operating systems principles (pp 73-88), Banff, Alberta, Canada

Coverpages (2003, April 16). Patents and open standards. Retrieved March 14, 2003 from: http://xml.coverpages.org/patents.html

Cusumano, M., Selby, R.W. (1997). How Microsoft build software. Communications of the ACM, 40, 6, 53-61

DIRC, (2002, December 11). A weakest link issue. Retrieved March 14, 2003 from: http://www.dirc.org.uk/overview/what.html

Ecklund, S., Feldman, M., & Trombley, M. (2001, December 12). StarOffice Calc v. MS Excel: Improving the usability of an open source spreadsheet application, Retrieved March 18, 2003 from : http://www.sims.berkeley.edu/courses/is271/f01/projects/StarCalc/

Everitt, K., & Lederer, S. (2001, December). A usability comparison Sun StarOffice Writer 5.2 vs. Microsoft Word 2000, Retrieved March 18, 2003 from: http://www.sims.berkeley.edu/courses/is271/f01/projects/WordStar/

Feller, J., & Fitzgerald, B. (2002) Understanding Open Source Software Development, London: Addison-Wesley

Feller, J., Fitzgerald, B. (2000). Framework analysis of the open source software development paradigm. Proceedings of 21st annual international conference on information systems

Fenton N.E., & Pfleeger, S.L. (1997). Software metrics: a rigorous and practical approach. 2nd ed. London: International Thomson Computer Press.

Ferre, X., Juristo, N., Windl, H., Constantine, L. (2001). Usability Basics for Software Developers. IEEE Software, 19, (1), 22-29

Freericks, C. (2001). Open source standards on software process: A practical approach. IEEE Communications Magazine, 39, 4, 116-123

Ghosh, R.A. (2002). FLOSS. Workshop on Advancing the Research Agenda on Free / Open Source Software. International Institute of Infonomics, University of Maastricht, The Netherlands.

Halstead, M. (1975) Elements of Software Science. Holland: Elsevier

Hissam, S.A., Plakosh, D., & Weinstock, C. (2002). Trust and vulnerability in open source software. IEE Proceedings – Software, 149, 1, 47-51.

International Organization for Standardization. (1991). Information technology-Software product evaluation: Quality characteristics and guidelines for their use. ISO/IEC IS 9126. Geneva: ISO

Kernel.Org Organization, Inc. (2003) The Linux kernel archive. Retrieved March 14, 2003 from: http://www.kernel.org

Kitchenham, B., & Pfleeger, S.L. (1996). Software quality: The elusive target. IEEE Software, 13, 1, 12-21

Koch, S., & Schneider, G. (2000). Results from software engineering research into open source development projects using public data. Diskussionspapiere zum Taetigkeitsfeld Informationsverabeitung und Informationswirstschaft, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22, Wirtschaftuniversitaet Wien.

McCabe, T. (1976). A complexity measure. IEEE Transactions on Software Engineering, 2, 4, 308-320

McConnell, S. (1999). Open source methodology: Ready for prime time?. IEEE Software, 16, 4, 6-8

Nimh, T.D. (2003, April 4). Migrating to Linux not easy for Windows users. Linux World. Retrieved April 10, 2003 from: http://www.linuxworld.com/2003/0401.tsu-p4.html

Miller et al. (1995). Fuzz Revisited: A Reexamination of the Reliability of Unix Utilities. Retrieved March 14, 2003 from: http://www.cs.wisc.edu/~bart/fuzz/fuzz.html

Mockus, A., Fielding, R.T., & Herbsleb, J.D. (2002). Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11, 3, 309-346

Netcraft (2003). The NETCRAFT web server survey. Retrieved March 14, 2003 from: http://netcraft.net/survey/

Nichols, D. M., & Twidale, M. B. (2003, January). The Usability of Open Source Software. First Monday, 8, 1. Retrieved March 18, 2003 from: http://firstmonday.org/issues/issue8_1/nichols/index.html

Nichols, D.M., Thomson, K., & Yeates, S.A. (2001). Usability and open-source software. In Kemp, E., Phillips, C., Kinshuk & Haynes, J. (Eds.), Proceedings of the Symposium on Computer Human Interaction (49-54). Palmerston North, New Zealand: ACM SIG-CHI New Zealand

Nielsen, J. (1993). Usability Engineering. Boston: Academic Press

Raymond, E.R..(1999). The revenge of the hackers. In Chris DiBona, Sam Ockman, & Mark Stone (Eds). Open Sources: Voices from the Open Source Revolution. Cambridge, MA: O'Reilly and Associates

Raymond, E.S. (2002, September 27). The cathedral and the bazaar. Retrieved March 01, 2003 from: http://www.catb.org/~esr/writings/cathedral-bazaar/

Reasoning. (2003, August 10). The Reasoning Library. Retrieved August 10, 2003 from: http://www.reasoning.com/library.html

Relevantive AG. (2003, August 09). The Linux usability report. Retrieved August 09, 2003 from: http://www.relevantive.de/Linux_e.html

Riel, A. J. (1996). Object Oriented Design Heuristics. Addison-Wesley

Ritchie, D., Thompson, K. (1974). The UNIX operating System. Communications of the ACM, 17, 7, 365-375

Samoladas, I., Stamelos, I. (2003). Systematically assessing the quality of an open source ERP/CRM system. In Proceedings 1st International Conference for Mathematics and Informatics for Industry, MATHIIND 2003, Thessaloniki, Greece

Samoladas, I., Stamelos, I., Angelis, L., & Oikonomou, A. (2003). Open source should strive for even better maintainability. Forthcoming in Communications of the ACM

Schach, S.R., Jin, B., Wright D.R., Heller, G.Z., & Offutt, A.J. (2002). Maintainability of the Linux Kernel. IEE Proceedings – Software, 149, 1, 18-23

Schmidt, D.C., & Porter, A. (2001) Leveraging open source communities to improve the quality & performance of open source software. In Feller, J., Fitzgerald, B. & van der Hoek, A. (Eds). Making sense of the bazaar: Proceedings of the 1st workshop on open source software engineering

Scholtz, J. & Shneiderman, B.(1999). Introduction to Special Issue on Usability Engineering. Empirical Software Engineering, 4, (1), 5-10

Smith, S., Engen, D., Mankoski, A., Frishberg, N., Pederson, N., & Benson, C. (2001, July) GNOME Usability Study Report, Retrieved March 18, 2003 from: http://developer.gnome.org/projects/gup/ut1_report/report_main.html

Software Engineering Institute. (2002). Maintainability index technique for measuring program maintainability. SEI. Retrieved March 24, 2003 from: http://www.sei.cmu.edu/str/descriptions/mitmpm_body.html

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G.L. (2002). Code Quality Analysis in OSS Development. Information Systems Journal, 12, 1, 43-60

The Open Group (2002, July 22). What about all those "Flavors"?. Retrieved March 14, 2003 from: http://www.unix-systems.org/what_is_unix/flavors_of_unix.html

Torvalds, L.(1999). The Linux edge. In Chris DiBona, Sam Ockman, & Mark Stone (Eds). Open Sources: Voices from the Open Source Revolution. Cambridge, MA: O'Reilly and Associates

Vixie, P. (1999). Software Engineering. In Chris DiBona, Sam Ockman, & Mark Stone (Eds). Open Sources: Voices from the Open Source Revolution. Cambridge, MA: O'Reilly and Associates

Wheeler, D. (2003, March 3). Is open source good for security. Retrieved March 18, 2003 from: http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/open-source-security.html

Witten, B., Landwehr, C., & Caloyannides, M. (2001). Does open source improve security?. IEEE Software, 18, 5, 57-61