HOW ARTIFACTS RULE WEB BASED COMMUNITIES: Practices of Free Software Development

Stefano De Paoli*

Department of Sociology and Social Research, University of Trento, Piazza Venezia 41, 38100, Trento, Italy
E-mail: stefano.depaoli@soc.unitn.it *Corresponding author

Vincenzo D'Andrea

Department Information and Communication Technology, University of Trento, Via Sommarive 14, 38050 Povo (TN), Italy
E-mail: vincenzo.dandrea@unitn.it

Abstract: This paper seeks to understand how complex tasks are accomplished by large web based communities. In order to do so we consider that technologies may embody organizational rules, hence becoming the core of coordination efforts in such communities. Our analysis is based on the concepts of translation and inscription taken from the Actor-Network Theory, while the data has been gathered using an ethnographic approach. We base our observation on the case of a Geographic Information Systems (GIS): the Geographic Resources Analysis Support System (also known as GRASS) is a system developed by a small group of developers but sustained by very large users' base, located all around the world. In particular we focus on small explanatory examples of everyday development activities of the GRASS community in relation to the internationalization process. We focus on artifacts, such as the Concurrent Versioning System and the GRASS source code, that greatly affect the everyday activities of the community. The main result of our research is the illustration that complex tasks are accomplished by large web based communities thanks to many small contributions which are shaped and organized by the rules embodied in artifacts.

Keywords: Bazaar style of development, programming practices, translation process, coordination.

Biographical notes: Stefano De Paoli was trained in sociology, he is now PhD student in "Information Systems and Organizations" at the Faculty of Sociology - University of Trento (Italy). His research interests comprises several aspects of Information Systems in particular phenomenology of software development and software licenses.

Vincenzo D'Andrea is an associate professor at the University of Trento, where he teaches Information Systems. His research interests includes service oriented computing, free and open source licensing, virtual communities.

1 Introduction

The purpose of this paper is very easy to state: we seek to understand how complex tasks are accomplished by large web based communities. In order to reach this goal we consider that technologies may embody organizational rules, hence becoming the core of coordination efforts in such communities. We then have to answer two questions: first, what is new in this statement? and second, why is so important to speak about technologies?

One well known case where web based communities realize complex tasks is that of Free/Libre and Open Source Software – hereafter FLOSS. This definition is combining the view of Open Source Software, emphasising the advantages of being able to read source code, and of Free Software, emphasising the superior value of freedom that characterise software seen as knowledge (Stallman, 1985; Open Source Initiative, 1997). FLOSS projects give us an extraordinary opportunity to study how complex artifacts, such as the well known operating system kernel LINUX, are developed in a cooperative and ongoing manner by large communities. According to the well-known paper by Eric Raymond *The Cathedral and the Bazaar* (1998), FLOSS communities may be conceptualized as Bazaars of different agendas and approaches, where many skilled developers and users collectively participate in software development and related activities. Bazaar communities' case studies then demonstrate that complex tasks are accomplished via the Internet by apparently unmanaged and decentralized efforts of many people and organizations.

In Raymond's work, FLOSS communities are conceptualized using the metaphor of the Bazaar as opposed to that of the Cathedral. This paper may be considered as an anthropological analysis, aimed at understanding what makes the FLOSS world work, and why FLOSS communities are able to produce extremely high quality software, in spite of constantly violating all the rules of mainstream software engineering (see for instance Brooks, 1975). In order to provide an explanation, Raymond set up a sort of contrast between two opposed styles of software development. The first is the conventional development style, defined by Raymond the Cathedral Style. This style is characterized by a tight specification of the objectives, and by small project groups run in a hierarchical, authoritarian manner. There are long release cycles of the software and one single management, mastering the entire peer review process of the software source-code development. On the other hand the Linux world seemed to Raymond to be a much more peer-to-peer, decentralized market, which he defined as the "Bazaar Style". In this style there are very short release cycles, and constant solicitation and feedback from people who are formally outside the project, providing an intense peer-review process that actually seems to be able to generate complex and high quality software.

Raymond's considerations have led researchers to take seriously the problem of the Bazaar style of development, as opposed to the Cathedral one. As a consequence, many efforts have been devoted to study the question "Why do people participate in the Bazaar?". It may be easily understood that if the procedures for creating software innovation are not specified in advance, as in the mainstream software engineering practices, then it becomes difficult to understand why so many people participate in FLOSS development. In that respect Raymond himself argues that the motivation for sharing the software is not only purely altruism. Rather, there is at work a social process that operates in FLOSS communities that helps to sustain a gift culture.

Discovering the motivations that drive people and even organizations has been one of

the main goals of several studies of FLOSS. For example, Himanen (2001) argued that FLOSS programmers are driven by a common set of ethical motivations recognized as the Hacker Ethic; Lerner and Tirole (2002) argued that expert programmers' participation in the Bazaar is motivated by the rewards that they may obtain in terms of career advancement; Lakhani and Wolf (2005) undertook a large survey in order to quantify the weight of extrinsic and intrinsic motivations for participating in a FLOSS project; and the list of these studies may go on (Lindenberg, 2001; Hertel et al, 2003).

Recent developments in the sociology of FLOSS have noticed that the large interests devoted to study the motivations has unfortunately hidden other relevant aspects of FLOSS communities. Many have noticed that the studies on motivations assume the existence of an unified FLOSS community whose participants share the same values, motives and approaches. Sociologists argue that studies on motivations seem to explain only some elements of FLOSS, lacking of an understanding of the dynamics of the Bazaar Style of development (Tuomi 2001, 2004; Lin 2005). For example, according to Lin (2005) «From a sociological point of view, the core elements that make the FLOSS development dynamic is the multiple cultures and a collective set of practices found across the social worlds that are not fixed or ordered, but rather, indeterminate, flexible and contestable, that shape the FLOSS innovation. It is obvious that the FLOSS communities are much more dynamic than the paradigm given by Raymond or the ones suggested by others».

Along this line of reasoning we move toward the idea of a "Bazaar of practices" where the focus is on everyday life activities of FLOSS communities. In other words, we do not seek to focus on "people doing" something inside an unified view of Bazaar community. Rather, speaking about practices we seek to move our focus on the "doing" itself (Gheradi e Bruni, 2007) and on the messiness of everyday practice, where this "doing" may be attributed not only to human actors but also to technologies and in general to non-human actors.

There are some relevant studies that seek to understand how "*Artifacts Rule"* FLOSS communities' activities (Lanzara and Morner, 2005; Shaik and Cornfrod, 2004). For example, Lanzara and Morner (2005, p. 67) argue that «technology can replace formal organizational rules and structures in the coordination and governance of complex activity systems», hence becoming a fundamental pathway to the understanding of collective task accomplishment, coordination and knowledge making process. In order to enquiring the peer-to-peer, decentralized FLOSS development, the role of technologies and artifacts become then fundamental.

In our work we seek to follow both the indications of Lin (2005) and Lanzara and Morner (2005). Our aim is to gain more empirical evidence about the organizational role of the source code of computer programs. The source code may be defined as the set of instructions that developers write when creating a computer program. In FLOSS projects source code has to be considered as an evolving artifact, which is shaped and reshaped in every day programming practices. Source code is at the centre of many activities of FLOSS communities, thus constituting an artifact that channels the attention of the many actors participating in the Bazaar. Moreover, the source code is managed, written, and rewritten using tools that greatly affect the development practices. During the development of source code,, technical knowledge is inscribed into it, hence stating what the computer program must do in a quite literal sense.

The paper is organised as follows: in the second Section we describe our theoretical perspective; then we describe the scenario of a FLOSS community; in the fourth Section we briefly describe the methodology of our work; we then move toward a rich description of the artifacts' rules; finally our conclusions close the paper.

2 Theoretical approach

Our theoretical approach moves from two related considerations. First we do not consider the term "community" as a static or unified element, rather we consider a community as something that is shaped in everyday practices of the many actors participating in the community life. Second, we do not consider technologies as a neutral medium of human activities; rather we claim that technologies and artifacts must also be considered as actors participating in the community life. In both cases we generally agree with the view of Actor-Network Theory (Callon, 1986; Latour, 1987, 1992 and 1999; Law, 1987; Akrich, 1992).

An Actor-Network is a network that is built by assembling various heterogeneous elements, both human and non-human, so that they can work together (Callon, 1987). The main goal is to look at the ways in which technological artifacts and human beings act upon each other symmetrically in order to produce action. According to Latour (1999, p 180), through this "material relationalism" we can learn «to attribute-redistribute-actions to many more agents» because we can shift our attentions to hybrid actors.

Central to this approach are the notions of translation and inscription. A translation is the active process of "actor-networking" practices, and may be conceptualized as a series of «negotiations, intrigues, calculations, acts of persuasion and violence, thanks to which an actor or force takes, or causes to be conferred on itself, authority to speak or act on behalf of another actor or force» (Callon and Latour, 1981). A translation process may be seen as composed of four interrelated moments (Callon, 1986):

Problematization, when an initial ensemble of actors, the spokesmen of the translation process, define a problem to be solved and a solution to this problem. This definition usually takes into account the heterogeneous humans and non-humans elements and their position within the composition of the actor-network. The aim of the actor-networking practices is to convince the various human and non-human elements to play the role assigned to them within the actor-network.
Interessement, the spokesmen must interest the elements to play the roles assigned to them. In that sense, interessement is the act of dividing (inter-esse) the elements which act against the spokesmen's solution from the others. It is especially thanks to certain devices that it possible to divide such elements. Such devices (devices of interessement) incorporate features that allows the separation of good and dangerous elements.
Enrolment, are the negotiations and trials of strength which could allow the success or the failure of the interessement. The enrolment has the goal of anticipating what other elements, humans and non-humans, may do in opposition (*anti-programs*) to the spokesmen program;
Mobilization of allies is the ability to definitely stabilize the heterogeneous associations of elements and make them acting as a whole. In that case, the spokesmen are able to speak in the name of the whole actor-network. Nonetheless, spokesmen may even fail to enrol the elements and a translation process cannot succeed.

In order to illustrate how this concepts works we can refer to an example. Latour (1992) describes the case of a hotel manager who has the following problem: the guests forget to leave their room keys at the reception desk when leaving the hotel. In order to solve this problem, the hotel manager tries to convince the guests to leave the keys, telling everyone "Please, bring back your keys". Nonetheless, the verbal statement does not achieve the goal of interesting the guests, and they still forget to leave the keys. The guests are not enrolled in the Actor-network of the Hotel Manager. As a second step the hotel manager decides to use a much more convincing strategy. He/she decides to put up a sign at the desk: "Please leave your room key at the desk before you go out". The

written sign produces some results, and some guests bring the keys to the desk. Nonetheless, the majority of the guests still leave the hotel with the keys in their pockets. Finally, the hotel manager introduces a material innovation: he attaches a metal block to each key. This results in a decisively convincing act: the metal weight is uncomfortable and cannot easily be forgotten in the pockets. The guests are forced to leave the key at the desk by the characteristics of the material innovation.

The verbal statement, the sign and also the metal weight, may be seen as an inscriptions. The notion of inscription is the act of inscribing in an artifact a framework of action which defines «actors with specific tastes, competences, motives, aspirations, political prejudices, and the rest, and assumes that morality, technology, science and economy will evolve in particular ways» (Ackrich, 1992, p. 208).

It is thorough inscription that an actor causes to be conferred on itself the authority to speak in the name of others. After the inscription of the metal weight there is no more need for the hotel manager to tell to every guest to leave the key at the desk. The metal weight will do this work for him: the weight is able to act upon the customers, forcing them to bring the key to the desk. The metal weight has allowed the manager to translate all the actor-network composed by the hotel, the customers, the keys, the hotel managers, and so forth, from the "not leaving the key" position to the "leaving the key" position. This translation process is then a confrontation between the programs: leaving versus not leaving the keys. The first program obtains more and more strength by loading itself with more and more inscriptions. A verbal statement, a written sign, a metal weight. As a consequence, the first program becomes more convincing and it is able to enrol the element: the hotel manager is able to mobilize the whole actor-network.

3 Setting up the scene

Our observations are based on the case of a FLOSS Geographic Information Systems[1] (GIS) known as Geographic Resources Analysis Support System (GRASS). The GRASS project started at the beginning of the '80 as a small development project of the United States Army. In 1996, the US Government decided, for several reasons, to stop the development of GRASS and invited the users to migrate toward other GIS solutions and proprietary version of GRASS (USACerl, 1996).

After a couple of years of transition, a new GRASS development team (GDT), mostly composed of volunteers not related to the Army, came up with the aim of re-launching the GRASS development. The GDT today is composed by an international group of developers affiliated to different institutions, companies and individuals. In order to give some ideas about GRASS, we may recall that the last version of the system (GRASS 6.3) is composed of more than a half million lines of source code written in C programming language. The development is led by a world wide team of developers (about 38 people, of which more than half are active) and sustained by an estimated user base of 25-30000 users[2], distributed all around the world.

The way in which a large project like GRASS is developed may be of course a central issue for the community. According to the development coordinator/leader of GRASS there are motivations behind the contribution to FLOSS development:

I think the driving force is that people see that the others are using their own development and they are grateful to them and so forth. But also that other people are solving problems for you, and this is some giving, some receiving, an interchange which is pretty, pretty nice.[3]

---

1 A geographic information system is a system for creating, storing, analysing and managing spatial data and associated attributes (see for instance Wikipedia article on GIS).

2 The number is an estimate based on downloads of the GRASS software.

3 Passages in bold face are our translations of the transcription of a talk given by Markus Neteler

In other words the motivations for participating in FLOSS development are not only pure altruism, rather there is at work the "social mechanism" expressed by Raymond and others. Nevertheless motivations are not all that constitute the essence of participation in a FLOSS project:

There is no rule, if you would like to contribute one minute per week it's OK. But the most important thing, more important than the time devoted, is the way in which the contribution is given.

Luckily, there is no need for the researchers to justify the shift of the focus from motivations toward practices of FLOSS development. Participants engaged in FLOSS projects are already aware of the importance of development practice. In fact, within the participation there is at work a confrontation, an opposition between two programs: "receiving good contribution versus receiving bad contribution", which seem to be not much dependent from "why" people contribute. In other words, for FLOSS developers the manner in which a contribution is received is more important than receiving whatever contribution:

Let me give you an example. Somebody starts to complain about a badly written statement in the documentation and suggests "write it better" or "Please explain". Generally this is almost useless. It is good that they pointed out the problem – it is good because they do something. But it would be better if they give suggestions for changing the statement.
[...]
I try to, let's say, to tell people "if possible, send me the differences against the CVS". With the power users this work well. They send me maybe two lines changed in the documentation, [...] or something textual. [...] And for me this takes 5 minutes for checking it. When the contribution is diffuse (related to several portions of the system) **it becomes expensive for me and others and so occasionally it is ignored. It could stay in the bug-tracker for maybe a year .**
[...]
every single contribution, even a very small one, properly made and easily integrable, is ten times more valuable than the comments.

These considerations give us a fresh look at contributions and peer-review dynamics of FLOSS development. In particular in the opposition between "god versus bad contribution" it emerges as a problem to be solved: a contribution needs to be easily integrable into the system. Moreover receiving a bad contribution is also not useful because this means wasting time in order to integrate the contribution into the project:

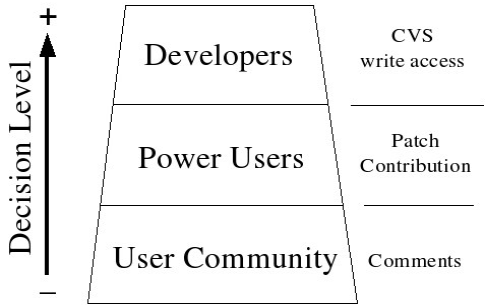
When an interaction requires some work from us, from us who have write access to CVS ... When, for example, somebody sends me a new command which is very badly written [...] that takes one hour just for cleaning it, just for testing it. So, before distributing this contribution in the public mailing list, I tell them if they can resubmit a better version. [...]

The solution to these problems lies in the statement "Send me the differences against the CVS". In the next paragraph we will carefully explain what a CVS is and how the source code development practices related to this artifact. The GRASS project explains this in the "Code Submission" section on the the web site. We will thoroughly discuss this subject in Section 5.2. For the moment, it is enough to notice that the statement is related to the process of newcomers joining a FLOSS project, who must demonstrate a certain degree of technical expertise (von Krogh et al, 2003). So, it is clear that into the statement "send me the differences against the CVS", several elements are indicated as part of the problem's solution: "power users", "CVS" or "us who have write access to CVS" and so on. But who are these elements? and what role do they need to play in the actor-network?

Developers, users and CVS

According to the development leader/coordinator of the GRASS project, the organizational structures of the GRASS Development team may be represented with the diagram in Figure 1. According to this scheme, a GRASS developer may be assumed as one who has "write access to CVS". In many FLOSS projects, software development is managed using technologies called Version Control Systems (VCSs). There are several VCSs solutions and the most known systems are probably the Concurrent Versioning System, Subversion and BitKeeper. VCSs support knowledge creation, learning, and innovation by providing a structured, updateable repository of source code versions, patches, and documentation. By a shared access to such systems developers and users are able to read, change and distribute code (Lanzara and Morner, 2005; Shaik and Cornford, 2004).

Figure 1 – Organizational structures of the GRASS Development Team, adapted from Neteler (2006).



Speaking about GRASS, the Concurrent Versioning System – hereafter CVS - became an active part of the technical framework of the project at the end of 1999 (Neteler, 2001). In order to describe CVS, we first have to describe what constitutes a project such as GRASS. The artifacts related to the project are a number of files, each containing a portion of the source code of the project's programs (we shall see that these files also include text related to the project documentation, user manuals and so forth). Each of these files is the result of the work of several members of the project, so undergoing many changes that are called "versions". The role of CVS and other VCSs is to manage a repository of all the versions of the project files. CVS is based on a client/server architecture: a server stores the current version(s) of the project and its history (all the previous versions of the system and submission comments), and clients connect to the server in order to "check-out" (read) a complete copy of the project, work on this copy and then later "check-in" (write) their changes (Wikipedia, article CVS). CVS keeps track of all work and all changes in the set of project files, and allows the GRASS developers to collaborate on-line in the software development processes.

Two of the main concepts of all VCSs, including CVS, are the notions of diff and patch. Storing a complete set of all the project files for each system version would not be feasible in terms of disk space needed, so VCSs use a different strategy. Instead of storing the full text of the new version of a file, they store only a description of the variation between the previous version and the new one. This description is coded using simple tags for identifying the text removed (for instance a "-" sign), the text added (for

instance a "+" sign), and few other operations. A "diff" is such a description of the differences between two versions of a file (quite similar to the report provided by word processors when comparing two versions of a document). With the same description of the differences, it is also possible to "operationalize" a diff by generating a set of instructions describing how to transform a version of a file into a new one. This operational description is called a "patch". In computing a "patch" is a small piece of software, designed to update or fix problems with a computer program. When somebody wishes to submit a contribution to the project, a patch is a useful way to submit a compact description of the proposed modification to the project source files. Of course, in order to maintain the consistency of the whole project, the patch must be prepared and applied to the last version of the project itself. We will provide examples of patches later in this paper, see for instance Figure 2.

Looking at technologies like the CVS reveals fundamental characteristics of the FLOSS communities' dynamics, especially in relation to existing hierarchies in development communities. There are two important aspects to consider. First of all, write access to CVS may be limited to a restricted group of people: people who want to "check-in" their source code into the CVS, and then actively participate in the development, should be able to demonstrate their ability in development practices before being allowed to have write access. Thus, while everybody has read access to the CVS server (the check-out), only the developers can also write into it. Second, as we already noted, it is a mistake to think that only software may be managed using the CVS (Fogel, 2005). In many cases it is useful to manage the whole project files, this means that also web pages (e.g. the project portal), documentation, and so forth may be managed using the CVS. As long as a file contains text, its versions can be managed by CVS. In addition, also program executables are stored in the CVS for convenience.

Developers may grant a write access with the aim of reducing their time and effort dedicated to integrate others' patch. Those who contribute to the software development, but who do not have write access to CVS have been indicated as power users. A power user is a computer user who can utilize advanced functions and programs that are outside the reach of normal users due to the complexity and advanced knowledge required to perform these tasks (Wikipedia, article Power User). According to Figure 1 a GRASS power user may be recognized because he or she is a person who send patches or detailed bug/enhancement reports.

Patches include fixing software bugs, enhancements, new functionalities, and improving usability or performance (Wikipedia, article Patch). It is on the peer-review of power users' patches that write access to CVS may or may not be granted. In principle, write access could also be revoked but in the seven years of CVS usage in the GRASS project this has never happened. We aim now to look more in-depth at the peer-review process itself.

4 Methodology

Before entering into the everyday practice of the GRASS community, we will devote a few words to data collection. Activities of FLOSS communities take place almost only on the World Wide Web (WWW), for this reason most data has been gathered from the WWW. We have based our data collection on the assumption of following the construction of technology in action (Latour, 1987) thorough the WWW.

The data collection has been based on ethnographic observations applied to the WWW (Hine, 2002; Wittel, 2002). We would like to stress that the ethnographic approach applied to the WWW seems to be useful in order to study how web based technologies can be considered as cultural artifacts. Using an ethnographic approach we have been directly able to observe the everyday practices of GRASS, following in

particular the controversies around the role of artifacts in community activities. In general we consider that small events, everyday activities and situated controversies are very explicative of the cultural and social character of technologies. Observing such practices and activities result then in a better understanding of how complex tasks are accomplished by large web based communities.

Relevant documents have been gathered from the main web site of the GRASS project (http://grass.itc.it) and from mailing lists' archives. Our focus on actor-networking practices of different heterogeneous actors, both human and non-human, has lead us to take into account the debates and discussion taking place within the mailing lists of the GRASS project. We have followed several discussion threads about programming practices and patching activities. Our focus has been mostly directed to messages sent by programmers who were not regular members of the development core team. These threads cover approximately the last two years of GRASS development. One interesting feature of mailing list threads is the pattern next-next of the messages, which allow us to see mailing activities much more as discussion, where objections and communication take place retrieving previous messages and answering them point by point. We have also investigated the source code of GRASS in order to fully understand the matters of the mailing lists discussions. To do so we have made large use of the web based GRASS Source code browser.

5 The case of i18n artifact rules

There are various scenarios where we can look at how a good contribution is realized. In this paper, we have chosen to focus on the process of "internationalization" (i18n) of the system. Internationalization is a means of adapting products for environments different from the one of the initial developer(s), especially other languages, nations, and cultures. i18n aims also at internationalizing software projects providing translation features, at the level of the interfaces and of the messages for the users about the systems processes. The problem to explore here is how a good i18n contribution is realized.

In making software products, internationalization and localization pose challenging tasks for developers, particularly if the software is not designed from the beginning with these concerns in mind. A common practice is to separate textual messages from the program code. Thus, supporting a different environment, ideally, only requires change in those separate resources without code modification. The development team needs someone who understands foreign languages and has a technical background; such a person may be difficult to find. Software libraries that aid this task are available, such as gettext[4] (Wikipedia, article i18n).

The GRASS project has started a lively "i18n" process (Masumoto et al, 2005) justified by the fact that «the great worldwide interest in GRASS suggests value in translating GRASS messages to languages other than English» (http://grass.itc.it/devel/i18n.php). There are few languages (18 to be precise) that have begun to be supported. Table 1 gives an idea of the most translated languages in the System Modules.

Table 1- The five most translated languages in GRASS Modules, retrieved and adapted on April 26th, 2007 from http://grass.itc.it/devel/i18n.php

[4] gettext is the GNU internationalization (i18n) library. It is commonly used for writing multilingual programs.

Author

Languages	***Translated messages***	***Partially translated messages***	***Untranslated messages***
Czech	4210	534	160
German	3451	798	655
Vietnamese	3411	1024	469
French	2923	1273	708
Spanish	2363	1698	843

One of the programming practices of the i18n is to add a gettext macro which is based on placing an underscore, square, and a closed square _(" ") after the message, when the message is included in the source code text. This macro has the goal to mark the message, for example the message _("Hello world "), ready for translation. During the compilation of GRASS special software extracts this piece of string/message and puts it into a different ASCII file (*.po files). For each language this file exists and in this way it is possible to have the English messages and the translated messages stored in such files.

CVS rules for contributors

Every translation file is also maintained in CVS like the rest of the project. This may imply the need for GRASS developers to give CVS write access to the translators who handle and update these files. It is also easy to recognize that there may be at least one translator for each language, and this may imply that many people may be a candidate to have a CVS write access. If we look at the CVS comparing it with another Version Control System, we can notice how the CVS introduces interesting rules that act upon the selection of contributors:

In CVS you do not have any possibility to restrict access. This you can do in the "SVN" software Subversion [...]. In Subversion you can lock individual directories, in CVS you [can only] grant access to everywhere. If I grant for example access to the GRASS Newsletter, which is also managed in CVS. This person directly has access to everything. So, we have to trust the people, that's important.
[...]
We are basically selecting the people. We say, you are interesting, you are contributing a lot, it would be easier for me if you could write directly to CVS, because otherwise people are sending everything to me and I have to check it in.

The CVS then embodies a rule for the enrolment of people in the "CVS write access circle". Because it is not possible to lock some parts of the projects managed in CVS, for example the GRASS Newsletter, whoever has write access (e.g. the editors of the Newsletter) have then access to the whole project. Who can have write access to CVS is then selected on the basis of trust but under the regulatory obligation of the CVS itself: the rule inscribed into the CVS forces the developers to select contributors.

In order to better understand how the CVS selection's rule is an active part of the everyday practices of the GRASS community, we propose to observe a real example coming from our ethnographic observations. The following thread from the GRASS Translation Mailing Lists (June 1, 2006) shows how the CVS rule of selection act upon the dynamics of i18n. In this message a translator was complaining about the CVS write access policy of GRASS developers, asking for a write access only to a limited area of the project: the "locale" directory where the translation files are stored. The answer from a developer explained how the concept of CVS makes this impossible:

> would it be possible to grant write access just to the locale directory on CVS?
> That should solve the issue of translators write access to CVS.

The concept of CVS doesn't support it: you get access to the entire system (which worked well in the past). Once we migrate to SVN (not sure if and when), this will become easy to maintain.[5]
[GUML[6], MN, discussion June 1, 2006]

The developer indicated that a solution may be to change the Version Control Systems from CVS to Subversion, in this way people could have access only to a limited area of the project. Nevertheless, in the same email message the developer also indicated the requirements for obtaining write access to CVS:

*I am sure that we can grant access after a short "dry" round of 'CVS diffs'. It is just that *you* tell me that you are willing to*
learn the few commands.
[GUML, MN, discussion June 1, 2006]

This example shows how the rules contained in CVS play a fundamental role in organizing activities of the GRASS community. This rule organizes the i18n activities and forces the selection of the translators: since giving write access to CVS means opening up the whole project to this translator, before enrolling him the developers want to ensure that his contributions are given as "good contribution". The CVS rule of selection then acts upon the community, forcing a selection on those which are allowed to directly contribute to the GRASS project. In fact a certain degree of experience and a series of "good patches" (a short dry round of CVS diffs) are needed before a person can be enrolled in the "CVS write access circle". And this is valid also for non-crucial contributions, such as the translation contribution. What sets the boundary between a "good and a bad contribution" is at a first level the knowledge of the basic rules and commands for using the CVS.

Artifact rules for computer programming

According to the main GRASS CVS web page: «*Write access to the server is granted to everyone actively coding or helping with documentation in the project*» (http://freegis.org//grass/howto_grass-sshcvs.en.html). While it seems that write access may be easily obtained, there are constrictive rules that set up a strong boundary between good and bad contributions. Until here we do not have touched much the problem of what makes a contribution a good contribution, easily integrated into the system. We spoke more about the technical expertise of the contributor for which the knowledge at working level of the basic command of the CVS is fundamental. According to the development leader of GRASS:

If I realize that they are delivering quality: that's what they develop as usually working and also submitted in a reasonable way, so following our rules to code development, then we are granting write access to this person.

One of the things that defines a "good contribution" allowing power users to obtain write access to CVS is the application of the GRASS "rules to code development". GRASS programming practice are in other words, inscribed in textual documents such as for example the "GRASS Submitting File" or the "HOWTO translate GRASS messages"

[5] Passages in italic face are excerpts from the GRASS Mailing Lists.
[6] GRASS Users Mailing List

- hereafter HOWTO.

Textual artifacts are powerful inscriptions of every translation process (Latour, 1987) and in that respect the GRASS "programming guidelines" are very important in order to have a "good contribution". Textual guidelines for programming act as devices of interessement, allowing the developers to discern between good and bad contributions. Documents such as the HOWTO act upon the power users forcing them to make their i18n contributions suitable for the project according the rules contained therein. The main GRASS web page dedicated to development clearly address the need for programmers to follow the rules of these documents because only «This ensures a smooth integration into the standard GRASS code base» http://grass.itc.it/devel/index.php#submission. Here we propose an example extracted from the GRASS Users Mailing lists and also related to i18n GRASS project. But first of all we need to look at the main rule that work upon the i18n programming practices:

REQUIRED SOURCE CODE CHANGES (programming required)
Generally, to support i18n multiple languages, message strings in GRASS must be modified from

```
 fprintf ( ..., "...\n", ...);
to either
 fprintf ( ..., _("...\n"), ...);
or (omit \n)
 G_message ( _("..."), ...);
```

Careful: G_message should be used for messages - informations about the process for user while fprintf(stdout...) for data output.
(HOWTO translate GRASS messages, http://freegis.org/cgi-bin/viewcvs.cgi/grass6/locale/README?rev=1.31)

The HOWTO guideline asks people to program according to this rule. The document asks two tasks: developers are asked to add the gettext macro, but they are also asked to use the C programming language functions in a certain way. In particular the HOWTO guidelines clearly ask to use a function called G_message() instead of the fprintf() in the cases where the function's result, the message, gives information to the users about system processes. G_message() is a portable GRASS function to communicate user messages[7].

There is one important thing to notice regarding the role of the programming guidelines: it is misleading to think that every contribution will be incorporated or integrated into the system. There is a rule that is inscribed into the textual guidelines that organizes the way contributions must be prepared, received and integrated into the GRASS source code. Only the contributions that are compliant with the programming guidelines will be finally incorporated into GRASS. In that respect the programming guidelines act upon the GRASS community forcing people to provide contributions prepared in accordance with the rules contained therein. In order to better understand this point we can observe again a real example taken form the GRASS User Mailing List.

One of the issues of the i18n GRASS project may be that the source code written before the launch of the i18n, needs to be updated according to the HOWTO rules. In other words, the introduction of the programming guidelines forces the developers to update the source code according to the artifact rules. In order to better explain this point we can say that it may be that the need to remove somewhere in the code the C function fprintf() and substitute it with G_message(). What follows is a small part of a patch sent by a power user and related to this issue. In this case the user already know how to use the basic CVS rules and seems to be in "a dry round of CVS diffs". The message

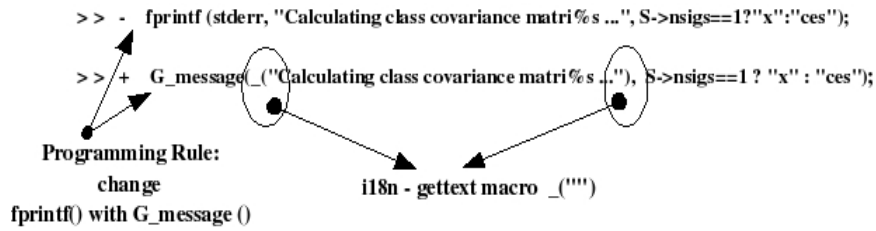
[7] A portable function is a function that can be used in several operating systems without the need to change and adapt.

incorporates the following statement :

Any objections to the attached patch? Just some benign updates to i.gensig to do with message output and localization.
[GUML, BD discussion dated July 21, 2005]

It is clear from the text that the patch aims to update the source code for the purposes of internationalization (message output). The question "Any Objections?" is instead about the process of source code review by the expert developers. It is now expected that the developers and also other users review the patch and state their objections. The message then continued with the updates to the source code, generated with a CVS diff. Figure 2 shows a small piece of the patch where it is applied the rule inscribed into the "HOWTO".

Figure 2 - A small part of the patch



The first line of the source code (marked by the symbol minus -) contained the function fprintf() and the updates implied the substitution of it with the function G_message(), according to the requirement of the "HOWTO". It is clear from this example how the guidelines' rules force the power users to modify the GRASS source code. Only the application of the guidelines' rules ensures that the contribution will be prepared as a good contribution. In fact only the knowledge and the application of such rules ensure that the contribution can be easily integrable into the system. The guidelines' rules act then upon the community, forcing people to prepare and write contributions which are totally complaint with the rules themselves.

We can notice that in the above patch it is also added the gettext macro. This is also a rule of the HOWTO guideline that allows to mark the string/message "Calculating class covariance matri%s ...", ready for translation. First reaction to this patch was from a developer and it was a positive one. No objections about any rule application.

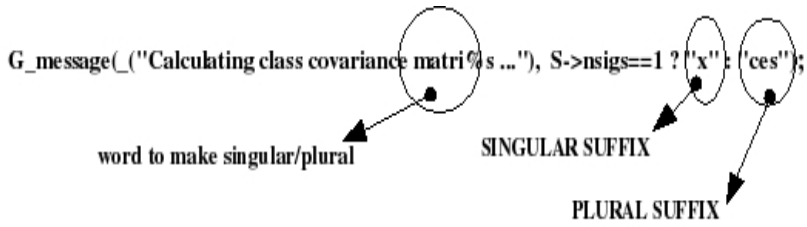
Source code rules

Programming guidelines have an important role in the realization of what the source code must do, because they create a standardised way of inscribing rules into the code. Nevertheless, the source code may already embody rules of programming that may become sources of small-scale controversies in community development practices. These rules, as we will see, are contestable and flexible and make it difficult for a good contribution to be realized. In that respect negotiations are required before that a contribution may be considered good.

In order to look at this problem we need to understand how work the line of code we see above, and repeated here emphasizing other aspects: see Figure 3 and 4.

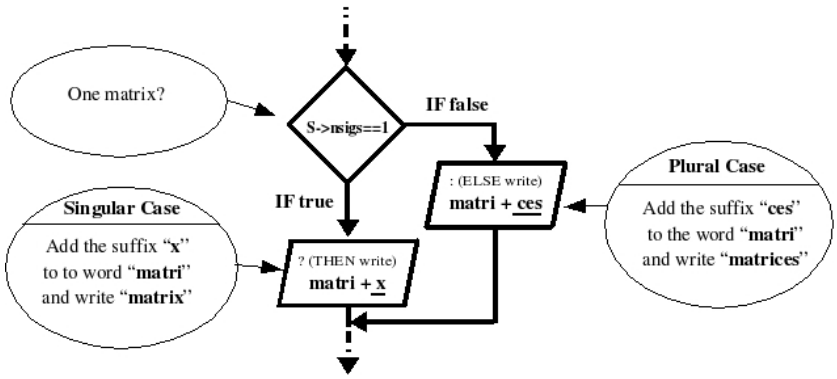
Figure 3 - Generating singular/plural sentences

Author



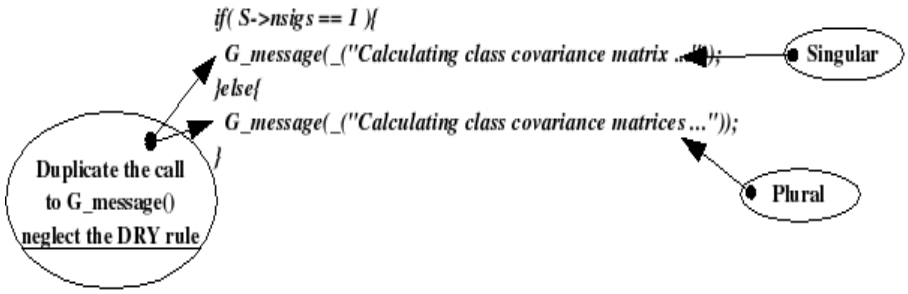
Within the line of source code in Figure 3 and 4, there is inscribed a rule of code programming that aims to not duplicate information. A common process, particularly in computing, called Don't Repeat Yourself (Wikipedia, article DRY), emphasizes that information should not be duplicated, because duplication increases the difficulty of change, may decrease clarity and so on. Thus, instead of duplicating information (matrix/matrices) in order to construct singular/plural sentences, this line of code add to the word "matri" a suffix: it adds the suffix "x" if the variable "S->nsigs" is equal to one, then generating the singular word "matrix"; it adds the suffix "ces" if "S->nsigs" is different from one, thus generating the word "matrices". In other words within one single call to the function G_message() it is possible to generate two distinct sentences (plural/singular). This trick works especially well for English, where the singular/plural handling may be (usually) easily obtained adding an "s" to the word (e.g. cat/cats). This way to handle singular/plural messages was a common practices within the C programming practices, at least before the introduction of i18n process (Free Software Foundation, 2002).

Figure 4 - Block diagram of the line of source code in Figure 3



Peer-review process revealed a problem in this rule. The modification in Figure 5 was suggested by a translator.

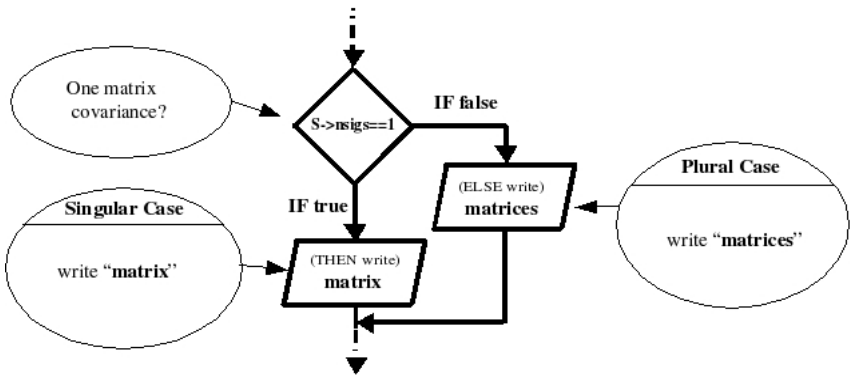
Figure 5- Suggested modification to the line of code



Within this modification, singular/plural cases are no longer constructed adding a suffix ("x" or "ces") to the word ("matri"). Instead, singular/plural messages are split in two separate messages, one for "matrix" and one for "matrices", hence neglecting the DRY rule. In fact, there are differences between languages that justify these modifications: the way in which plural forms are built differs and especially if there are languages with irregularities (e.g. German); the number of plural forms differ: some language families have only one form (e.g. Japanese) while others have many forms (e.g. Polish). Thus the DRY rule applied to singular/plural sentences in English cannot easily work for other languages. Figure 6 better explains the above piece of source code.

Figure 6- Block diagram of the suggested modification

Author



The suggested modification thus lead to duplicate the messages (singular/plural) and consequently the call to the function G_message(). The reasons for this review are indicated in the message:

1) Not every language has an easy suffix replacement plural/singular handling.
E.g. in spanish you also need to change an article that doesn't exist in english.

2) "x" and "ces" are not being translated.

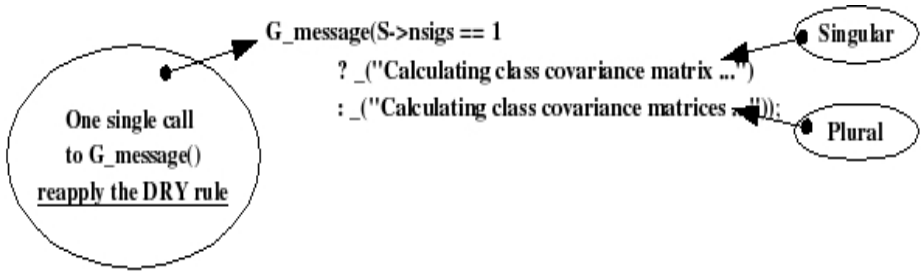
3) It's not at all obvious in the .po file what "... matri%s" corresponds to.
[GUML, DCA, discussion dated 21 July, 2005]

All these objections clarify why there is the need to modify the line of code in order to have two separated singular/plural messages. In that respect we can consider that the programming rules contained in the source code also act upon developers. In this example they are forced to modify the DRY rule in order to prepare a good i18n contribution. As it is shown, the DRY rule is in contradiction with the ways singular and plural cases are constructed in languages different from English. This is the reason why the author of the patch is asked to split the singular/plural construction in two separate sentences, hence duplicating also the calls to the G_message() function.

Nevertheless, the peer-review process identified the possibility of taking into account both the DRY rule and the reasons for splitting singular/plural cases into two separated messages. We can observe the following patch, coming from the GRASS Mailing Lists,

where an important developer suggested the line of code shown in Figure 7 in order to solve this issue [GUML, GC, discussion dated 22 July, 2005].

Figure 7- Final suggestion incorporating both the singular/plural handling and the DRY programming rule



From this last modification it is clear how the DRY rule acts upon the development practices, forcing the developers to not duplicate information. Also the source code and programming rules then contribute to the ways a good contribution to the i18n project can emerge.

We can now observe that in all the modifications of the initial line of source code, which can be better seen as inscriptions, the results are almost the same: in all cases the same plural singular sentences are written. But what the source code must do has to do with rules that shape and reshape the initial lines of code. There are rules, such as the DRY, already contained in the source code, that affects what the source code must do. Only after a series of negotiations can the final contribution emerge: this last inscription (Figure 7) contains thus both the translation needs and the DRY programming rule.

In the peer review process of this patch we have a first reaction that identified the patch as a "good patch", with no violation of the GRASS programming rules. Nevertheless, a patch, in order to be considered a good contribution, needs a major review from many points of view. Only after a series of negotiations, where the source code is shaped and reshaped, can a good contribution emerge from practices and can be written into the CVS, hence becoming integrated into the system.

6 Mobilization and artifact rules: conclusion and future work

This paper started by asking how complex tasks are accomplished by large web based communities. In order to better understand this point we can recall the example of the hotel manager provided by Latour, in which a translation process is seen as a confrontation between the programs: leaving versus not leaving the keys. In that example the hotel manager accomplishes his task using an artifact, the metal block, that embodies a rule that acts upon the hotel guests, forcing them to leave the key at the reception desk.

We have addressed the tasks accomplishment in the GRASS community using the point of view provided in the hotel manager example. In particular we have observed the translation process that has to do with the programs of producing "good contribution versus bad contribution" in the case of the internationalization process of the system. We have argued that it is thanks to the rules embodied in artifacts such as the CVS, the programming guidelines and the source code that complex tasks such as software development and the internationalization of messages are accomplished. We have argued that the artifacts' rules act upon the GRASS community participants, organizing their work and forcing them to provide good contributions.

One further point to understand in our argument is that we did not discuss in detail the

mobilization of the whole actor-network and the success of the translation process. The conclusion of the paper is the right place to address this issue. Our empirical examples represent small events in the whole complex world of GRASS development. Nevertheless it is not secondary to remember that it is thanks to the collection of many small good contributions that FLOSS development goes on. New and frequent release cycles of the software are possible because of the integration of many small contributions. Moreover, the enrolment of new programmers within the "CVS write access circle" is also based on the peer-review of small contributions.

In that sense the intuition that artifacts may embody rules for coordination of FLOSS communities (Lanzara and Morner, 2005) is a fundamental one. Focusing on artifacts we can grasp the rules that lead to organizational processes and to the coordination of many dispersed resources. We have clearly observed using an ethnographic approach how the CVS, the programming guidelines and the source code have a powerful role in the coordination processes of the many, small contributions. Given the rules of the CVS that force the developers to select who can participate in the "write access circle", the knowledge of the basic rules of "CVS diffs" is a necessary precondition for a good contribution to emerge. Programming guidelines are also powerful in order to make patches written in a standardised way and easily integrated into the system. Nonetheless, a contribution must be judged from many points of view: source code has also is own rules that emerge from the technical knowledge inscribed into it.

Mobilization and success of the translation process in FLOSS projects thus may happen when many who contribute will act according to the basic artifacts rules. Only in that case can the developers really claim to be able to speak in the name of their community: the actor-network of a FLOSS project can then act as single whole, really delivering high quality software from a Bazaar of ideas and approaches. Again the mobilization process in the case of GRASS can be compared to the hotel manager example. It is only when all the hotel guests act according to the rule embodied into the metal block, that the hotel manager succeeds in his project. The same dynamic emerges from our analysis: it is only when all the participants to the GRASS community act according to the artifacts rules, that a complex task such as the internationalization can be accomplished.

A final consideration: our enquiry into the role of source code in FLOSS communities has been limited to the i18n GRASS project and to translation issue. In retrospect, this has been a good choice in order to explore the coordination efforts in FLOSS community, mainly because i18n is simple enough to be understood by those who are not developers. We are anyway aware that there are situations that are far more complex and where the realisation of a good contribution implies many more negotiations than the ones we saw here. Future studies may address these problems.

Aknowledgement

This research has been funded by the Italian research project PRIN 2004 "Apprendimento tecnologico e tecnologie dell'apprendimento". We thanks also all the participants to the workshop on "Online learning communities" for the interesting feedbacks and discussions, especially John Cuthell for his careful review of the paper. Finally, we are grateful to the whole GRASS project and in particular to Markus Neteler. Without his support our research would have not been possible.

References

Akrich, M. (1992) 'The de-scription of technical objects', in Bijker, W. and Law, J. (Eds.): Shaping technology building society, MIT Press, Cambridge, MA, pp. 206-224.

Bar, M. and Fogel, K. (2003) Open Source Development with CVS, 3rd Edition, Paraglyph Press.

Brooks, F.P. (1975) The Mythical Man Month, Addison-Wesley, Reading, MA.

Bruni, A. and Gherardi, S. (2007) Studiare le pratiche lavorative, Il Mulino, Bologna.

Callon, M. (1986) 'Some elements of a sociology of translation : domestication of the scallops and the fishermen of St. Brieuc Bay', in Law, J. (Ed.): Power, Action and Belief : a New Sociology of Knowledge?, Boston and Henley, Routledge and Kegan Paul, London, pp. 196-233.

Callon, M. (1987) 'Society in the making: the study of technology as a tool for sociological analysis', in Bijker, W. Pinch, T. and Huges, T.P. (Eds.): The Social Construction of Technological Systems : New directions in the Social Study of Technology, MIT Press, Cambridge, MA, pp. 83-103.

Callon, M. and Latour, B. (1981) 'Unscrewing the big Leviathan: how actors macro-structure reality and how sociologists help them to do so', in Knorr Cetina, K. e Cicourel, A. (Eds.): Advances in Social Theory and Methodology, Toward an Integration of micro and Macro Sociologies, Routledge, London, pp. 277-303.

Fogel, K. (2005) Producing Open Source Software: How to Run a Successful Free Software Project, O'Reilly Media Inc. http://www.producingoss.com/

Free Software Foundation (2002) 'GNU gettext Utilities', Retrieved January 10, 2007, from http://www.gnu.org/software/gettext/manual/

GRASS Development Team (2006) 'Geographic Resources Analysis Support System (GRASS)', ITC-irst, Trento, Italy. http://grass.itc.it

GRASS Development Team (2006) 'SUBMITTING FILE v.1.22', Retrieved April 23, 2007, from http://grass.itc.it/grass63/source/SUBMITTING

GRASS Development Team (2007) 'HOWTO translate GRASS messages v.1.31', Retrieved April 23, 2007, from http://freegis.org/cgi-bin/viewcvs.cgi/grass6/locale/README?rev=1.31

GRASS Users Mailing List Archive (1991-2007) available at http://grass.itc.it/pipermail/grassuser/

GRASS Translation Mailing List Archive (2004-2007) available at http://grass.itc.it/pipermail/translations/

Hertel, G. Niedner, S. and Herrman, S. (2003) 'Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel', Research Policy, Vol. 32(7) , pp. 1159-1177.

Himanen, P. (2001) The Hacker Ethic and the Spirit of the Information Age, Random House Inc., New York, Usa.

Hine, C. (2002) Virtual Ethnography, Sage Publications, London.

Kernighan, B.W. and Ritchie, D.M. (1988) The C Programming Language, 2nd Edition, Prentice Hall, Englewood Cliffs, NJ.

Lakhani, K. and Wolf, R. (2005) 'Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects', in Feller, J. Fitzgerald, B. Hissam, S. and Lakhani, K. (Eds.): Perspectives on Free and Open Source Software, MIT Press. Cambridge, MA.

Lanzara, G.F. and Morner, M (2005) 'Artifacts Rule! How Organizing Happens in Open Source Software Projects', in Czarniawska, B. and Hernes, T. (Eds.): Actor-Network Theory and Organizing, Liber & Copenhagen Business School Press, Malmo, Den., pp. 67-90.

Latour, B. (1987) Science in Action, How to Follow Scientists and Engineers through Society, Harvard University Press, Cambridge.

Latour, B. (1992) 'Technology is Society made durable', in Law, J. (Ed.): A Sociology of Monsters, Routldege & Kegan Paul, London, pp. 103-131.

Latour, B. (1999) Pandora's Hope, Harvard University Press, London.

Latour, B. (2004) 'On using ANT for studying Information Systems', in Avgerou, C., Ciborra, C. e Land, F. (Eds.): The Social Study of Information and Communication Technology,, Oxford University Press, Oxford, UK, pp. 62-76.

Lerner, J. and Tirole, J. (2002) 'Some simple economics of open source'. Journal of Industrial Economics, Vol. 50 (2), pp. 197–234.

Lin, Y. (2005) 'The Future of Sociology of Floss', First Monday, Special Issue #2: Open Source, October 2005, http://www.firstmonday.org/issues/special10_10/lin/index.html

Lindenberg, S. (2001) 'Intrinsic motivation in a new light', Kyklos, Vol. 54 (2/3), pp. 317-342.

Masumoto, S., Raghavan, V., Nonogaki, S., Neteler, M., Nemoto, T., Mori, T., Niwa, M., Hagiwara, A., and Hattori, N. (2005) 'Multi-Language Support and Localization of GRASS GIS', International Journal of Geoinformatics, Special Issue on FOSS/GRASS 2004 & GIS-IDEAS 2004, Vol. 1(1).

Neteler, M. (2001) 'Towards a Stable Open Source Gis', presented at the Second Italian GRASS Users Meeting, University of Trento, Feb. 1-2, 2001, http://montagna-europa.itc.it/markus/papers/geomaticsWB2_devel.pdf

Neteler, M. (2006) 'Community based software development: The GRASS GIS project', presentation at the University of Trento (Italy), 30 November 2006.

Open Source Initiative (1997) 'Open Source Definition', Retrieved January 10, 2007, from http://www.opensource.org/docs/definition.php

Raymond, E.S. (1998) 'The Cathedral and the Bazaar', First Monday, Vol. 3(3), http://www.firstmonday.org/issues/issue3_3/raymond/

Shaik, M. and Cornford, T. (2004) 'Version control tools: a collaborative vehicle for learning in F/OS', in Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. (Eds.): Collaboration, Conflict and Control: Proceedings of the The 4th Workshop on Open Source Software Engineering, May 25, 2004, Edinburgh, Scotland, pp. 87-91.

Stallman, R. (1985) 'The GNU Manifesto', Retrieved April 18, 2007, from http://www.gnu.org/gnu/manifesto.html

Stallman, R. (2002) Free Software, Free Society: Selected Essays of Richard M. Stallman, GNU Press, Boston.

Tuomi, I. (2001) 'Internet, Innovation, and Open Source:Actors in the Network', First Monday, Vol. 6(1), http://firstmonday.org/issues6_1/tuomi/

Tuomi, I. (2004) 'Evolution of the Linux Credits file: methodological challenges and reference data for open source research', First Monday, Vol. 9(6), http://www.firstmonday.org/issues/issue9_6/tuomi/index.html

United States Army CERL (1996) 'Announcements', Retrieved April 18, 2007, from http://web.archive.org/web/19970619195255/www.cecer.army.mil/announcements/grass.html also available at http://grass.itc.it/announces/cerl1996/

von Krogh, G., Spaeth, S. and Lakhani, K. (2003), 'Community, Joining, and Specialization in Open Source Software Innovation: A Case Study', Research Policy, Vol. 32, pp.1217-1241.

Wittel, A. (2002) 'Ethnography on the Move: From Field to Net to Internet', Forum Qualitative Sozialforschung, Vol. 1(1), pp. 1-9.

Wikipedia articles:

- Concurrent Versioning System, Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/Concurrent_Versions_System
- Don't repeat yourself, Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- gettext, Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/Gettext
- Geographic Information System, Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/Gis
- Internationalization (i18n), Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/I18N
- Patch, Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/Patch_%28computing%29

- Power User, Retrieved December 13, 2006, from http://en.wikipedia.org/wiki/Power_user