Exploring the Quality of Free/Open Source Software: a Case Study on an ERP/CRM System

Ioannis Samoladas, Stamatia Bibi, Ioannis Stamelos and Georgios L. Bleris

Department of Informatics, Aristotle University
54124 Thessaloniki, Greece
Email: {ioansam,sbibi,stamelos,bleris}@csd.auth.gr

Abstract. Free/Open Source Software (F/OSS) is an alternative way of software development. Software is produced by a community of volunteer developers over the Internet, its source code is available to everyone and can be freely used, modified and distributed at no cost. Maintainability is considered the most important quality factor for such type of software. Recently, F/OSS ERP/CRM products have appeared, and an increasing number of installations is observed worldwide. Proponents of F/OSS claim that their software quality is better than proprietary software quality. Various empirical studies have been conducted to examine these claims, but no study has considered F/OSS products in the ERP/CRM field up to now. This paper presents an analysis of the source code of Compiere, the most successful F/OSS ERP/CRM system written in Java, producing an appraisal of its quality, using a predefined quality model. The internal quality attributes examined are stability, analyzability, testability and changeability. By observing the values of these attributes, assumptions about the degree of the maintainability of the system may be obtained. Moreover an analysis based on multi-organizational productivity data, indicates what would be the cost for producing Compiere in close source mode. The findings of the study, provide evidence provide evidence that Compiere has an above-average level of internal quality, leading to a good level of maintainability as well. However, further empirical research is needed to assess the external quality of Compiere (e.g. functionality and usability)

1 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 [2]. 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" [11]. 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 are incorporated into the source code of the product and the next version of the software is released. 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 spreaded worldwide, contributing over the Internet, and they don't know each other.

In many cases F/OSS seems to have solved the main problems of software engineering, i.e. development speed (late delivery of the product), cost (budget deficits) and low quality of the delivered product. Some of the most successful and well known F/OSS products are the Linux kernel, the Apache web server and BIND (Berkeley Internet Name Daemon). In fact Apache is used by the 62.7 % of the web server installations [8], while the BIND is by far the dominant – a "category killer" - name resolution server. Other F/OSS include applications such as compilers, for a great variety of programming languages, various network applications such as mail servers, database management systems and more. For a little evidence about how many F/OSS exist there are 61,000 applications registered at one of the biggest F/OSS host, Sourceforge.net[2], by the time this was written (April 2003).

Of course F/OSS has both advantages and disadvantages. Indeed, in many cases, as mentioned above, F/OSS seems to solve the main problems of software engineering. The main advantage of F/OSS is the existence of a large pool of developers, which are at the same time users. This fact along with the free distribution of the source code facilitates true peer review of the code and parallel debugging. These two give a large boost on the software development resulting in a more effective software production, with true field testing and fast bug fixing. It is worth mentioning here the famous postulation by Raymond [11] that describes this situation: "Given enough

[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

[2] Sourceforge.net provides hosting for F/OSS projects offering a set of tools to their coordinators in order to facilitate the F/OSS development process (bug tracking systems, CVS - Control Versioning System- repository and more)

eyeballs, all bugs are shallow". Other advantages include extensive interoperability of F/OSS products and portability in many platforms and environments. On the other hand, disadvantages of F/OSS include poor usability of F/OSS and lack of formal documentation or technical support [9]. However the lack of documentation is replaced by the existence of hundreds of mailing lists of F/OSS products users, where someone can ask questions and usually get answers.

The majority of the F/OSS applications stem from 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 F/OSS engineering was involved mainly in projects that had to do with networks or operating systems applications [1], [10]. Almost all the successful cases of F/OSS are coming from the areas mentioned above and not from the desktop or GUI applications category[3]. [1] suggested 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. 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 for applications with well defined (pre-defined) requirements. It can be said that F/OSS seems better suited to horizontal requirements, addressing the needs of broad user categories [7]. McConnell explained this lack of requirements 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". An open question is whether or not F/OSS is appropriate to vertical requirements, such as ERP/CRM systems, that require a lot of customization.

2 ERP/CRM Systems

ERP systems are for quite long time in the market. They are in fact a combination of management practices and Information Technology (IT), with the letter to be integrated into the company's information flow infrastructure. ERP automates the various practices of a company and connects the various departments of it. What ERP mainly does is to break any information barriers between the departments of a company, allowing information sharing and thus helping a company to run effectively. For example when a sale is taken place, automatically the accounting department is informed about the sale, the inventory is being updated and if there the remaining stock is lower than a limit, the system automatically arranges for the additional stock to be ordered. The implementation of an ERP system demands the existence of a central database, where all the information of the company is stored. CRM systems are relatively a new idea. The main purpose of a CRM system is to help a company to manage the needs and problems of their customers more effectively. Usually using the same central database with the ERP system of the company (CRM systems are, in most of the cases, a module of an ERP system) and various intelligent techniques (data mining for example) a CRM system can produce buying trends and models that

[3] A simple visit at Sourceforge.net can justify that. Although, we have to admit that there are famous GUI applications, like the GNOME, the KDE and the GIMP, an image manipulation program

help the marketing policy of the company. From its nature, ERP/CRM systems have vertical requirements. Every company has different practices and workflow and the installation of an ERP/CRM is customized to that company in order to meet its needs.

3 A Case of a Free/Open Source Software ERP/CRM System

As mentioned earlier, the majority of the available F/OSS projects lie in the area of network or operating systems applications and not from systems that usually require a more detailed requirements document, like the ERP/CRM systems. However, such systems exist and it is worth to study them, since they seem not to be suitable to be developed under the F/OSS paradigm. An example of a F/OSS ERP/CRM system is Compiere [http://www.compiere.org]. Compiere is suitable for small-medium enterprises and it offers automation for their accounting and an effective inventory and customer management and additionally it offers a web store module, helping a company to have an on line shop. For a more detailed presentation on what Compiere does, someone can access its website mentioned earlier.

Compiere itself is written mostly in Java, although it contains a lot of code in PL/SQL and for its web store functions, portions of JavaScript. For the time being, Compiere requires Oracle Database for its database management system, because of the usage of embedded transactions (especially for Oracle), but there is a huge effort to make Compiere database independent. Currently apart from maintaining its own web site, Compiere development process is hosted on Sourceforge.net. By the time this paper was written (April 2003) Compiere has reached the number of 495,000 downloads and it is usually among the top ten of the most active and popular software products of the Sourceforge.net repository. It has been started in August 2001 and currently it involves 36 developers, with its coordinator to be an experienced developer in the area of the ERP/CRM systems. Apart from the source code developed exclusively for Compiere, the latter contains portions of source code from other F/OSS products like the Apache Tomcat.

4. Quality and Measurement of the Source Code

Software is a product and like any other product it has a level of quality. This quality, software quality, can be separated into internal and external quality, with the former to be related at some extends to the latter. Thus measuring the internal quality of a software product can give us serious evidence about the level of the external quality. For instance, characteristics of external quality such as the maintainability testability and changeability can be figured out if we measure internal quality. Usually each one of these characteristics is the outcome of some formula or an aggregation model, which in turn reside on quantities like lines of code, the density of the comments in the code and others that can be directly measured from the source code. Therefore, by measuring the source code of an application and aggregating the results with the help of a model someone can assess software internal and external quality.

For our case study we have used Logiscope® by Telelogic Tau™. Logiscope® provides a set of tools which perform, automatically, code coverage analysis, code

measurement, code checking against predefined programming rules and quality assessment which help developers to identify error prune modules of the software being constructed. Additionally Logiscope® is capable of assessing software quality according to ISO/IEC 9126 [5] and some of the ISO-9001 quality characteristics. It is worth mentioning here that Logiscope® has sold over 10,000 licenses all over the world and it is used by some of the major companies in the areas of telecommunication, aerospace and defense, automotive and transport, energy and process control.

Logiscope® can perform code analysis for software written in Ada, C, C++ and Java. In our study we have used Logiscope® Java Audit tool, which performs source code measurement in order to perform quality analysis of a software product and produce a quality report. Logiscope® quality analysis is separated into three levels: Application Level, Class Level and Function Level. The Application Level provides information about the whole application examined, the Class Level for the classes (as in Object Oriented Programming Methodology), which are in the source code and finally, Function Level about the functions (or methods) found in the source code. Each one of these three levels is further separated into detailed metrics, which in turn are further analyzed into other ones. These are either distinct metrics or the result of a formula consisting of metrics measured from the source code directly. Examples of such low level metrics are the Lines of Code (LOC), McCabe's cyclomatic complexity [6], Halstead effort metrics [4] and various object oriented metrics such as the depth of the inheritance tree. For a more extensive reference to software metrics and measurement, someone can refer to Fenton and Pfleeger [3].

5 Results of the Measurement of Compiere

Since now there have been very few similar studies concerning the quality of F/OSS and its measurement (for example: Stamelos et al., [14], Samoladas et al, [12] or Schach et al., [13]), but none of these studied a F/OSS ERP/CRM system. For the purposes of our study we have downloaded all the files of the source code of the product, as they exist in the CVS repository of the product. In total we have downloaded 594 Java files (or 101,490 lines of code). These files were parsed in Logiscope® and the analysis of this parsing produced the results presented below. As mentioned earlier, among those files we have downloaded there are parts from other F/OSS products[4]. We present both measurement results with and without those files in order to gain better understanding of the quality of the product.

As far as the Application Level is concerned, according to the results, the Maintainability of Compiere appears to be GOOD. Without the portion of the included code Maintainability is EXCELLENT. Maintainability is set to be the sum of CHANGEABILITY and TESTABILITY which in the first case (whole code) are EXCELLENT and GOOD respectively, while in the second case (without the included code) both are EXCELLENT.

In the section below we present the overall results for the application Classes and Functions along with some of the results of the important metrics. We do not go into deeper analysis since the overall results allow for a safe assessment of the overall

[4] From now on, referred as “included code”

quality of Compiere. We present the overall results for the whole code of the product, while for results of measurement without the included code are presented in parentheses:

	Excellent	**Good**	**Fair**	**Poor**
Maintainability	13% (15%)	51% (60%)	35% (25%)	1% (0%)
Analyzability	75% (89%)	25% (11%)	0% (0%)	0% (0%)
Changeability	47% (56%)	21% (23%)	0% (0%)	32% (22%)
Stability	80% (80%)	17% (18%)	0% (0%)	4% (3%)
Testability	17% (19%)	47% (54%)	32% (23%)	3% (4%)

Table 1. Application Classes Results of the whole code

	Excellent	**Good**	**Fair**	**Poor**
Maintainability	73% (63%)	23% (31%)	3% (4%)	1% (2%)
Analyzability	81% (75%)	13% (16%)	4% (6%)	2% (3%)
Testability	84% (14%)	14% (19%)	2% (3%)	1% (1%)

Table 2. Application Function Results of the whole code

Below we present some results of the most important source code metrics. Again in parentheses are the results for the product without the included code:

Metric Name	**Max**	**Min**	**Value**
Average of the VG[5] of the application's functions	5	1	2.17 (2.67)
Number of levels in the inheritance graph	4	1	7 (4)

Table 3. Application Metrics Results of the whole code

Metric Name	**Max**	**Min**	**Classes out of limits (%)**
Class comments frequency	+∞	0.20	1.55% (1.17%)
Number of base classes	3	0	24.03% (10.53%)
Number of children	2	0	4.03% (3.12%)

Table 4. Classes Metrics Results of the whole code

Metric Name	**Max**	**Min**	**Functions out of limits (%)**
Comments frequency	+∞	0.50	17.88% (22.52%)
Number of statements	20	1	10.31% (13.33%)
Cyclomatic complexity	10	0	4.54% (5.06%)
Number of out statements	1	0	12.50% (17.58%)
Number of PATHS	60	1	4.56% (4.72%)

Table 5. Functions Metrics Results of the whole code

[5] VG stands for cyclomatic complexity

6 Measuring the Productivity of Compiere

It is important to have an idea of the cost that this open-source system would have in the case that it was developed under different circumstances, i.e by a company. The effort needed to implement such an application under certain time and budget constraints is a matter of interest of all the involved parties. Indicative of the effort and the cost is productivity, which is the amount of work per unit of time.

In order to estimate productivity of Compiere, some projects with similar attributes will be taken into consideration from ISBSG data set. ISBSG [http://www.isbsg.org.au] is a publicly available multi-organizational data set. This dataset was selected because it contained data from many companies, the projects were relatively recent, (1990 and later) and a variety of methods and tools were used for their implementation. As a consequence, projects developed in various environments and with different techniques will provide a reliable estimation for comparison purposes.

Seventeen projects relevant to our application were chosen such as projects concerning the development of Management Information Systems, Office Information Systems and Transaction Production Systems. All these systems are used for the distribution of information in an organization and the automation of services. The distribution of the project productivity values measured in LOC per work hours is shown in Figure 1:

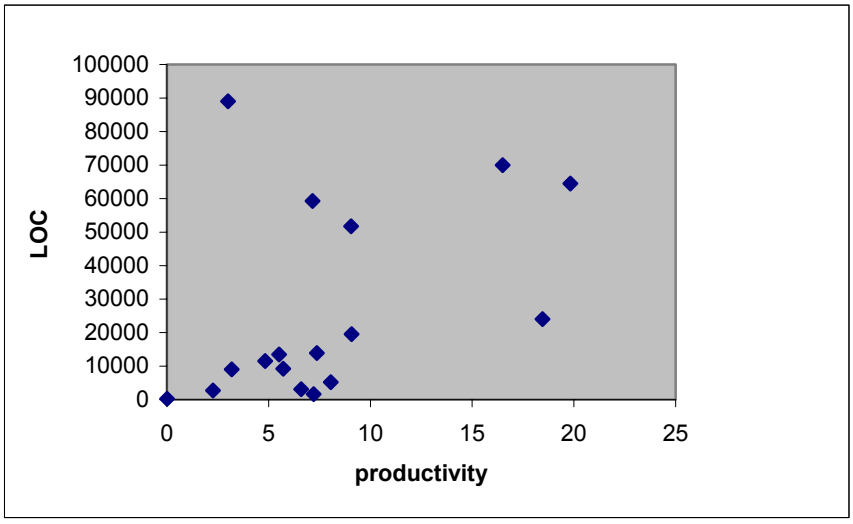

Fig. 1. Productivity per Lines Of Code

From the plot it is obvious that, in most projects, productivity concentrates around small values, 5-10 lines of code /hour.

A productivity value indicative of the ERP/CRM system coming from the average values of the projects under study is 7.8 lines of code per hour. It is known that for Compiere 101,490 lines of code were written. As a consequence 13011.5 man hours

≈ 325.28 man weeks ≈ 83 man months ≈ 7 man years would be necessary in order to implement Compiere as a proprietary software.

Taking the median of productivity values 6.5 lines of code per hour are needed for the completion of such a project. So in order to develop Compiere, 15401 man hours ≈ 385 man weeks ≈ 97 man months ≈ 8 man years would be needed. Assuming that 1 man year costs 55,000 euros on average to a software house, proprietor development of Compiere cost is found to be 385,000-440,000 euros.

7 Conclusions and Further Research

According to the results of our measurement the overall quality of the source code of the application we measured, Compiere, is quite good. The maintainability of the application has been evaluated as GOOD, which implies that the product can be modified for adding new functionality of for fixing bugs, without a lot of effort, with respect to its size. The rest of the quality factors perform well and a more detailed look on the results given above, shows that the quality level of the software is above the average. These results are quite good for an F/OSS application, considering the decentralized nature of its development and the fact that an ERP/CRM system is not a piece of software that seems to be suited to the F/OSS development model. However, no matter how good these results are, further investigation is needed, especially in terms of functionality. High quality source code is not the only issue to take into mind, if someone is looking for an ERP system. Future work includes installing the system and exploring its functionality in real world case studies and assessing other external quality characteristics such as its usability.

References

1. Behlendorf B. Open Source as a Business Strategy. In: C. DiBona, S. Ockman, M. Stone (eds). *Open Sources: Voices from the Open Source Revolution*. O'Reilly and Associates, Cambridge, MA (1999)
2. Feller J., Fitzgerald B. *Understanding Open Source Software Development*. Addisson-Wesley, London (2002)
3. Fenton N.E., Pfleeger S.L. *Software Metrics: a Rigorous and Practical Approach*. 2nd edition, International Thomson Computer Press, London (1997)
4. Halstead M. *Elements of Software Science*. *Elsevier* (1975)
5. International Organization for Standardization. Information Technology-Software Product Evaluation: Quality Characteristics and Guidelines for their Use. ISO/IEC IS 9126. Geneva (1991)
6. McCabe T. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4): 308-320 (1976)
7. McConnell S. Open Source Methodology: Ready for Prime Time?. *IEEE Software,* 16(4): 6-8 (1999)
8. Netcraft. The NETCRAFT Web Server Survey. Avalable at: http:// netcraft.net/ survey/
9. Nichols D.M., Twidale M.B. The Usability of Open Source Software. First Moday, 8, 1. (2003) Available at: http://firstmonday.org/issues/issue8_1/nichols/index.html

10. Raymond E.R. (1999). The Revenge of the Hackers. In: C. DiBona, S. Ockman, M. Stone (eds). *Open Sources: Voices from the Open Source Revolution*. O'Reilly and Associates, Cambridge, MA (1999)
11. Raymond E.S. The Cathedral and the Bazaar. (2002) Available at: http://www.catb.org/~esr/writings/cathedral-bazaar/
12. Samoladas I., Stamelos I., Angelis L., Oikonomou A. (2003). Open Source Should Strive for Even Better Maintainability. *Communications of the ACM* (2003)
13. Schach S.R., Jin B., Wright D.R., Heller G.Z., Offutt A.J. Maintainability of the Linux Kernel. *IEE Proceedings – Software,* 149(1):18-23 (2002)
14. Stamelos I., Angelis L., Oikonomou A., Bleris G.L. Code Quality Analysis in OSS Development. *Information Systems Journal,* 12(1):43-60 (2002)