

# Free software contributions to improve traditional software management projects

Ma. Jesus Marco Galindo

Universitat Oberta de Catalunya  
Av. Tibidabo 39, 08035 Barcelona  
email: mmarcog@uoc.edu

**Abstract.** Free software development emerges like a quite effective proposal that improves final quality and reduces development time in certain types of software construction. For that reason, current software engineering research concentrates important efforts to analyze its peculiar properties. This paper examines some aspects of free software related to the management world, and study the relationship among this two environments. It analyzes how free software projects are managed and which practices can be taken to improve traditional project management models.

**Key words:** Free software, proprietary software, project management, management software.

## 1 Introduction

The definition of free software is fundamentally a legal concept. The only difference between free software and proprietary software is the license [16]. The most popular free software license is the GNU GPL - General Public License -, by which the author of software allows its use, modification and the distribution on a free basis.

A part from this legal definition, if we analyze free software projects from a technical point of view, we find important differences with respect to traditional development of proprietary software. In fact, the license has lead us to new forms of software construction. Therefore, besides legal environment in which free software is located by definition, we can deepen in these differences from other perspectives such as development model, management model, social context and even business model. In this paper, we will center exclusively in management model.

This paper is organized as follows. In section 2 we will describe the main problems of traditional software project management (mainly based on management software). The goal of section 3 is to analyze the most prominent properties that determine the management processes carried out in free software projects. In section 4, we point out how a free software projects analysis can contribute to improve proprietary software development. Finally, in section 5 conclusions are exposed.

## 2 Traditional software projects management

One of the main problems of software engineering is the fact that most proprietary software construction projects do not finish by the time and predicted costs or they do not perform the designed functionalities. Some authors roughly estimate that 70 % of projects fail in those categories [5]. Some projects fail for technical reasons inherent to the own application, but many others do due to an inadequate project management. A more detailed analysis can be found in [3, 6]. Therefore, a key point in the software project development is its management. Improving software project management will have effects on better project development and on better project result.

### 2.1 Properties

According to Duncan [4], project management is an elaborate body of knowledge produced starting from the professional exercise of whom practice it, increasingly structured and coded through case studies, methodologies and academic approaches.

Project management can be defined like a management process oriented to the conception, the starting, the monitoring and the evaluation of a particular information called project. The main objective of project management is, therefore, to guide the project development that means to obtain their functionalities, in the established terms and with the assigned resources. A proprietary software project is, thus, somewhat temporary that can not use the resources assigned indefinitely.

Therefore, proprietary software development project will be competitive only if a tradeoff between the following three variables is achieved:

- Quality, understood as requirement satisfaction.
- Available resources.
- Foreseen term.

Proprietary software development projects are developed in different phases. These phases depend on the chosen methodology but its main features are:

- Selection and approval: the project is conceptualized and its viability is studied.
- Requirements definition: it determines specifically what the system is intended to do.
- Estimation and planning: it identifies and estimates the specific work to be done and tasks are scheduled.
- Implementation: it controls and monitors the project in order to detect estimation deviations and allow them to be corrected.
- Closing: the project is delivered to the user and correctness between the original requirements and the final product is analyzed.

## 2.2 Project management problems

Olson in [11], argues that most frequent errors produced in project management are the following:

- Not enough user/client participation during the project.
- Lack of communication among the managers, the project team and the client.
- Difficulties to determine the project objectives and the project reach.

The first two affect to the project from the beginning until the end, while the last one impacts fundamentally during the estimation phase. In the following sections we analyze each one of them deeply.

**User's role** In proprietary software projects, the user and the project client are usually the same person, department or entity. However, sometimes the client is not the one who will finally use the application. For example, in a company with different departments, the project client can be the company committee while users are employees of a specific department. Nevertheless we refer both of them indistinctly.

User's participation is critical for the project correctness. This participation is important and indispensable especially during:

- The requirements establishment that the project has to carry out.
- The budget approval (that will determine the term and the available resources).
- The validation of the functional documentation of the project.
- The configuration of standard solutions.
- The final validation of the project where adjustments between the application and the foreseen requirements are checked.

Usually, estimation and execution phases lack of client participation and commitment. In many projects, the user takes part only at the beginning, to determine initial requirements and at the end doing tests to evaluate adjustments between the requirements and the final application. The poor user/client participation during the central project phases is one of the reasons that makes the final application not enough adjusted to user real needs. Therefore, to guarantee the project quality, it is indispensable that the user takes part in all project phases. If this requirement is not achieved, it will be necessary to spend much more time and resources in order to adjust the application to the real user needs. Some methodologies make a special emphasis in the user's participation, for instance the extreme programming [1]. In order to avoid these problems, they integrate the user as a part of the development team.

**Communication** Another key point in the project development is the communication between team members, managers and users. Many projects suffer a clear lack of guidelines, communication lines and authority. The team members of project ignore, for example, the communication channels and how the relationship with the user must to be addressed. Neither they have clear each team member responsibilities. This fact causes important problems that affect all project phases for example creating a priority conflict, multiplying the change volume, elevating the resistance to the change or causing a lack of motivation. Finally, it increases the final project cost, postpone the terms execution and diminishes the final quality.

Project manager has the responsibility of the establishment of clear and effective communication channels, the definition of each team role responsibilities and the determination of client's participation.

Therefore, the project manager should know how to lead the project showing a clear vision and direction, generating trust and creating ownership feeling and desire of being a team member. It's also important to have charisma and, also, to have technical competitions recognized by the team.

A software configuration management plan makes easier communication during the project development. According to the IEEE [8], software configuration management is a discipline that oversees the entire life cycle of a software product (or family of related products). Specifically, configuration management requires identification of the components to be controlled (configuration items) and the structure of the product, control over changes to the items (including documentation), accurate and complete record keeping and a mechanism to audit or verify any actions.

Therefore, a configuration management plan allows all team members to know the state components in every moment, how they have been achieved, why a specific modification has been carried out and who has implemented them.

Many projects do not have a configuration management plan and this fact causes serious problems especially in the changes management and in the project evolution knowledge. All these problems hinder the communication. Therefore, it is important that the project manager defines from the beginning a configuration management plan.

**Objectives and reach** One of the main problems of project management is the clear specification and the project reach. For that reason, the estimation phase is a complex phase and one of the most difficult parts that generates a lot of errors. Thus it is necessary to readjust the project estimation several times during the project development. The success in this phase determines the rest of the project.

To make the estimation project, it's necessary to determine the work to be done and schedule the tasks keeping in mind the available resources. Also, it's necessary to consider that usually there are not very stable requirements since those are in continuous evolution. During the development project, new needs arise (this situation is known as phenomenon of growing requirements). In fact,

software is specified along its construction, although, from the beginning, in this estimation phase, it's necessary to make a first cost estimation.

A fundamental aspect in computer projects estimation process is the effort estimation which is a software attribute. This estimation is performed through measure units - metric - that allow us measure the effort. Furthermore, we need cost model estimations that determine systems and methods to obtain some values in hour/person of the different activities to produce a piece of specific software.

When models didn't exist, the project manager used to make cost estimation based on his previous experience in other projects. This fact usually gave subjective estimations as a result. Once software metrics appear we begin to have real data collected from already finished projects. Which such data, every task defined in a computer project is represented within a model. Using such models we can do estimation costs in software construction.

There are several estimation models that use the metric as estimate base: historical, statistical, theoretical, composite and standard based. The most accepted models are composite models that gather the advantages of statistical and theoretical ones. The most known and used composite model is the COCOMO - Constructive Cost Model -. This model is based mainly on statistical data, analytic equations and adjustments from experts opinion. It was created by Barry Boehm and published for first time in 1981 [3]. Also, it has given place to several versions like Ada COCOMO and the current COCOMO II [2] which is being elaborated together with the CSE - Center of Software Engineering - at the University of Southern California. Each one of these models remains valid while it will be applied to projects that follow the same paradigm, life cycle and tools for which the version was defined. For example the main objective of COCOMO II was to develop a cost estimation and planning model specially appropriate for the most current object oriented paradigms used during the 1990s.

However, most of the models used, arose three decades ago and nowadays they are quite obsolete to estimate current software construction projects because such new projects are developed in new programming environments, languages, tools and methods. Furthermore, the most important models are subjected to revision, but such revisions are not yet completely finished since there are not enough data to analyze. Therefore, in many cases, somebody decides to elaborate his or her own productivity standards picking up previous project data. As we have already mentioned once the initial estimation is established, it is necessary to revise and upgrade during the development project. During the project monitoring phase, the project manager must define a schedule that allows to manage the risk in a fast and flexible way without interfering the project objectives, terms and costs.

### 3 Free software project management

First of all, it's worth pointing out that not all free software projects are developed in the same way. In fact, each one has a particular evolution. When we

refer to "free software projects" we will generalize the properties which are more common to the most popular and successful ones. According to Eric Raymond's essay *The cathedral and the bazaar* [12], most software projects are managed in a quite more chaotic way than the traditional ones, similar to a bazaar where there is not any established authority which plans and controls the process. However, free software projects are also managed by milestones and planning. Most software projects require some kind of management, but this management is reduced to the indispensable minimum.

### 3.1 Properties

The management of free software projects usually follows reasons which are quite different from those of the traditional software development projects which have been pointed out in the previous section. In general, free software projects are developed in circumstances which are radically different from those of the proprietary software.

Initially, there is not any planning in free software projects, but as the project grows and new users and developers get involved, tasks must be planned and scheduled in time. Although there aren't established methods to perform this minimum planning, there are some tools which are used to perform error monitoring and also allow to organize and to distribute tasks for example Mozilla. Another example is MrProject that tries to be a simile of MsProject to project management. Therefore, it would not be appropriated to define the free software project management as a process. We can define it as a need which arises in a natural way and which reduces to the minimum and indispensable tasks. This minimum management is limited to:

- task scheduling and,
- monitoring and control of the project development state, especially error correction and detection.

### 3.2 Project management problems

We can point out three free software project properties that make it difficult to implant a traditional project management in this type of projects:

- A free software project is usually an informal process.
- A free software project has no limitations of either time or resources.
- A free software project has not a defined reach at the project beginning.

This section analyzes each one of these properties.

**Process type** A free software project is usually an informal process. The main reason for this is the fact that most of the developers, if not all, work as volunteers with no income. Therefore, the organization process of the team and the tasks to be made cannot be planned at the beginning.

Free software projects usually begin as an individual's initiative in order to satisfy some need or specific interest. This individual usually becomes the project leader at the beginning and, initially, his or her main task is to involve other developers in the project to build a development team. This team will vary during the project since new developers will be added to the project whereas others will leave. Even the leader or project promoter can leave the project in hands of a new leader to continue the development.

On the other hand, since the project arises from someone's idea, the final reach and requirements are defined while the project is developed due to the contributions and suggestions of all the developers. It is important to point out that these developers are also software users at the same time. In fact, the free software community users are considered like developers, since, although they don't write code directly, they carry out one of the most relevant tasks in the processes of free software: the code tests.

The fact that most of system developers are also final users speeds up the determination of the requirements. This fact guarantees that the final software is very well adjusted to the real needs, which, benefits the increment of the quality and of development speed. This circumstance is rarely found in the proprietary software development.

On the other hand, free software projects follow a short development cycle. This cycle is based on a quick and continuous release of versions which results in a quick error detection. Therefore, this cycle leads to a fast correction of the errors, reducing test time, costs and improving the final quality.

Finally, although the process is informal, free software development has established a configuration management procedure for the system components. This procedure, which is available to all team members, mainly regulates the detected errors and the changes which are performed. In this way, the developers know the current project state and which tasks have not been finished yet.

**Contractual project limits** Free software projects are not subjected either to terms or to limited resources from the beginning. In fact, they are not subject to any economic or contractual condition. Therefore, the possibility of a project direction change in any moment is very high. This fact has very positive effects, especially because it's not necessary to care about terms of execution. The project simply concludes when the developer team decides that it's no longer interesting to continue working on it, perhaps because all the functionalities have been achieved, perhaps because it's considered that the project is not useful.

In fact, the absence of execution terms allows for more margin to prioritize, above all, quality. This is one of the most relevant properties that is attributed to free software, and it makes its study specially interesting .

In spite of this, it's important not to forget the third variable that takes part in the management: the resources. Even without established terms, in order to achieve quality it's necessary to have enough resources during the scheduled time. As we have seen, the resources, fundamentally developers, work as volun-

teers without no income. They work in the project because of a high motivation. Therefore, it's a important task of the project leader to maintain a high motivation of the team during the project.

It's also important to point out that not having stable resources neither predetermined terms makes it difficult to schedule tasks, especially at the beginning of the project. It's possible to start planning when the project has a stable enough team and some agreed requirements. On the other hand, planning can establish which tasks are necessary to do and in which order, but it cannot assign them to specific developers. Participation is voluntary and, therefore, it's more convenient that each one chooses the tasks according to his or her preferences.

**Project reach** It's difficult to make an initial estimation, since the project arises from a individual's idea. Thus, the final reach and requirements are defined while the project is developed due to the contributions and suggestions of all developers.

On the other hand, we have also seen that developers are at the same time software users and that this fact speeds up the requirement determination. This guarantees that the final software product fits the real needs very well. Therefore, although it's difficult to think about an initial estimation, the fact that requirements are defined while the project is developed is not a major difficulty, since they are adjusted continually. This is another important differential characteristic of the free software.

#### 4 Free software contributions to improve traditional software management projects

As we have seen in previous sections, there are important differences between free software and proprietary software projects.

As described in first section, most frequent errors produced in the management have mainly three reasons:

- Not enough user/client participation during the project.
- Lack of communication among managers, the project team and client.
- Difficulties to determine the project objectives and the project reach.

On the other hand, free software projects have the following properties:

- There is a high user participation. In addition, users are usually voluntary developers of the project. Therefore users are highly motivated.
- The communication between the team project flow easily since user participation is voluntary, there are not tasks neither defined or assigned from the starting and a good configuration management is made.
- There are difficulties to determine the objectives and the project reach, but it's not too important to be able to make this estimation from the beginning since the project is not subjected to contractual relations that enforce terms and limit resources.

Thus, management difficulties produced in one and another environment either are different or have a different relevance and impact.

In spite of this, a detailed study about the way how free software projects are developed allow us to propose improvements for both the free software and proprietary software projects management. This implies an important software engineering advance.

Next, we expose some ideas about how to apply free software project practical to improve most problematic aspects in traditional project management.

#### 4.1 User/client participation improvement

##### - To integrate user in project team.

Obviously, you cannot expect that the proprietary software user (in this case also client) of the project would be also software developer. Although, if users are integrated in the development project and shorter and iterative development cycles are used, users will be able to make a continuous project monitoring in order to detect disagreements quickly. This methodology will improve development efficiency.

It can also break the usual trend in proprietary software projects development where user/client usually takes part only at the beginning to capture the requirements and at the end to validate the product. Therefore user/client would be close during the whole development process. In this way, users can define and detail requirements and assist any doubt that appears during the development in an effective way.

#### 4.2 Communication improvement

##### - To establish channels and communication procedures from the beginning.

Determining roles and responsibilities of each member team and communication mechanisms among them, and also between the team and the project managers and users allows any member team to know where to find needed information in a certain moment.

##### - To motivate the team.

To maintain all the team motivated by the project is essential so that everybody carries out the assigned work in the most efficient and most effective way.

##### - To negotiate configuration from the beginning.

To implant the configuration management procedure from the project beginning is a practice which is not carried out in many projects. However, since project documentation and requirements are also considered system components, it's very important that different versions and changes can be managed as they occur, making it possible to know the real project state in any moment. It's

not necessary to have developed code to start configuration management. It's essential, for example, that all team members know the mechanism to follow for code modification or to know where the last version of the requirements can be found.

### 4.3 Estimation improvement

#### **- To analyze projects in detail to improve the current models and reference standards.**

As described in the previous section, traditional models have an important problem because they are built starting from the experience analysis of some specific and old enough projects. Hence, their results wouldn't be completely applicable to any current project. For example, only little more than a hundred of some specific properties projects were analyzed in the case of COCOMO, [13].

Therefore, it's important not to depend on models exclusively. It's very useful to complete information provided by models with own productivity data from previous similar projects and with development teams with personal characteristics qualification known. Therefore, it's indispensable to have management documentation and the data of previous computer projects.

Free software opposite of proprietary software provides with new and important possibilities for a detailed study projects. On the one hand we can have code source for its analysis and, on the other, we have files, mailing lists and different version repositories, where all the project evolution is reflected. A careful analysis of these sources allows us to know free software construction process more thoroughly.

Robles [13, 15], proposes a methodology to analyze information contained in free software projects, to extract knowledge of analyzed experiences that it can be replied in other projects. On the other hand, Germans and Mockus [7], propose a system to automate measuring tasks and to analyze the data of free software projects.

Starting from real data, we could find a next theoretical model closer to reality that would allow us to elaborate more precise metrics to predict development speed and applications quality in a more exact way [14].

There are several studies guided to this objective, for example the study of Koch and Schneider [9] of the 2000 that tries to check the validity of classic costs prediction, and for example COCOMO model, by analyzing mailing list interactions and version repository of the GNOME project. This study concludes that, even when their application has many problems, the results are quite satisfying. There are other later studies of Kockus et. about the Apache and Mozilla projects [10].

These possibilities of free software analysis are a very good opportunity to software engineering to study these project types and to extract knowledge to progress traditional engineering methods and techniques.

## 5 Conclusions

This paper analyzes on the one hand, the traditional development management projects and, on the second hand free software projects management.

We have seen that, in each case, the specific particularities cause different management problems.

While in traditional software projects the main problems are fundamentally in:

- Not enough user/client participation during the project.
- Lack of communication among managers, project team and client.
- Difficulties to determine the project objectives and the project reach.

In free software projects it stand out the following ones:

- A free software project is usually an informal process.
- A free software project have not time neither resources limitation.
- A free software project have not a defined reach at the project beginning.

Finally, the analysis of main free software projects properties has allowed us to suggest some contributions to improve traditional projects management that fundamentally consist in:

- To integrate the user to the team project.
- To establish communication procedures and channels from the beginning.
- To motivate the project team.
- To negotiate the configuration from the beginning.
- To analyze the projects in detail in order to improve current reference models.

In this way, we see that free software projects analysis can bring new practices that help us to correct and to improve the main problems related to traditional software project management.

## References

1. Beck, K. (2002). Programming carry to an extreme Explained. Changes clasp. Firsr Edition. Addison Wesley
2. Boehm, B. W. i altres(1995). Cost Models for Futures Software Life Cycle Processes: COCOMO 2.0. J.D. Arthur i S.M. Henry (ed) Annals of Software Engineering Special Volume on Software Process and Product Measurement (vol. 1, pp. 57-04). Amsterdam: J.C. Baltzer, AG, Science Publishers.
3. Boehm, B.W. (1981). Software Engineering Economics. Englewood Cliffs: Prentice Hall.
4. Duncan, W.R. To Guide to Project Management Body of Knowledge (PMBOK Guide) San Francisco: Project Management Institute, 1996.
5. Feeny, D; Willcocks, L. (1999). Transforming IT-based innovation into business paouff, in Mastering Information Mangement. Davenport T: Marchand D.

6. Flowers, S. (1996). *Software failure: Management failure*. London: Wiley.
7. German, D.; Mockus, A. (2003). Automating the Measurement of Open Source Projects. In *Proceedings of 24th International Conference on Software Engineering (ICSE 2003)*. pp 63-68. Portland. IT USES. ACM Press. (Last acces: October 9 2003). <http://eecs.oregonstate.edu/icse2003/>
8. Standard IEEE/ANSI. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990 (revision and redesignation of IEEE Std 729-1983). IEEE Press. New York. 1990.
9. Koch, S.; Schneider, G. (2000). Results from software engineering research into open source development projects using public dates. Nr. 22, *Wirtschaftsuniversitt Wien*.
10. Mockus, A.; Fielding, R., Heerbsleb, J.D. (2000). To he/she marries study of open source software development: the Apache server. In *Proceedings of 22nd International Conference on Software Engineering (ICSE 2000)*. pp 263-272. Limerick. Ireland. ACM Press.
11. Olson, D.L. (2001). *Introduction to Information Systems Project Management*. Nova York: McGraw Hill
12. Raymond, E. (2001). *The Cathedral and the Bazaar. Musings on Linux and Open Source by Accidental an Revolutionary*. O'Reilly and Associates. (Last acces: October 9 2003). <http://catb.org/esr/writings/cathedral-bazaar>
13. Robles, G. (2002). *Igeniera del Software Libre. Una visin alternativa a la ingeniera del software tradicional*. [Last acces: October 9 2003]. <http://congreso.hispalinux.es>
14. Robles, G. (2002). *Los desarrolladores de software libre*. [Last acces: October 9 2003]. <http://congreso.hispalinux.es>
15. Robles, G.; Gonzlez, J.M.; Rye, J.; Matelln, V.; Rodero, L. (2003). Studying the evolution of free software projects using publicly available dates. In *Proceedings of 24th International Conference on Software Engineering (ICSE 2003)*. pp 111-116 Portland. IT USES. ACM Press. [Last acces: October 9 2003]. <http://eecs.oregonstate.edu/icse2003/>
16. Stallman, R. *The Free Software Definition*. [Last acces: October 9 2003]. <http://www.fsf.org/philosophy/free-sw.html>