Program and programmers evaluation using an annotation model: a case of open source systems

Robert Charles
Laboratoire Lorrain de Recherche en Informatique et ses Application (LORIA)
LORIA- Campus Scientifique, B. P. 239, 54506 Vandoeuvre-Lès-Nancy, France
Tel : +33383592087, Email : charles.robert@robert-scientist.com

Abstract
The open source initiative is a welcome development but some associated problems remain unresolved. 1. How do we know who is doing what in an open-source initiative? 2. How do we resolve problems connected to intellectual property? 3. It may be difficult to know the current status of a project 4. If an individual is interested in older versions of the open-source program, how can he access it? In trying to solve these and other problems, we propose an annotation model based on the user, the program and the time. We consider every contribution to the open source initiative as annotation. These are stored separately from the existing program. The newer program (old program + modification suggested) are stored with newer name. New program is stored differently in a database containing the old program. Two other databases store the profile of the programmer and the contributions (annotation) made by users. With this approach, it is possible to know who did what or what has been done with time. It is also possible to know the progress of work in the open source initiative. Intellectual property right is adequately resolved with this method.

1. Introduction

The open source initiative is a welcome development that has been able to awaken the potentials in software developers and reduce the monopoly of some software marketing strategy. The talents of thousands of young programmers would have been buried without the provisions and the “podium” of open source initiatives. Open source initiative provides a forum for “positive competition” among programmers. The participants tend to create a kind of club that encourages discussions on experiences related to an open source development. These experiences are shared through email, newsgroups, blogs and wikis. The open source initiative effectively reduces the monopoly of software vendors by bringing alternative software to the reach of users. Because of the openness of product resulting from the open sources, the use is promoted by the developers. The developers tend to advertise the product among their colleagues and acquaintances.

It will be impossible to qualify or quantify the contributions of open source projects in software development. The whole objective of open source initiative is to make programming experience open and accessible to all who may be interested. There are major advantages and major disadvantages of these open source projects. For example, Linux (Mandriva) was rooted on the fact that MINIX source codes were available (Hasan, 1999). GnuCash was initiated to provide forum to develop software for personal and
small business financial-accounting under Linux, BSD, Solaris and MacOSX (Gilbert, 2002). With the availability of GNU/Linux and Debian operating system, Apache web server, Mozilla Firefox browser, OpenOffice suite and other open source software, it is almost possible to have private or public functioning computing environment on all-open source platform.

Apart from the fact that the open source projects encourage the participation of wide range of programmer, it also encourages cross-platform application development. In the Apple world, the development of FreeBSD 5.0 and the Mach 3.0 microkernel, Mac OS X is said to be equipped for UNIX users (Apple development connection, 2006).

A work on open source project (Levesque, 2004), rightly emphasized the importance of documentation. It reported that “without adequate documentation, Open Source projects are inherently at disadvantage”. Our interest is to know if the inherent problem referred to here can be effectively managed with documentation.

2. Challenges and problems in open source projects

With the immense possibilities and advantages in open source initiative, there remain some challenges that we attempt to address in this paper. In a related work, Gonzalez-Barahoma (Gonzalez-Barahoma, 2000) identified three problems associated to open source program development. We will itemize some of the obvious problems observable in the development of Linux over the years. We state them as follows:

- No guarantee that development will happen
  There is no evidence that an open source project will receive attention. One of the reasons why an open source project may not receive attention is the fact that it is open and free. In the subconscious mind of man, what is good must be costly. A freely accessible system may not after all valuable. The popularity of a project or its acceptability can depend on the personalities behind the entire project. If the “supporters” of the project and forceful, they may be able push their ways into the mind of their audience.

- There may be significant problems connected to intellectual property.
  The problem with intellectual property is evident because, the concept of open source does not encourage patent of contributions. FOSS licences was reported as been used by many software projects (Gonzalez, 2005)(Lemyre and Willemant, 2006). According to (Wheeler, 2005) Free Open Source Software (FOSS) are programs whose licences give users the freedom to run the program for any purpose, to study the program and to redistribute copies of either the original or modified program (without having to pay royalties to previous developers). When software is redistributed, how do we attribute authorship to the published work?

- It may be difficult to know the current status of project
  Since the development effort is meant to attract as many as possible, many people are expected to be involved in an open source project development, this may make it very difficult to know exactly the current situation of the project. To know who is doing what is almost impossible. It is very possible to have repetitive activities in this kind of system.
How can we prevent overwrite? Some interesting functions, procedures can be overwritten by some seemingly less important procedure. It may even be interesting to keep some functions and procedures despite the fact that they are perceived to be less effective (from a particular point of view). For example they may be considered less effective in terms of speed but they may be of greater importance in terms of security.

A function, procedure or subroutine may be interesting to a group of users while not important to other group of users, how do we reconcile the differences in interest in an open source development?

3. Annotation for open sources

We attempt to resolve some of the above stated problems with the use of an annotation model. Annotation was defined as an act of interpreting or evaluating a document. Interpretation or evaluation is of a specific context and is expressed on the document (Robert and David, 2006) Annotations normally take a different form and look compared to the original document. The different in look may be noticeable in form of character used, font, style, color or additional signs and images that do not form part of the original document. A document for annotation can include various entities like punctuations, words, images, artefacts terminologies, phrases, sentences, passages, collection of homogeneous documents, a collection of heterogeneous documents. Two types of annotations can be seen in respect to these types of documents. An annotation can be made to classify the document like in a collection of documents, entire document. In the case of sentences, phrases and section of documents, the objective is generally to restructure the document. We can effectively describe contributions to programming efforts in Linux as annotation.

Contributions to programming effort can be seen as annotation meant to restructure the existing program. It is the same as when an editor makes annotations by the side of document for publication. He may “markup” document for type bold, underlining, emphasizing etc… In the case of programmers, they “markup” programs with their function calls and subroutines. The essence of their program is to change the structure of existing program. Call this “structure” or functioning, performance or behaviour we are still talking about the general objective of an annotation.

In a generalized annotation system shown in figure 1 as in the case in most annotation system, the concerned is the production of annotation based on the document sent to a parser (an annotation engine) and then the result. The question of “why” the annotation was made is usually not addressed. The question of “how” an annotation is made and the technology behind it is the general concern.

The basic components of most of annotation systems can be divided into three (a) the document, (b) the annotation parser and (c) the resulting annotated document. The methodology and parameters for storage is not even applicable in some of the existing tools. In some cases, for example Amaya, provision is made for storage of annotation in either a local or a remote location (Kahan, 2004).
A research group in France presented a tool called Dinosys (Desmontils et al., 2004) that was meant to apply annotation as a mean of sharing resources among students. The approach in Dinosys was more of explorative and there was no provision made for the evaluation of students’ participation. Another work (Vasudevan and Palmer, 1999) proposed that an annotation framework should be customizable to support variety of document management function, and to be non-intrusive to enable easy insertion into enterprise Intranets or the public Internet. This approach was good but not good enough for evaluative purpose.

AMIEX is Annotation Model for Informatics Events and EXceptions. Events and exceptions are specific programming terminologies. We can see it as Annotation model for Computer Programming. AMIEX can be viewed as a specific application version of the annotation system described in the work of (Robert and David, 2006). We developed AMIEX to (a) assist in evaluation of users’ (programmers’) contribution to the development effort and (b) to assist in monitoring progress in a Linux itself.

4. Architecture of AMIEX
In this section, we will discuss key features and the architecture of AMIEX. AMIEX is an extension of AMIE (Annotation Model for Information Exchange) which is a conjunction of annotation characteristic based on observations and needs in an information system. The basic components of AMIE are the user (a programmer, a potential contributor), document and time. Three important facets can be observed of an annotation system for evaluative purposes. It is expected that evaluation would be made based on information aggregated from a set of document and time.
In all, AMIEX model attempt to recognize five parts (a) a user is a member in a collaborative workspace (b) Program repository (c) annotation database (d) contribution (e) time. We will specify details of some of these components necessary for us to be able to arrive at a model to evaluate the development in an open source initiatives with particular emphasis on Linux.

4.1 User

The user is identified with the following parameters:

- Usercode (which can be his official school code)
- SurName
- FirstName
- PostalAddress
- EmailAddress
- City
- Usergroup (programming or academic group he belongs)

4.2 Program Repository

Program repository is a database of program. It may also be a link to program and not the programs themselves. Program names must be unique. It that can be identified with the following parameters:

![Figure 2: Conception of AMIEX](image-url)
4.3 Annotation database

- Date of program
- AssociatedCreator (same as usercode in 4.1)
- ProgramReferenced (Same as Programcode in 4.2)
- AnnotationCode (specific to the contribution)
- DateandTime of annotation creation
- Objective of annotation (Personal, Answer to a question, new concept, …)
- AnnotationType (Delete, Update, Addition)
- Annotation (Suggested proposition: This is the full program or function, call, applet, script, method suggested by user)
- Associated file (URL or path)
- Location (Node to place proposition (Module name, Calls, Function, Subrouting etc… which consist of module name, line number in module)
- CallParameters (Calls to the module)
- Returns (Output from the module)

4.4 Workspace

- SessionCode
- AnnotationCode (Same as in 4.3)
- UserCode (Same as in 4.1)
- OlderProgramCode (Same as in 4.2)
- NewerProgramCode (Modification to OlderProgramCode)
- DateandTime
- A line summary of work done (this can be used to summarize work done by date, realize that this can be null if nothing was done)

A user in a collaborative workspace will normally have identification (usercode). He uses the identification code to request for a particular program from the software repository. The user is granted that program as a common task if he is a member of the collaboration. He is free to edit, modify the program. Instead of sending the modified program back as a replacement in the repository, we propose that, what he added or deleted is stored as a parameter of his contribution. Realize that a newer version of the modified program is created with a newer version number while the older version of the modified program is kept in the repository.

The newer program is returned to the repository with newer version code. User’s identity is stored in the workspace with date and time and the code of the program he worked on (the older and the newer codes). A code is created for the proposition he made (in form of annotation). Annotation database receives the proposition as “annotation” and other parameters linked to the proposition.

5 Application to open source evaluation
With this model, it is possible to know the frequency of participation of users (from WorkSpace database). It is possible for a user to decide exactly what version of program he will want to work with since newer programs are always created from existing ones. He can do this by making reference to a particular date. We can evaluate the methodology of each user from the annotation database. This is because a contribution of each user is stored separately independent of the global program. It is possible to monitor the growth of the program (from the program repository). We can also study the growth of each user from a set of his contributions in the annotation database.

From the WorkSpace database, we can see the period that is most favorable for particular user of the entire participants. Realize that summary of what has been done and date are stored in session database. A contribution that is judged “unnecessary” can be identified and eliminated because there is no overwrite of program. It is easy to understand the specific issue(s) learnt by user by comparing his initial “annotation” with the latest. This can be a way of honoring users who made substantial contribution the open source initiative. It can be seen also if a user is static in his activity. Comparison can be made across years, across groups of users (say by country) and across period of time.

6 Conclusion
From the propositions above, we are able to answer the questions concerning intellectual property that is recurrent in open source system development. The current status of project can also be evaluated. There is no issue of overwrite since each program is saved independently of the other. Individuals can decide what functions or modules are interesting to him and use that as starting point for his own use. Practical implementation of these proposition is on course.

Bibliographies


