On the Influence of Free Software on Code Reuse in Software Development

Marco Balduzzi <marco.balduzzi@madlab.it>

Abstract

Software reuse has become a topic of much interest in the software community due to its potential benefits, which include increased productivity, quality, and reliability, and decreased costs and implementation time. There are many potential alternatives to consider including Commercial Off The Shelf (COTS) components, Free/Open Source Software (F/OSS) components, or Custom software. This paper analyses the main advantages and issues related to reuse of Free Software components in software development.

1. Introduction

The Free Software movement represents one of the most interesting and influential trend in the software industry over the past decade. Today, more and more software houses adopt Free/Open Source Software (F/OSS) instead of Commercial Off The Shelf (COTS) in their development processes. The choice of reusing Free Software components has profound implications on the quality attributes of the process and of the resulting product.

This paper discusses the main advantages and issues correlated to the reuse of F/OSS products in term of software availability, customization, integration, portability, scalability, cost, licensing, security, human skill and support.

The remainder of this paper is organized as follows: section 2 presents the concept of Free Software, explains the differences between “Free Software” and “Open Source” terminologies, and illustrates the Free Software development approach. Section 3 analyses and discusses benefits and issues of the Free Software reuse. Conclusions are presented in Section 4.

2. Free Software

The original definition of Free Software [1] asserts that a program is Free Software if the user has the freedom to run, copy, distribute study, change and improve the software. More precisely, it must guarantee four kinds of freedom:

- the freedom to run the program, for any purpose (freedom 0).
- the freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- the freedom to redistribute copies so you can help your neighbours (freedom 2).
- the freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

A usual error is to interpret the "Free" term for "not to pay" instead of "freedom". So, the "father" of the Free Software movement (Richard Stallman [6]) suggests thinking of free as in free speech, not as in free beer.

Usually, Free Software programs are distributed using the GNU General Public License (GPL) [2], but exist many other GPL-compatible licenses [3]. The Free Software movement, contrarily to the Open Source community, does not accept licenses which permit to give out a program without the its source code, like the Berkeley Standard Distribution (BSD) [4] license.

2.1 Free Software or Open Source?

Today both "Free Software" and "Open Source" terms are used to indicate the same kind of software. However the two terms refer to two separate movements with different views, goals and ways of looking to the world. For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, "Open Source is a development methodology; Free Software is a social movement". For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and Free Software is the solution.

2.2 The Free Software approach

The Free Software approach is the a clear example of large-scale software reuse.

Eric Raymond has characterized the Free Software development process as a bazaar [5]. The bazaar model of software development is based on the two key principles: “given enough eyeballs, all bugs are shallow” and “release often and release early”. The idea is to build a “draft” code, to modify and to distribute it through Internet, waiting for the feedback of the users. The users themselves are programmers that run, debug and patch the current software version, or implement new features. The contributions are sent to the project development repository so as to be immediately available to the Free Software community. This approach takes advantage of the collaboration of many unknown worldwide eyeballs, which act only “just for fun”, like a famous Linus Torvalds sentence says.

The bazaar approach is in direct contrast with the traditional software development process, which Raymond characterizes as a cathedral. In the cathedral model, the development process is centralized and only an enclosed group of programmers develop the software. The architectural designs are carefully drafted, and the user feedbacks are limited because the software releasing process is very slow. The whole software cycle is well defined from the beginning and uses formal notations.

So, the Free Software approach looks like the typical din and chaos of a bazaar, rather than the formal and quiet rooms of a cathedral.

The success of the Free Software development model is evident from many case studies of large successful systems including Linux Kernel OS [7], Apache [8], Mozilla [9] and OpenOffice [10]. Sourceforge [11], which is the largest repository of F/OSS applications available on the Internet, shows in its homepage the concrete results of the Free Software approach: 99,329 registered projects and 1,060,263 registered users.

3. Free Software and reuse

The decision of using existing software versus building from scratch custom software is one of the most complex and important of the entire development process. The correct trade-off is reached after having analysed advantages and issues correlated to the reuse of the two solutions. Otherwise, this paper does not focus on problems concerning reuse itself, already widely discussed in literature ([R8] and [R20]), but tries to point out the benefits and issues concerning the reuse of Free Software components instead of COTS in software development.

3.1 Software availability

The market of the Proprietary Software is not homogeneously distributed: there are few huge products and many little ones. At the same time the hugest ones own the largest share of users of their category. Two famous examples are the Microsoft Explorer browser and the Outlook Express email-client, both widely diffused in the IT community. This situation creates two types of problems: the choice of the COTS components is often dictated by market rules and not by software quality and the number of competitive products available is restricted.

On the other hand, in the Free Software market a real competition and a fair distribution exist. The greater availability of heterogeneous software components, offered by the Internet F/OSS repositories (i.e. Sourceforce [11] and Freshmeat []), allow software engineers to choose the more appropriated component, without a dialogue with the market. Otherwise in the development process, the steps of choosing, analysing and testing software components demand much time and effort. So, the larger number of possible candidates increases complexity and time, affecting the entire development process cost. In contrast the choice between COTS products is usually easier and quicker.

In spite of the wide availability of different F/OSS components, nowadays many proprietary products doesn' thave a really competitive F/OSS correspondent. I.e. AutoCAD is a widely diffused bi-dimensional CAD which doesn' have a fair "competitor" in the Free Software community.

3.2 Customization and integration

One of the main innovations of Free Software is the availability of the programs' source code. The "freedom 1" expressed in the Free Software definition encourages the study of the internal software structures and functionalities, allowing the end-user to adapt the component to his requirements. So, the developer is facilitated to the customization and the integration of the reusable component into the existing system.

The COTS software instead denies "freedom 1": the source code is not released and the modification and the optimization of the software are usually strictly prohibited. Since most of the commercial components cannot be changed by the user, the integration of COTS products may require considerable effort and costs. In [12] is introduced an pratical method to estimate integration costs using COTS components.

Vendors of Proprietary Software are typically under pressure to differentiate their product with extensions. These value-added features can often lead to incompatibility and confusion. To protect market share, vendors must selectively add proprietary extensions to maintain a lock on their client-base.

The Free Software community has no such pressures. Instead its value proposition is to support the standards very closely. The Free Software community hopes to use the consensus achieved in that standard and experiences from multiple product implementations, as a way to stabilize the technology domain, create a commodity item from that technology and thus create another stable building block in the technology layers that make up distributed systems.

Reusing standard-compliant components simplifier integration process between different modules, while the use of a proprietary COTS extension increase drastically integration effort. An evident problem could occurs when a component is not able manage the output of another one, like when an audio reader needs to reproduce a proprietary extension audio file as Window Media Audio (WMA). Standard technologies like Ogg Vorbis simplify code reuse ensuring a correct integration.

3.3 Portability and scalability

Free Software code is normally much more portable than Proprietary Software. The availability of the source code permits or facilitates the adaptation and the recompilation of the components for different environments (O.S.[1] and architecture). Instead, a binary code compiled for a specific environment difficulty works on different one. This is the case of COTS software which is normally compiled and distributed for a specific environment. The end-user is not able to adapt and run the program in a different environment, but needs to purchase of a new specific version, which could be unavailable or too cost expensive. Moreover portability requires standards. Standard-compliant software does not need a particular environment to work; in fact it uses standardized API and libraries. So it can be ported more easily.

The scalability of the Free Software applications is unbeatable, mainly because the source code can be specifically optimized for different platforms. Many of the F/OSS

[1] Operating System

components guarantee an extreme flexibility and personalization of the compilation parameters and of the configuration.

GNU/Linux for example works on many PDA [13], old low-performance computers, old and recent game-box consoles, common modern PC hardware, high availability servers, and it is also used to support massive parallel processing [14]. On the other hand, Window XP requires a high computational X86 processor with at least 512Mb of RAM and difficulty works on old hardware.

3.4 Cost

Price is one of the most touted benefits of reusing F/OSS components ([15] and [16]). Although in the Free Software terminology the "Free" term is interpreted as "freedom" and not "not to pay", usually it is possible to obtain F/OSS components without paying money, simply downloading freely from Internet. This is in agreement with "freedoms 0" and "freedom 2" of the Free Software model which allow redistributing and reusing the component for any purpose. However, most people (especially beginners and those without high-speed Internet connections) could pay a small fee to a distributor for a nicely integrated package with CD-ROM, paper documentation, and support.

Total Cost of Ownership (TCO) is an increasingly important factor to be considered when making any software or hardware purchase. Simply put, the TCO should express not only the initial cost of purchasing the components, but also the ongoing costs incurred by continuing to maintain and support the system (the "life-cycle of costs"). John Favaro in [R14] compares some further approaches to reuse investment analysis.

So, the F/OSS reuse is not completely cost-free, because the cost of paper documentations, support, training, system administration must be considered, just as it was with proprietary software reuse. Reusing F/OSS products implies a reduction of upgrade and maintenance costs. A standard cost for upgrading a COTS component is about half of the original price, while the upgrade patch of a Free Software component can be optained from Internet without paying, and used multiple times. In contrast, usually Proprietary licenses prevent from applying a single upgrade patch to many products.

At the end, the high scalability of Free Software products on slowest hardware platforms results in lower hardware costs and in some cases requiring no new costs.

3.5 Licensing

The freedoms expressed by Free Software licenses have the big advantage to permit the free distribution of the programs, lowering drastically the purchasing cost of F/OSS components. Then, the possibility to access at low cost an unlimited Free Software repository, has deep implications on integration of Free Software components with the existing system.

The main idea is that a program reusing F/OSS components is considered to be a “derivative work”. A great number of licenses approved by the Free Software community (i.e. the GNU Public License), prevents the user from releasing the software with a different license. So, a F/OSS component integrated in a commercial software must be released with the original license and must agree with the same obligations (i.e. the source-code must be available and freely modifiable). Many organizations consider an evident loss of reserved information the commercialization of the resulting software (or only the F/OSS modules) with a Free Software license. The public availability of the source code could damage the profit and the competition with other software houses.

Close to the GPL, other Free licenses (i.e the Lesser General Public License [17]) permit to integrate a F/OSS module in a non-free software environment avoiding to distribute its source code. Otherwise, if the reused F/OSS component has been modified, the normal conditions of the GNU Public License are applied.

Also, the license violation risk derived from the reuse of F/OSS components often discourages the software organizations; even the remote possibility that a large scale software product could be claimed to be risking its commercial distribution by reusing F/OSS components, frequently results in the decision to include only Proprietary Software. The many shades included in the software licenses induce most commercial software organization to pay a legal expert, who is specialized in ensuring the legal ramifications of reuse of the F/OSS software.

3.6 Security

No one can assert whether a Free Software module is going to be more secure than proprietary one, or vice versa. The discussion about the security offered by the two approaches is still open and every day information security experts publish in the Internet articles that lean towards either of them.

An important truth is that does not exist completely secure programs and that it's very difficult to quantify the security of a software component. Every program could contain a lot of conceptual or syntactic programming errors, or could work incorrectly due to a wrong configuration or to a problem with an external library or component.

The security of the Free Software applications comes from the bazaar principle “given enough eyeballs, all bugs are shallow”. In other words, since the source code is open for all to review, numerous programmers worldwide will be examining the code for security vulnerabilities. The high availability of contributors improves the process speed of bug fixing: auditing, exploiting and patching.

On the other hand, closed source applications are only audited for vulnerabilities by paid programmers working behind “closed doors” on proprietary products. It is easy that the vendor allocates a few amounts of human resources to the bug fixing process, since it is more profitable to sell many bugged products than a single secure one.

The vendors of Proprietary Software promote the security of their products with the concept of “security through obscurity”. A system relying on security through obscurity may have theoretical or actual security vulnerabilities, but its owners or designers believe that the flaws are not known, and that attackers are unlikely to find them. Sometime, the cost for the vendor to fix the bug may be too high and not profitable advantageous to build and release a relative patch.

At the end, since the user community does not have access to the source code, the external auditing process is very slow and difficult: the bugs are discovered by disassembling the program or testing the program with a black box approach[1].

3.7 Human skill and support

One of the primary reasons many organizations are slow to adopt Free Software solutions is the lack of IT people skilled in the Free Software technology. The adoption of F/OSS components requires a different modus operandi: the IT user must be able not only to reuse the component like a black-box, but also to modify and adapt it to its own

[1] A software testing technique whereby the internal workings of the item being tested are not known by the tester. In a black box test on a software design the tester only knows the inputs and what the expected outcomes should be and not how the program produces those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications.

requirements. It must know the techniques to develop software in a Free Software environment.

The major reason for the lack of skills is the current state of the education sector. Due to the great demand for IT trained people, many IT related courses are poorly designed courses, which do not teach the theory and the principles of IT. Instead these courses explain how to use COTS packages; and programming is learned using Proprietary Software development products and not studying the principles of programming, algorithms and languages theory. The result is poorly trained graduates without any foundation in the principles of their craft and little or no analytical ability. These graduates also go into industry with a preference for the COTS system they are familiar with. It is not surprising that many employers now days require job applicants to have additional professional qualifications.

Moreover a lot of Free Software components are UNIX derived , which traditionally lacks a consistent look-and-feel with which many IT professionals are familiar. Free Software components usually run from command line or could adopt a particular GUI (i.e. the UNIX-like O.S. are distributed with at least five different Window Managers and GUI libraries). Instead much of the COTS software inherits the GUI from the well-known Windows interface.

Another issue with the adoption of F/OSS components is the different modality of support offered by the Free Software community. Technical issues arising from COTS software are directly handled by the vendor: by telephone or by mail (i.e. Microsoft has a "Knowledge Base" of technical issues for all of their software). Such issues arising from a Free Software component may be handled by a common support group of contributors to the project (often via an on-line mailing-list or discussion forum). Custom software technical issues would be referred to the development team.

4. Conclusion

This paper has presented and discussed the most important advantages and issues related to the Free Software reuse. The lower cost and the greater portability, scalability and integration of Free Software induce to consider the reuse of F/OSS components a winning solution. However, the issues concerning the use of Free-licensed software, and the higher and particular skills required by handling the Free Software products slow down the adoption of F/OSS components by the software houses.

The many implications correlated to the use of Free Software, make difficult and important the correct choice between the two alternatives. Every software application complies with specific requirements that affect the evaluation of the reusable components. Globally, it is easy to think that the diffusion of Free Software components will increase and that much more Information Technology company will found their business on Free Software.

5. References

[1] The Free Software Definition, http://www.gnu.org/philosophy/free-sw.html

[2] GNU General Public License (GPL), http://www.gnu.org/licenses/gpl.html

[3] List of Free Software Licenses, http://www.gnu.org/licenses/license-list.html

[4] The BSD License, http://www.opensource.org/licenses/bsd-license.php

[5] Eric S. Raymond, “The Cathedral and the Bazaar”, http://www.catb.org/~esr/writings/cathedral-bazaar/

[6] Richard Stallman's Personal Home Page, http://www.stallman.org/

[7] Linux OS, http://www.linux.org

[8] Apache Web Server, http://www.apache.org

[9] Mozilla Browser, http://www.mozilla.org

[10] OpenOffice, http://www.openoffice.org

[11] SourceForge, the largest F/OSS development website, http://sourceforge.net/

[12] Yakimovich, Bieman, Basili: “Software Architecture Classification for Estimating the Cost of COTS Integration”, ICSE 1999

[13] “Linux sets its sights on the PDA market”, http://www.linuxdevices.com/articles/AT8728350077.html

[14] TOP 500 List, http://www.top500.org/lists/current.php

[15] Brendan Scott, “Why Free Software's Long Run TCO must be lower”, http://www.members.optushome.com.au/brendanscott/papers/freesoftwaretco150702.html

[16] “Windows VS Linux TCO study”, http://www.theage.com.au/news/Breaking/TCO-study-Linux-wins-again/2004/12/13/1102786990788.html?oneclick=true

[17] GNU Lesser General Public License, http://www.gnu.org/copyleft/lesser.html

[R8] M. Morisio, C.B. Seaman, A.T. Parra, V.R. Basili, S.E. Kraft, S.E. Condon, “Investigating and Improving a COTS-Based Software Development Process”, ICSE 2000

[R14] John Favaro, “A Comparison of Approaches to Reuse Investment Analysis”, ICSR 1996

[R20] Brereton P., Budgen D., ”Component-based Systems: a Classification of Issues”, IEEE November 2000