The construction of peers and artifacts:
the organizing role of "Programming Guidelines"

Vincenzo D'Andrea
Dipartimento di Ingegneria e
Scienze dell'Informazione –
University of Trento
vincenzo.dandrea@unitn.it

Stefano De Paoli
Department of Sociology –
National University of Ireland
Maynooth - Ireland
Stefano.DePaoli@nuim.ie

Maurizio Teli
Faculty of Sociology –
University of Trento
maurizio@maurizioteli.eu

Abstract

In this paper, we[1] study new organizational forms for production processes that emerge in large scale Free/Libre Open Source Software (FLOSS) projects. We will focus on the textual artifacts known as "Programming Guidelines" and on the rules and practices they contain. Our reflection is grounded in the practical activities of the people involved in the innovation processes. In particular, we take into account how in FLOSS to become a peer ("Peering") is an hybrid result of social dynamics and artifacts actions. We will show how Programming Guidelines participate in building boundaries around the potential participants and in defining the legitimate form of participation in terms of coding practices. Our conclusions emerge from the analysis of two empirical cases: the operating systems OpenSolaris and the geographical information system GRASS.

1. Introduction: "New" production and innovation models

The literature about FLOSS and Internet related phenomena stresses how the emergence of new forms of organizational structure can be dependent on new technologies, such as, for example, in the argument by Weber [21], who underlined how:

"Standard arguments in organizational theory predict that increasing complexity in a division of labor leads to formal organizational structures….In contrast,… much recent literature on the Internet and the 'new economy' argues that Internet technologies radically undermine organizational structures because they reduce the costs of communication and transaction toward an asymptote of zero. This is supposed to enable the formation of 'episodic communities on demand,' so-called virtual

organizations that come together frictionlessly for a particular task and then redistribute to the next task just as smoothly" [18, page 171]

This argument is common among scholars from different background and it can be described referring to the work by Tapscott and Williams in their recent book Wikinomics [20]. According to them, the mass collaboration (the episodic communities on demand, in Weber terms) that characterizes the 21th century economy (the wikinomics) is based on four structural factors, namely: the Openness, the Sharing, the Peering and the Acting Globally. Their view is that users innovation and the mass collaboration – widely used today by many enterprises, operating in the global market - is strictly a results of the very existence of these four structural factors. Tapscott and Williams call the user innovation process as "prosumption", whereas the prosumer is a "professional consumers", namely a person who is at the same time the innovator and the consumer of innovation [20].

While we share with Tapscott and Williams a genuine interests for the mass collaboration issue, that is a fundamental characteristic of FLOSS, we disagree with their general perspective. Adopting a constructivist approach [15] we claim that the Openness, the Sharing, the Peering and the Acting Globally are not what explain the user innovation and the mass collaboration, but rather what has to be explained, because they don't show up as exogenous elements in the experience of groups construction and evolution, but they are part of the different processes bringing to mass collaboration as a result.

In a prior work [5] we have discussed how the "Openness" of a FLOSS project is more a matter of negotiations and instability, rather than a stable structural factor. Our observations tell us that many different degrees of Openness do exists in a software project and in particular the Openness seems to be related to the choice of licenses, which again is a negotiated process among developers and users. The Openness therefore is more the results of innovation processes than what explains innovation in itself. This idea is very close to the argument put forward

for example by Raymond in his famous essay The Cathedral and the Bazaar [16], in which he explains how the real innovation of Linux was not the software per sé, but instead the organizational process behind the development, namely the Openness, Sharing and Peering rules of the Linux development model (the Bazaar).

In contrast with Tapscott and Williams[20] arguments, in this paper we aim at discussing how the "Peering" is a negotiated process of innovation, based on practices of submission of pieces of code and practices of judgement on the same pieces. Peering is an important element of FLOSS projects, sometime taken as the key of FLOSS technical success and ability of delivering high quality software, such as in Raymond's expression "Given enough eyeballs, all bugs are shallow" [16] shows, underlining the fact that the existence of a wide base of peer-reviewers lead to a fast problem's solution. So, source code review aims at fixing bugs and known problems and at improving the quality of software. In this pattern of practices of code submission and judgement, an important activity is carried on by "Programming Guidelines", casting into quite stabilized artifacts the proper use of code conventions, discriminating between good and bad coding practices. In summary, we can state that: "Programming Guidelines" participate in the definition of which pieces of code are legitimately part of the FLOSS project collective and which are not.

In studying the "Programming Guidelines", we attempt to describe the ways these textual artefacts participate to the organizing process in large Open Source Software projects, that we see as a case of new organizational forms for production processes [21]. We will show in this paper how "Programming Guidelines" inscribe rules and practices organizing the software development project, thus becoming a critical pathway for our understanding of the collective actions and task accomplishment in these human groups [14]. In the rest of this paper, we will start by grounding our contribution in the study of the contemporary processes of production and innovation, showing how some of the key elements of innovation are the results of the practical activities of the people involved. Focusing on one of these social dynamics, Peering, we will show how in FLOSS projects to become a peer is an hybrid result of processes involving artefacts actions. More precisely, we will show how Programming Guidelines participate to the construction of boundaries around the potential participants and defining the legitimate form of participation in terms of coding practices. In other words, we will show how Peering is the result of the processes taking place whithin and around the Programming Guidelines.

In the last part of the paper, we will provide an account of the empirical foundation of our reflections, describing how the complex role of Programming Guidelines as organizing artefacts emerges through the analysis of two empirical cases, the Operating System OpenSolaris, and the Geographical Information System GRASS.

2. Coding conventions and Programming Guidelines

The emergence of coding conventions (both formally written and informally transmitted) can be considered as the need of rules that computer programmers follow in order to ensure that their source code will be easy to read and maintain, in one phrase to allow the sustainability in time of the code base [18]. As an example, the Sun Microsystems introduction to Java[2] coding states that code conventions are fundamental because:
- 80% of the lifetime cost of a piece of software goes to maintenance;
- hardly any software is maintained for its whole life by the original author;
- code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly;
- if you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

When coding conventions are formalized in written documents, those documents are usually known as "Programming Guidelines". These Guidelines are central in Free/Libre Open Source Software (FLOSS) development, especially in the case of large projects that could not take place without them. In fact FLOSS development teams are usually globally dispersed all over the globe. Given this situation, developers must find solutions for ensuring what they socially recognize as quality and easy integration of their work in the main project and several artifacts are used for this reasons [12, 3]. Programming Guidelines are among these artifacts and in particular they act in a double direction: first, they allow the standardization of the result of the programming activity, in order to let the development teams to receive "standardized innovations" by the users; second, they build a boundary around which kinds of user are allowed to participate to the innovating process. As any artifacts, "Programming Guidelines" are not without history or social assembling processes; instead, they

---

make possible to open up a path of comprehension of the assembling power of production efforts, letting us understand more which kind of criteria helps defining the legitimate peer, which kind of previous existing relationships are translated into them and how they are part of sets of instructions that cooperate to the definition of legitimate claims of participation.

3. Theory and relevance of Programming Guidelines

We study "Programming Guidelines" in two ways. First, we observe how programming practices are literally inscribed into them as textual artifacts, separating one specific open source community from others. As a first theoretical framework for our exploratory analysis we assume the contribution by Lanzara and Morner [14]. For the authors, if we want to observe the coordination efforts and the complex task accomplishment in FLOSS development, we need to move the focus of our attention from the organizational aspects to the technological artifacts. We agree with Lanzara and Morner that «technology can replace formal organizational rules and structures in the coordination and governance of complex activity systems» (p. 67). In this light, in order to enquiring the spatially distributed software development, the analysis of the role of technologies and artifacts in FLOSS projects become fundamental.

The notion of inscription taken from the Actor-Network Theory 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» [1, p. 208]. This notion for Lanzara and Morner [14] help to grasp the organizing role of the technology in FLOSS project because it helps to see how human agents delegate to artifacts the ability to do things and perform functions.

Using a semiotic approach then, we seek to observe the stabilization of certain programming practices (coding conventions) or, in other words, how and why programming practices continue to be practicized within an FLOSS project. In fact, it is important to remark that "Programming Guidelines" constitute a stabilization, over time, of how the source code of a computer program is shaped. In addition, the "Guidelines" may show why a certain practice has been selected to become a standard programming practice within a specific open source community.

Second, we observe how these practices are socially sustained and negotiated by FLOSS developers. According to Akrich [1], in order to maintain a complete methodological perspective it is important to observe not only the construction of technologies and artifacts (i.e. the inscription process), but also how artifacts are used by users: "we have to go back and forth continually between the designer and the user, between the designer's projected user and the real user, between the world inscribed in the object and the world described by its displacement." (pp. 208-209). Following this intuition we remark that the legitimization of the "Programming Guidelines" is in fact a process achieved through different paths of negotiation among users and developers. Indeed, a contribution to a FLOSS project might only be included in the main code base if this is compliant with the "Programming Guidelines", thus creating a situation for inclusion or exclusion of those who are not programming according to them [4]. In this light we focus also on the description of how users and programmers do concretely use Programming Guidelines.

4. Methodology

Our research is empirically based on two large FLOSS projects: the Geographical Information System known as Geographic Resources Analysis Support System (GRASS), developed by a world-wide team on a voluntary basis, and the Operating System known as OpenSolaris, backed by Sun Microsystems, Inc. The data collection has been based on on-line observation, informed by a cyber-ethnographic approach [11, 19], while the data analysis has been carried on relying on a Grounded Theory approach [10].

In the OpenSolaris case, we referred to documents downloaded from the project website[3] and on mailing lists, wikis, and blogs observation, observation that took place since June 2005 to March 2007. In particular, in relation to this paper, the information we focused on are the ones related to the path an external developer is supposed to follow in order to obtain the integration of source code in the OpenSolaris code base, drawing upon the mailing list "code"and on the about 500 threads on it in the considered period.

Relevant documents of the GRASS projects have been taken from the main web site of the project[4]. We have also followed several discussion about the application of programming practices and software development of our case study during the period

---

January 2005 – December 2007. These discussion have been extracted from the projects mailing lists, in particular from both the GRASS Users and Developers Mailing lists, using the internal search engine. We have analyzed 55 emails organized in 20 threads. Our focus has been directed most on messages sent by programmers who were not member of the development core team. In order to know who these programmers were we used the GRASS Changelog files[5]. We have also extracted data form the GRASS source code using the GRASS Source code browser[6].

5. Guidelines as a re-production of corporate practices: OpenSolaris

The OpenSolaris project was opened to the public on June 2005, when a corporation, Sun Microsystems, Inc., released part of of the code of its operating system, Solaris, under the terms of a FLOSS license, the Common Development and Distribution License (CDDL). As the license name let us understand, OpenSolaris was released according to a strategy of increasing the number of developers and distributors of the operating system (based on the code of ATT Unix System V mixed with Berkeley Software Distribution code in the '80s). According to this plan, a set of technological and textual artifacts has been created, both by the corporation itself and by the group of people participating to what is known as "The OpenSolaris Community", that is the group of people practising software development and use in connection with the project website. During our research on this project we focused our attention on how the practices of participation to the community activities are strongly mediated by the artifacts at play, artifacts which defines the boundaries of participation relying on the practices coming from different arenas of activities, like the business oriented structure of the corporation, the history of Solaris programming practices, and the wider FLOSS panorama. The intermingling of these different practices backs the emerging of groups, discourses, and strategies in "The OpenSolaris Community". Programming Guidelines, in the name of "Style Guidelines", which should be correctly considered as part of "The Developers Reference", are one of this artifact, mainly acting as a separator between who has the knowledge to contribute source code to Solaris and who has not.

In particular, we consider two different aspects of "Style Guidelines", that are: their origin, connected with the proprietary stage of Solaris development, and their ability to participate to two different paths in the OpenSolaris life. First, "Guidelines" align with the marketing-oriented need for stability between the different versions of the operating system, shaping the network of entities potentially participating the project; second, they strengthen the role of Sun history and practices in carrying on the innovation process, due to the stability of software production inscribed in them.

5.1. The Developers Reference brings to the history of Solaris

> "The Developers Reference is big. Really, really big. I mean, you just won't believe how vastly, hugely, mind-bloggingly big it is" Mike Kupfer[7]

The Developers Reference[8] is a comprehensive ensemble of documentation that helps defining a path of use and development of OpenSolaris that goes from the installation to the code submission, passing through the description of the system and the processes of test and review of the contributed code. In this path we find the "Style Guide"[9], as a sub-section of a piece of documentation titled "Best Practices and Requirements"[10]. As this title show, contributing code to the project involves a double path of adherence to constraints (requirements) and of enactment of good programming practices. All of them entered the OpenSolaris community through the mediation of Sun stabilized practices and Sun's employees habits, as the introduction to this section reminds:

"These rules must be followed by all developers. These rules are the same ones which Sun engineers have been required to follow when developing Solaris, and are the primary reason Solaris, and now OpenSolaris, is among the world's best available software"

This piece introduces clearly the relationship of translation of Solaris into OpenSolaris, stressing the continuity between them. In particular, talking about "Guidelines", the fact that "these rules are the same one" construct newcomers in OpenSolaris development as followers of the practices and rules of the original developers, which are the Sun engineers involved in the system development previously than it was released under the CDDL

---

[5]   Available at http://grass.itc.it/devel/grassreleases.html
[6]   http://web.soccerlab.polymtl.ca/grass-evolution/grass-browsers/code-browser.html

[7]   http://opensolaris.org/os/community/on/devref_toc/
[8]   http://opensolaris.org/os/community/on/devref_toc/
[9]   http://opensolaris.org/os/community/on/devref_toc/devref_7/#7_2_style_guide
[10] http://opensolaris.org/os/community/on/devref_toc/devref_7/#7_best_practices_and_requirements

license. In that way, not only artifacts participate to the definition of the knowledge prospective participants have to gain, but also make it in connection with the stabilized company habits and needs, enrolling new participant to the Sun views of software development. As the next sub-section will show, these views are equally important when we look at the requirements phase, into which a defined commercial strategy has been inscribed.

5.2. The "Bill Joy Normal Form"

As already mentioned, the history of Solaris has been translated into instructions for OpenSolaris developers, also in the form of "The Developers Reference", and it participates in an equally important way to the shape the final contributed code needs to have, that is the style Solaris code has since its first appearance as SunOS (the previous name of Solaris). Small changes have taken place in the style, mainly adhering to the ANSI C style, dismissing the K&R style (from the initial letters of Brian Kernighan and Dennis Ritchie, the writers of one of the most famous handbook on the C programming language, [13]), but the biggest contribution to the style was the one by one of the founders of Sun Microsystems, Bill Joy. As a developer wrote in the code-discuss mailing list:

"I have read the style guidelines. Given that my driver was not written to conform to "Bill Joy Normal Form", hand conversion would be painful.
Does anybody have an indent recipe, or sed/perl/etc script to re-format C code into something which can pass cstyle.pl? "
[code-discuss, Jul 12, 2006]

This piece of email moves our attention to two other elements: the first one, the need to change the style can be framed in negative terms (painful); the second one, this change is needed in order to pass cstyle.pl, that is an automated script able to check the code style before the code enters a review from peers. OpenSolaris (and Solaris) have two of those scripts, the cited cstyle.pl and hdrchk, which construct a first technological boundary to the participation of code to the project and to the standardization of results before entering the review process carried on by peers. So, coding conventions, at this time, are the results of history, different practices coming from different panoramas, and automated tools to check the adherence to them. How the guidelines and the review process affect the technological result is what we are going to learn from the case of GRASS.

6. The construction of good functions and good developers: GRASS

In the GRASS case we use the "Programming Guidelines" as the key for understanding the source code contributions done by programmers that were not member of the GRASS Development Team. In fact, the social dynamics related to the software source code development still remain largely unstudied, due to the difficulties for social researchers to access the source code as empirical data. Programming Guidelines constitute an interesting entry point in this problem.

6.1. GRASS Guidelines

As previously stated, we base our observation on the case of a FLOSS GeGIS known as GRASS [7]. GRASS started at the beginning of the '80 as a small project of the United States Army. In 1996, US Army decided to stop the development of the system and invited the users to migrate toward proprietary GIS a. At the end of 1998 a new GRASS Development Team (GDT) came up with the aim to re-launch the GRASS development and community. The GDT is today composed by an international group of voluntary researchers affiliated to different institutions; the GDT has also a structure very close to the "town council" model [2].

In large FLOSS projects, software development is usually managed using internet based technologies known as Versioning Control Systems (VCSs) [17]. The VCSs keep track of all work and all changes in a set of files, and allows several developers to collaborate on-line in the software development processes. The VCSs are based on a client/server architecture: a server stores the current version(s) of the project and its history, and clients connect to the server in order to check-out a complete copy of the project, work on this copy and then later check-in their changes.

The knowledge and application of the GRASS programming guidelines and of the ANSI C standard are fundamental for getting a write access for source code in GRASS Version Control System[11]. GRASS programming practice are inscribed in two file respectively called "SUBMITTING FILE" and "SUBMITTING SCRIPT". The main GRASS web page dedicated to development clearly address the need for programmers to follow the rules of these files:
"C language coding standards: Check your code against the rules defined in the 'SUBMITTING' file .

---

[11] During the period we consider here GRASS VCS was the Concurrent Version System. However recently the project has been migrated in SVN.

This ensures a smooth integration into the standard GRASS code base:"[12]

The rules follows this invitation and as we said their application is fundamental for becoming an active member of the development team. In fact the expert developers judge those who aspire to become part of the development team in the light of their knowledge of the guidelines, as the following excerpt from a talk identifies :"*If I realise that they are delivering quality, that is, what they develop is usually*

working and submitted in a reasonable way, then, following our rules to code development, we grant write access to this person.".

Here we would like to address just one of these rules, even if it may seem very basic. One of the more interesting characteristic of the GRASS programming practices is the existence of C Language dedicated library of functions:

"Use the following GRASS library functions instead of the standard C functions. The reason for this is that the following functions ensure good programming practice (eg always checking if memory was allocated) and/or improves portability.[..] They may perform a task slightly different from their corresponding C library function, and thus, their use may not be the same." [8]

As an example of the difference between a standard C function and the GRASS dedicate function we propose here a the difference between the C standard function *malloc()* and dedicated GRASS function *G_malloc()*. The *malloc()* function is the base function used to dynamically allocate a portion of memory in C. Its prototype is: *void *malloc(size_t size)* which allocates "size" bytes of memory. With a successful completion of the function, the *malloc()* functions return a pointer to the allocated space [9]. Below we have the GRASS *G_malloc()* function.:

```
void *G_malloc (size_t n)
{
  void *buf;
  if (n <= 0) n = 1;
    /* make sure we get a valid request */
  buf = malloc(n);
  if(buf) return buf;
  G_fatal_error("G_malloc: out of memory");
  Return NULL;
}
```

GRASS Dev. Team (2006).[13]

The dedicated GRASS function *G_malloc()* performs the same basic operation as the correspondent C function, allocating a memory portion of "n" bytes and returning a pointer to the allocated space. In addition , the *G_malloc()* : it

manages the error message in a standardized way for all the GRASS programming framework, using another dedicated function called *G_fatal_error()*. Thus it is evident that for a programmer the knowledge and application of the basic standard C programming practices is not enough to gain a write access to GRASS VCS . Programmers need also to know how the dedicated GRASS C function operates and how the GRASS programming practices are implemented in the software system.

We will now provide an example of application of the GRASS Programming Guidelines as related to the *G_malloc()* function, taken form the GRASS Developers mailing list. The example we present is about source code "cleaning" with the elimination of the standard C function from the GRASS code and the substitution with the GRASS analogous function. This task of cleaning the software – in the example we propose – is done by a developers who aims at becoming part of the GRASS Development Team. Hence, his work has to be judged by GRASS expert developers. In the example a programmer send a patch opening his message as follows: "*Attached is a patch to lib/imagery to update memory allocation. Objections?*"

The question "Objections?" is about the process of source code review by the expert developers. As we can note now from this excerpt of the submitted patch, some of the cleaning work has been done on the substitution of the standard C function *malloc()* with the GRASS dedicated function *G_malloc()*. The symbol minus (-) represents the eliminated lines of code, while the symbol plus (+) represent the new lines of code:

```
+void *I_malloc(size_t n)
{
  void *b;
- b = malloc (n);
(* standard C malloc() removed *)
- if (b == NULL)
- printf (stderr, "Out of Memory\n");
(* elimination of the error message *)
+ b = G_malloc(n);
(* add GRASS memory allocation function
*)
+ return b;
}
```

GRASS Developers Mailing list, message sent Thu, 04 Aug 2005, (italic comments added)[14]

It is worth noticing that the GRASS code cleaning done by this developers follows the prescription inscribed in the SUBMITTING FILE and in the *G_malloc()* function:

1. the standard C function *malloc()* is replaced with the *G_malloc()*

2. the error message displayed by the C function *fprintf()* is eliminated, because it is already contained

[12]   http://grass.itc.it/devel/index.php#submission
[13]   GRASS Software: grass6/lib/gis/alloc.c
[14]   http://lists.osgeo.org/pipermail/grass-dev/2005-August/019294.html

and managed in a standardised way by the *G_malloc()* function using the *G_fatal_error()*.

If no objections came up, it is possible for the developers to commit the code of the patch to the GRASS CVS.

Even if the example we provided above seems to be simplistic, in fact it isn't. It is worth noticing that it is thanks to small contributions and their coordination that FLOSS development is possible [5, 14].

It is interesting to note that also the SUBMITTING file (i.e. the GRASS programming guidelines) are subject to his kinds of dynamics. In the following message a programmer (one of those aiming at becoming member of the GRASS Development Team) sent a patch to the SUBMITTING:"I rearranged the messaging section of the SUBMITTING file to make it a *little easier to read (IMHO). Objections to committing?"[15]*.

Again the question "Objections" is related to the peer review of his patches. In this case one of the member of the GRASS Development Team suggested a change to the patch, related to the order in which the programing rules were presented. The programmer accepted the modification suggested by the expert developer and the patch was finally committed to the GRASS VCS.

This last example tell us two interesting things. First of all programming guidelines are not static things. They change during the times, in the same ways the source code change. Second, it is interesting to remark that also programmers who aspire to become member of the GRASS Development Team may concur to modifying the documents on which they are judged (even if the modifications are very small).

7. Guidelines between inscriptions and de-scription

We started underlining how our concerns are related to the organization of mass production in the contemporary society, mainly in the Internet-mediated groups that characterize the recent development of organizations [21]. In particular, we have conceptualized Peering as a productive process of peers construction instead of conceptualizing it as a structural predetermined factor, as Tapscott and Williams do [20]. As the dynamic approach to FLOSS has shown [6] the practices of participation in FLOSS projects follow trajectories that needs to be scholarly understood as a socio-technical

processes. In this processes, the importance of artefacts could not be underestimated and need an empirically oriented research.

On our side, we have empirically shown how "Programming Guidelines" are part of these processes as translated both in formal procedures, the OpenSolaris Developers Reference, and in technological choices, like the *G_malloc* function. That translations participate in the life of the two projects standardizing the result of the programming activities and building a boundary around which kind of entities could participate in that.

In this sense, we have shown not only how software is the result of an organizational process, (a mirror of the same organizational structure producing it as the Conway's Law highlight, [12]), but also how the software is participating in defining the same organizational unfinished structure. Software not only tells something about organization, it organizes too [14].

In particular, the processes of inscription of actions in the "Programming Guidelines" have underlined how we can't understand FLOSS projects without relating to their history. In the OpenSolaris casehistory is able to explain the actual shape of the programming guidelines and the conventional way developers refer to the form of code. At the same time, looking at the "Programming Guidelines" like we did with GRASS, on one side let us understand the network of paper relationships that surrounds the contribution to code, on the other one showed how the same artefacts are potentially under discussion in a dynamic way.

8. Conclusions

In conclusion, in this paper we have tried to provide an initial account of innovation processes based on mass collaboration. The Peering process on which is a characteristics of this collaboration is a negotiated process in which artifacts do play an important roles. In fact "Programming Guidelines" act as a separator between good collaboration and bad one, affect the way developers produce software, and the amount and kind of knowledge that needs to be mobilized.

From this point of view, the ability of source code to enhance the time lasting dimension of conventions, that will need strong interventions to change the yet existing code, is also the ability to bring into code the history of projects, their technical specificity and their power relationships, especially the one between actual participants and prospective ones.

If the OpenSolaris case shows how we can't ignore history and the related practices while

---

[15] GRASS Developers Mailing List message sent Fri, 05 Aug 2005, http://lists.osgeo.org/pipermail/grass-dev/2005-August/019304.html

discussing about the shape formal coding conventions have, the GRASS case shows how this conventions are strictly connected with the precise technological result that can be achieved. .

The examples we provided here are probably not enough for drawing general conclusions and also not representative of a wide range of negotiations. In our future work, we will enlarge our empirical analysis, taking in account controversial situation, litigations on the final shape of the source code and even disagreement on the application of the code conventions.

10. References

[1] Akrich, M., The De-Scription of Technical Objects. In Bijker W. and J. Law (eds.), Shaping Technology/Building Society: Studies in Sociotechnical Change, MIT Press, Cambridge, 1992, 205-224.

[2] Cox, A., Cathedrals, Bazaars and the Town Council. Slashdot, 1998, Retrieved June 8th, 2008, from http://slashdot.org/features/98/10/13/1423253.shtml

[3] Crowston, K. and J. Howison, "The Social Structure of Free and Open Source Software Development", First Monday, 10 (2), 2005, Retrieved June 8th, 2008, from http://www.firstmonday.org/issues/issue10_2/crowston/index.html

[4] De Paoli S. and D'Andrea (2008). How artefacts rule web-based communities: practices of free software development, International Journal of Web Based Communities , Volume 4(2), 199-219.

[5] De Paoli, S., M. Teli and V. D'Andrea (forthcoming). Free and Open Source Licenses in Community Life: Two Empirical Cases, accepted for publication, First Monday

[6] Ducheneaut, N., Socialization in an Open Source software community: A socio-technical analysis, Computer Supported Cooperative Work, 2005, 14 (4), 323 – 368

[7] GRASS Development Team, Geographic Resources Analysis Support System (GRASS) Software. ITC-IRSTt, Trento, Italy, 2006. http://grass.itc.it

[8] GRASS Development Team. SUBMITTING FILE. Retrieved August 23th, 2008, from http://grass.osgeo.org/grass63/source/SUBMITTING

[9] IEEE. Definition of malloc in IEEE Std 1003.1 standard. Retrieved June 8th, 2008, from http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html

[10] Glaser, B. G. and A. L. Strauss, The discovery of grounded theory: strategies for qualitative research, Weidenfeld and Nicolson, London, 1967

[11] Hakken, D., Cyborg@Cyberspace? An Ethnographer Looks to the Future, Routledge, New York, 1999

[12] Hersleb, J. D. and R. E. Grinter, Splitting the organization and integrating the code: Conway's law revisited. In Proceedings of the 21st International Conference on Software Engineering (Los Angeles, California, May 16 – 22, 1999), IEEE Computer Society Press, Los Alamitos, CA, 85 – 95

[13] Kernighan, B. W. and D. M. Ritchie. The C Programming Language, NJ: Prentice Hall, Englewood Cliffs, 1978

[14] Lanzara, G. F. and M. Morner, "Artifacts rule! how organizing happens in open software projects". In Czarniawska, B. and T. Hernes (eds.), Actor Network Theory and Organizing, Copenhagen Business School Press, Copenaghen, 2005, 67-90.

[15] Latour, B., Reassembling the Social. An Introduction to Actor-Network-Theory, Oxford University Press, New York, 2005

[16] Raymond, E. S.. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly and Associates, Sebastopol, CA, 1999

[17] Revision Control. Wikipedia. Retrieved August 23th, 2008, from http://en.wikipedia.org/wiki/Revision_control

[18] Spinellis, D., Code Reading: The Open Source Perspective, Addison-Wesley, 2003

[19] Teli, M., F. Pisanu and D. Hakken. "The Internet as a Library-of-People: For a Cyberethnography of Online Groups" [65 paragraphs]. Forum Qualitative Sozialforschung / Forum: Qualitative Social Research, 2007, 8(3), Art. 33.

[20] Tapscott, D. and A. D. Williams. Wikinomics: How Mass Collaboration Changes Everything, Portfolio, USA, 2006

[21] Weber, S., The Success of Open Source, Harvard University Press, Cambridge, MA, 2004