Enterprise Applications: Taking the Open Source Option Seriously

Nicolás Riesco B. and Jaime Navón C.
Computer Science Dept., P. Universidad Católica de Chile
{nriesco,jnavon}@ing.puc.cl

Abstract. Free and Open Source Software (FOSS) is becoming more popular. Nevertheless most CIOs wouldn't even consider this option for their enterprise information technology needs. We found that the three main concerns about FOSS have to do with legal issues, costs and support. We propose an initial framework to look at FOSS in a balanced, unbiased and systematic manner that can be used for evaluation of specific scenarios from very small companies to large ones.

1 Introduction

The interest for Free and Open Source Software (FOSS) has been growing in the last 5 years. Nevertheless, at the enterprise level, only a few companies use FOSS as their main software platform. They either buy proprietary enterprise application suites (ERP, World Class) or choose instead between Microsoft's .Net and J2EE platforms. Among the reasons often given by CIOs for this situation are:

- Fear of legal consequences
- It could end up being even more expensive
- Lack of technical support (no one to call)
- Doubts about critical issues such as performance, reliability, scalability, etc.
- Insufficient information to perform an in-depth analysis

We believe that this early ruling out of the FOSS option is a bad idea. There are many scenarios where this option represents in fact the best or even the only reasonable option. This is especially true in developing countries where there are few big enterprises but many small to medium size companies (in Chile these companies are known as PYMES).

We propose a framework that helps in the decision making process. First, we analyze the needs and requisites of enterprise software and the main concerns about FOSS related to these needs. Then, we examine the distinctive characteristics of FOSS in depth. Finally, we focus on the most important issues to develop metrics that allow us to say, more or less, how adequate would be FOSS for a given scenario.

Please use the following format when citing this chapter:

Riesco B., N., Navón C., J., 2006, in IFIP International Federation for Information Processing, Volume 214, The Past and Future of Information Systems: 1976–2006 and Beyond, eds. Avison, D., Elliot, S., Krogstie, J., Pries-Heje, J., (Boston: Springer), pp. 107–118.

2 The Enterprise Applications Habitat

An enterprise is an organization of people or entities that work together towards common goals; usually, large corporations. Enterprises often have important information technology related needs: information storage and retrieval, resource planning, customer management, accounting, etc. The software that supports these needs is called enterprise software or Enterprise Application (EA)[1].

Historically, EA used to run on mainframes, using proprietary systems such as HighExPlus, BancsConnect, and EX[1]. Nowadays, the mainframe approach has evolved first into the client-server computing model and then to an "n-tier" architecture where the presentation is separated from the business logic, and the business logic from the data.

EA does not run over bare machines. Many other software products could be operating between the EA and the hardware itself: the operating system, web servers (including special modules and plugins to support different programming languages), database servers, application servers, etc. Moreover, software development tools, libraries, IDEs, etc. represent also software products that should be considered. This paper is especially useful in choosing FOSS for this kind of software.

Since the above mentioned software will be supporting the EA, it is important to know whether there are special requirements that we need to take care of. Emmerich et al.[2] point out special requirements associated to enterprise software: high availability, scalability, reliability, performance, changeability and security.

According to the Gartner Group unplanned application downtime is caused: 20% by hardware, 40% by application failures (bugs, performance issues or changes to applications that cause problems) and 40% by operator errors[3]. If we leave aside human mistakes we would still have to consider both hardware and software faults. Since hardware problems are independent on whether we are using FOSS or not, we focus here in software faults only.

Replication, redundancy and clustering (including farms) are just a few of the techniques that can be used to respond to the above mentioned requirements. Generally speaking, all those terms refer to duplicating resources in order to achieve a certain degree of availability and/or to provide a faster response. All these techniques are available through proprietary products, but they are not exclusive to the proprietary software world; FOSS can provide them too.

3 Facts about Free and Open Source Software

Open Source Software (OSS) is any software that has its source code available[2]. It is based on the principles of Free Software, which defines four levels of freedom:[3]

- Freedom 0: To run a program, for any purpose.
- Freedom 1: To study how a program works, and adapt it to your needs.

[1] We will refer to EA and enterprise software as synonyms

[2] A formal definition at http://www.opensource.org/docs/definition.php

[3] http://fsf.org/licensing/essays/free-sw.html

- Freedom 2: To redistribute copies of a program to help your neighbor.
- Freedom 3: To improve the program, and release your improvements to the public, so that the whole community benefits.

In other words, Free Software is "The freedom to run, study, copy, redistribute and improve software". We will consider Free Software equivalent to OSS, and refer to them as FOSS.

3.1 Type of License

Licensing is very important when using a piece of software. Using FOSS does not make you the owner of the code; you can use it but with certain restrictions. Some important facts about licenses to take into account are:

- Author takes no responsibility for the software.
- There is no Warranty for the program.
- You are not required to accept the license, since you have not signed it. However nothing else grants you permission to modify or distribute the program[4].
- You may not sublicense the program except as expressly provided under the License[4].

A complete description of the different Open Source licenses can be found on the Open Source web site[4]. The more popular flavors are GPL, BSD and LGPL. Other flavors are, in general terms, variations of those three.

GPL stands for "GNU Public License" which gives you the right to use, distribute and modify any GPL software, as long as if you distribute it (whether you have modified[5] it or not) you must include the source code with it. That means the software stays free forever, and any improvements will be eventually available to anyone. This is called "Copyleft". As defined by the GNU, "is a general method for making a program or other work free, and requiring all modified and extended versions of the program to be free as well"[5]. Proprietary software cannot be based in part upon GPL software. If so, it cannot be distributed without the source code. GPL software should not be used on proprietary software (that would be senseless).

Thanks to these features, GPL license can be considered "viral", because any software released by the GPL license is "infected" with it. A "marketing-oriented" meaning for this is that the GPL software spreads at the speed a virus does, because it is free, good and users recommend it to other users.

LGPL (Lesser General Public License) works like the GPL license but applied to libraries. A program that is linked to a LGPL library, may be distributed without including its source code, but the library itself, must be distributed with it. Drivers usually fall in this category. MySQL connector, a driver/API provided by MySQL AB, had LGPL license once, but is now distributed under the GPL license. The consequence of this license change, is that, if you use the driver on your project, then your project must be released under the GPL license. This change is also known as GPLed (that code has been GPLed).

[4] http://www.opensource.org/licenses/index.php

[5] Lawyers call this "derivative work"

A BSD (Berkeley Software Distribution) license gives you the right to use, distribute and modify the software. No need to distribute the source code along with the binaries, but you must keep the original information about the author and some other stuff.

In a Dual Licensing scheme, an author willing to authorize other users to benefit from his work can freely determine the type of license to use. He is not obliged to give equal rights to all users and can therefore use several licenses[6]. FOSS vendors usually offer a GPL version of their software for public use and a proprietary version for those that might have problems with copyleft. Dual licensing can be considered as a licensing scheme that allows software vendors to provide a high quality enterprise-compatible FOSS product, with which they can profit. It also allows legal concerns about the origin of the software to be dismissed.

3.2 Legal Risks

Before we get into the legal risk analysis we need to remember a few things. First, the fact that the software is free does not imply property. No matter if the software was obtained for free or paying money for it. Second, there is no warranty. Third, modification is allowed, but it has to be explained within the concept of distribution. Finally, distribution is allowed as long as it fulfills the license's requirements. By using GPL, any derivate work must be distributed as GPL. By using LGPL you are allowed to link libraries into non-free programs, without "infecting" it, and by using BSD you are not forced to distribute the source code. Table 1 shows a comparison between restriction levels among licenses.

Table 1. Restriction level by License

License	Distribution
BSD	Non Restricted
LGPL	↑↑
GPL	↓↓
Proprietary	Restricted

Let's get now into the legal issues. Starting with the intellectual property (IP), companies need to be very careful about infringing IP rights when using FOSS. In the US, copyrights have been filed for not only lines of code, but also for topics such as look and feel, technical, or operational processes[7]. Copyrights can be infringed easily, what makes it difficult to manage. On the other hand, GPL itself has never been challenged in court[8, 9, 10], lawyers can offer only theories, not facts[9].

Including FOSS as proprietary: CherryOS, a MacOSX emulator for Windows, was discovered to use GPL code on its proprietary software. As a result they had to release its software under GPL. Recently, MySQL sued NuSphere for GPL violation. The case was settled out of court[10].

The recent SCO case and the following controversy contributed to increase the public awareness about the legal issues involved with FOSS. A lot has been said, nothing is really clear and SCO has been unable to prove actual IP infringement. Is the opinion of R. Stallman that if SCO was "right", then it would be enough to remove some small part of the Linux code and the problem would be over. If SCO's intention was to generate FUD (Fear Uncertainty and Doubt), they might have succeeded[10]. Companies like IBM, HP, RedHat and JBoss offer insurance for this kind of legal problems on their products. Other companies like OSRM[6] provide insurance services to mitigate Open Source License risks and related IP issues.

Some people, like OSDL's CEO Stuart Cohen, think that the SCO controversy "was the best thing that ever happened to Linux"[11], giving Linux a boost. A detailed timeline of the SCO Controversy can be found at Linux.org's site[7].

4 Towards a Decision Framework

As we said before, there are many factors involved in the decision to use or to rule out the use of FOSS in the enterprise: costs, support, control and flexibility, open standards, product maturity, security, scalability, legal issues (unexpected license costs and possible lawsuits like SCO), etc. Of all these factors there are three that to many are considered as "fundamental factors":

- Legal: according to Gartner[12] legal concerns are at the top of the list of "fear factors of Open-Source Adoption". Forrester[13] considers "unexpected license costs" as an important concern too.
- Costs: according to Forrester[13] and Dravis[14] cost is a significant factor.
- Support: lack of it is considered by 57% as the biggest concern[13].

Other good reason to focus only on these three fundamental factors is that the rest depends more on the specific product we are considering. The "Business Readiness Rating for Open Source", which is being proposed as a new standard model for rating FOSS or the "Open Source Maturity Model (OSMM)"[15] and "CapGemini Open Source Maturity Model" may all be used to this end.

Let's focus then on those fundamental factors. First, each of these three factors is more or less relevant depending on what stages of software development we are considering: Planning, Deployment or Operation. As table 2 shows, "Legal Concern" is mainly associated to the Planning stage, meanwhile "Cost Factors" and "Support" are more relevant during Deployment and Operation.

Table 2. Fundamental Factors v/s Stages

	Planning	**Deployment**	**Operation**
Legal	*		
Cost		*	*
Support		*	*

4.1 Legal Aspects

If we want to avoid "Unexpected license costs and possible lawsuits" it is critical to pick the right license type. There are some other issues to be considered that have

[6] http://www.osriskmanagement.com

[7] http://www.linux.org/news/sco/timeline.html

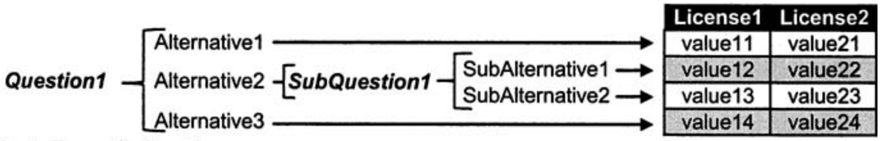
been analyzed and explained in [9, 16, 17], but the fundamental decision involves finding out what type of license is adequate or "compatible" with our company.

One way to go is to perform a rigorous legal analysis of every license of every piece of software that is offered to us. This process could be expensive and time consuming. A better way to go is to conclude the type of license we need from a strategic analysis of our business and then get only products that offer these licenses.

Here we propose a framework that can be used to determine the compatibility that the different kinds of licenses have with a given company. This framework, that we call OSCoM (Open Source Compatibility Metrics)[8] is based on a series of questions that must be answered by managers or people who really know the company. The questions are grouped into categories according to:

- The use that the software is going to have[9]
- Whether we plan to perform modifications to the software
- The type of distribution the software is going to have

Each question has several alternatives, which might have a sub-question, allowing to divide a complex question into a series of simple ones. The sub-questions inherit the category of the "father". Figure 1 shows the idea.



License1	License2
value11	value21
value12	value22
value13	value23
value14	value24

Fig. 1. Example Question

Another way to see it is trough a "n-ary" tree (Figure 2a) in which each level contains questions and answers and each leaf contains an option without sub-question (Figure 2b).

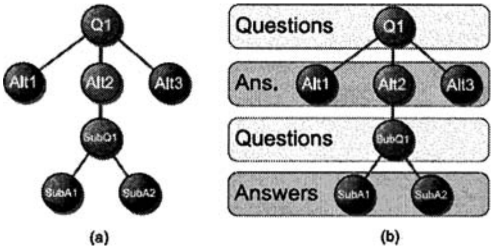


Fig. 2. Questions Tree (a) and Tree Structure (b)

Every leaf has some associated info. Each license is categorized into "best", "ok", "no" and "warning". These words represent different numeric values as shown in (1), (2), (3) and (4). For each license we can also keep some additional comments.

[8] http://oscom.sourceforge.net

[9] It is not the software itself what is important, but what we are going to do with it

It is possible then to go down through the tree to get the number and comments associated with a license when we arrive to a given leaf. This process is repeated three times for use, modification and distribution to obtain a total number and a final compatibility value that goes from 0 to 100. A negative value means incompatible

Scores:

$$\text{best} = 100 / \text{totalQuestions} \tag{1}$$

$$\text{ok} = \text{best} / 2 \tag{2}$$

$$\text{no} = -100 \tag{3}$$

$$\text{warning} = [\,\text{no} \mid \text{ok} \mid \text{best}\,] \tag{4}$$

About the numbers: (1) The total number is distributed among all the questions. (2) A percentage of the best, we take 50%. (3) More than the number itself the important thing here is that any incompatibility must show up clearly, even if it is only one. We take the value -100 so the total score will always be negative. (4) By default it corresponds to the "no" value, but it gets a new value if it is fixed.

Notice that although the analysis is not dependent on a particular piece of software, it is dependent on the use, modification and distribution that we have in mind. It is completely different, for instance, software for development (e.g. Eclipse), for in-house use (e.g. OpenOffice) for service providing (e.g. Compiere) or for selling. In this last case FOSS is not an option but we might opt for a dual licensing scheme.

4.2 Cost Issues

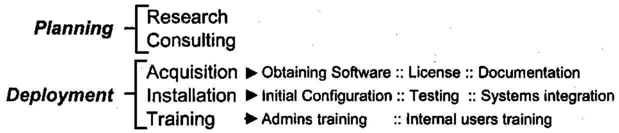
When considering costs, no matter if it is FOSS or proprietary software, we have to be very careful. It is wrong, for instance, to think that because there is no need to pay for the licenses the associated costs of the FOSS based solution is zero or near zero. The total cost in that case could even be higher than the costs of the licenses for a proprietary alternative.

Most people think that the best way to consider the cost variable is to take the "Total Cost of Ownership" (TCO). Another option would be the "Return Of Investment" (ROI) but here we are more interested in comparing alternatives than in knowing if the money we are spending in technology is well spend.

If we are taking TCO as our comparison criteria, it is necessary to consider each stage of the process (Planning, Deployment, Operation). Each of those has its own costs that need to be identified (some are of a fixed amount and some are variable), for example:

- Fixed Costs: Planning (*Research, Consulting*) and Deployment (*Acquisition, Installation, Training*)
- Variable Costs: Maintenance (*Basic Configuration, Reconfiguration, Specialized Support, Internal Support Personnel*)

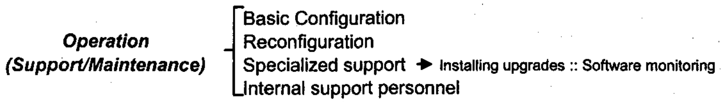
Figures 3 and 4 show a possible cost structure. Among the acquisition costs we can find for example the licenses. Installation costs will include initial configuration of the systems and the integration with other existing systems.



Planning – Research; Consulting

Deployment – Acquisition ▶ Obtaining Software :: License :: Documentation; Installation ▶ Initial Configuration :: Testing :: Systems integration; Training ▶ Admins training :: Internal users training

Fig. 3. Fixed Costs Structure



Operation (Support/Maintenance) – Basic Configuration; Reconfiguration; Specialized support ➔ Installing upgrades :: Software monitoring; Internal support personnel

Fig. 4. Variable Costs Structure

There are other issues that might be necessary to consider in the TCO analysis. For a more detailed analysis see for instance "Managing Your IT Total Cost Of Ownership"[18]. The authors include downtime, futz, virus resistance and power consumption in the list. They also explain that "only a 20% of TCO lies in initial acquisition costs and the rest lies in administration costs" which is coincident with what we said about the danger of putting to much emphasis in license costs.

An example of the use of TCO to compare Linux v/s Windows solutions can be seen in Cybersource's study[19]. It shows that, in this case, the FOSS alternative produces up to 36% in savings. Another study by IDABC[20] found savings of 66% and Robert Frances Group found even larger savings[21]. Finally, Forrester, taking into account a sample of 14 companies, concluded "software costs for Linux proved to be less expensive, on a per-server basis, than Windows by at least 60%"[7].

Not all studies found such big cost advantages for FOSS; Bearing Point concludes that costs within medium and enterprise scenarios over a five year period do not significantly differentiate Windows Server 2003, Red Hat Enterprise Linux 3 or Novell/SUSE LINUX 8[22]. They also say that "Areas of differentiation to consider include such factors as value-added functionality, vendor support, productivity advantages, and the costs to deploy, manage and maintain an infrastructure".

So the cost aspect ends up being a tricky business. The important thing to remember is that we must perform TCO calculations considering as many variables as possible. The results will depend on many factors like business size, period of time and service level considered.

4.3 Support

Support has always been an important component in enterprise software. We could argue that support is already taking into account when considering costs. Comments like "the number of people I have assigned to Linux is almost double my Windows staff for the same number of servers"[7] or "maintenance and support was 3-14% higher on companies using mixed operating systems environment than those using only Windows"[7] show that cost and support are indeed related one to the other.

We believe however that this is an issue that needs to be considered by itself. Consider for instance a solution that involves a piece of FOSS for which it is very difficult (almost impossible) to get support no matter how much money we are ready to pay. Managers would probably rule out immediately the use of any unsupported piece of software.

IT companies are taking advantage of a business opportunity. In fact, support has become an important source of income for IT companies that are finding more and more difficult to make money just by selling the hardware or the associated software. In some cases the product is FOSS and cannot be sold without a license violation, so they give away the software and charge for support. Companies like JBoss, RedHat, and Suse represent just a few examples of this.

Other companies like MySQL, Sleepycat (Berekely DB) and Trolltech (QT) offer a dual licensing scheme for companies where "copyleft" is not an option. Other interesting initiatives include Spike Source, which offers pre-tested FOSS stacks and Source Labs offering maintenance and support for FOSS. Dell has announced support for MySQL and JBoss software that run in their "Power Edge" servers. There is even a support search engine[10].

Although in many cases the hardware/software vendor also provides support (because it is good business) we are not forced to this. Any person or company with the relevant knowledge could do it. In that case FOSS poses a small additional challenge: documentation is often poor or even inexistent (FOSS projects struggle to get people who are willing to do documentation) making it very hard to fix uncommon problems. At this time Linux skills are harder to find compared to Windows but this may change in the future[7].

Contrary to what many CIOs may think, defect density in FOSS releases will generally be lower than commercial code that has only been feature-tested, that is, received a comparable level of testing. From other side, FOSS developments exhibit very rapid responses to customer problems. In successful FOSS projects, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems[23]. There is no methodology for evaluating support that we are aware of. We suggest the use of a sequence of steps as shown in Figure 5.

The first step is to decide whether we are going to use internal or external support. In the case of contracting external services we must assure not only availability but also the credentials of the provider, experience, service level options, etc. The internal support option requires answers to questions such as:

- Do we already have a support department? Do we want to create one?
- Do we have the knowledge or experience in-house?
- Do we have the necessary people? Gurus?

[10] http://www.findopensourcesupport.com/

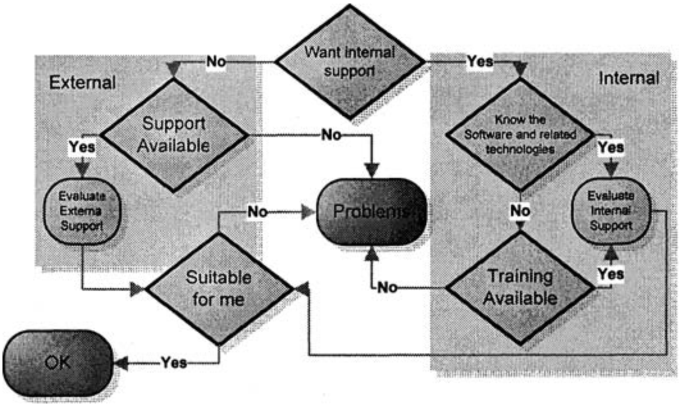
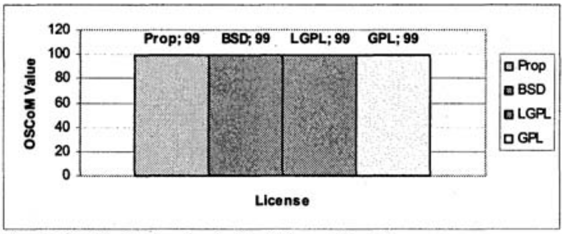


Fig. 5. Support Evaluation Diagram

4 Results

So, is FOSS appropriate for us? This is the question we started with. We have proposed a more structured way to answer the question that requires a view from three different sides: legal, cost and support. For each of them we need to look at the special characteristics of our company and the specific scenario where the decision is being taken.

Let's say we are considering a very small company (micro company) with one or two people using just one computer connected to the internet, used for reading emails, word processing and spreadsheets. The OSCoM analysis would show that all the licenses get similar scores and therefore license compatibility is not significant.



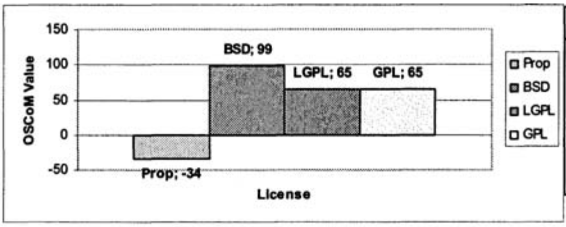
Category	Answer
Use	Normal
Modifications	Simple
Distribution	No
Future-Dist.	No

Fig. 6. OSCoM Analysis for Micro Size Company

The Cost analysis shows that the use of applications with no associated license fees produces important savings (OpenOffice instead of Office). Finally with respect to support, there is only one computer running well known application software. Probably all that is needed is to make a call in case of any problem. Although it might be a little harder to find a Linux expert compared to someone who can solve a Windows problem it is not indeed a big issue.

Now, if we consider a small company, the scenario may involve a LAN with one computer acting as a mail-web-db server. If the company chose FOSS it would be a combination of Linux, Apache, MySQL and PHP. Otherwise it could be Windows+IIS+SQLServer. It is not uncommon to start thinking in intranets and Web based applications which could involve the use of a more complete platform like .Net or J2EE (J2EE usually considered too complex for this type of scenario).

The OSCoM analysis is now a little more elaborated:



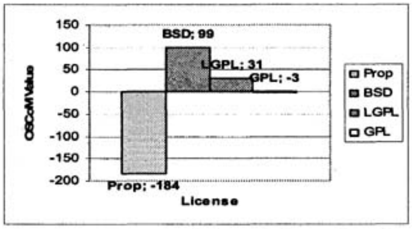
Category	Answer
Use	Normal
Modifications	Complex
Public Modif.	No
Distribution	No
Future-Dist.	No

Fig. 7. OSCoM Analysis for Small Size Company

Proprietary licenses that do not allow access to the source code make special purpose solutions hard to do. If we compare BSD with GPL and LGPL, the first one is the best if we do not plan to make all our changes and modifications public.

Cost analysis is also more complex. The use of a solution like Apache+MySQL might appear ridiculously low compared to a IIS+SQL Server equivalent but we showed that this is not the only cost that has to be taken into account. Support issues also start to be important. We may have no people inside that can assume a support role at all. In that case we should consider to put together a support group or find an external service. It should not be hard to find the needed support in case it is needed.

For a medium size company, we can find not only web servers but application servers and all the components of a full size software platform like J2EE or .Net. A decision to go open source is even harder if the critical applications are already running in one of these proprietary platforms (Java being easier than .Net). Here is an OSCoM scheme for this new scenario:



Category	Answer
Use	Extension
Program Use	Proprietary
Modifications	Special
Public Modif.	No
Distribution	No
Future-Dist.	Yes

Fig. 8. OSCoM Analysis for Medium Size Company

Here, future distribution potential increases the differences compared to the previous scenario. LGPL is now better than GPL because of the extensions to proprietary software.

5 Conclusion

Free and Open Source Software represents a big opportunity not only to individuals or small companies. Very good quality software with functionality and performance similar or even better than expensive proprietary software is available with no license payment associated to it. Nevertheless CIOs and IT managers often fail to even consider this option. We have proposed a framework that can be used to analyze the Open Source options in a balanced and systematic way. The framework considers three different views: legal, costs and support. Each of these views requires answering questions about the specific scenarios where the software is going to be installed and used. Although this is just an initial attempt and more work needs to be done, we found that it is already useful in specific scenarios from very small up to medium size companies.

References

1. A. Gokhale, D.C: Schmidt, B. Natarajan, and N.Wang. Applying model-integrated computing to component middleware and enterprise apps. Comm. ACM 45 (2002).
2. W.Emmerich,E.Ellmer, and H.Fieglein. Tigra - an architectural style for enterprise application integration. In: ICSE '01, IEEE Computer Society (2001) 567–576.
3. D. Scott. Making smart investments to reduce unplanned downtime. Gartner 1999.
4. GNU. Gnu general public license (2005) http://www.gnu.org/copyleft/gpl.html.
5. GNU. What is copyleft? (2005) http://www.gnu.org/copyleft/.
6. P.E. Schmitz and Castiaux, S. Pooling Open Source Software. IDABC (2002).
7. J. Giera. The Costs And Risks Of Open Source. Forrester Research (2004).
8. B. Fitzgerald and G. Bassett. Legal Issues Relating to FOSS (2003).
9. J. Michaelson. There's no such thing as a free (software) lunch. Queue 2 (2004).
10. A.G. González. The calm before the storm? legal challenges to open source licences. In: Proceedings of the Int. Conference on Open Source Systems (2005).
11. ZDNet-UK: Sco was the 'best thing that ever happened' to linux. http://news.zdnet.co.uk/0,39020330,39190780,00.htm (2005).
12. M. Driver. .NET, Java and OS: A 3-Way Race for Developer Platforms (2004).
13. T. Schadler. OS Moves Into The Mainstream. Forrester Research (2004).
14. P. Dravis. OSS: Case Studies Examining its Use. The Dravis Group (2003).
15. B. Golden. Succeeding with Open Source. Addison-Wesley Professional (2004).
16. P. Gustafson and W. Koff. Open Source: Open for Business (2004).
17. D. Ascher. Is os right for you? (a fictional case study). Queue 2 (2004) 32–38.
18. J:S. David, D.Schuff, and R.S. Louis. Managing your TCO. Comm.ACM (2002).
19. Cybersource: Linux vs. windows total cost of ownership comparison (2004).
20. S. Hnizdur, K. Matthews, E. Bleasdale, A. Williams, A. Findlay, S. Atkinson, and C. Briscoe-Smith. The IDA Open Source Migration Guidelines. IDABC (2003).
21. Robert Frances Group. TCO for Linux in the Enterprise (2002).
22. BearingPoint. Server op. system licensing and support cost comparison (2004).
23. A. Mockus, R. Fielding. Two case studies of open source software development: Apache and Mozilla (2002).