Collaborative Practices in Free/Open Source Software Community

ICS 280: Theories and Practices of Collaboration
TERM PAPER
Spring 2004

Sushil K Bajracharya

sbajrach@ics.uci.edu

Department of Informatics
School Of Information and Computer Science
University of California, Irvine

Abstract

Free/Open Source Software (FOSS) community has offered an alternative method to build and distribute software, making a great socio-technical as well as economic impact in the software industry. The way FOSS community has evolved and the huge success stories in cases like the GNU/Linux Operating System and the Apache Server Project show how despite of barriers like geographic distribution, differences in time zone and lack of synchronous and face to face communication, contributors (in some cases exceeding hundreds) are able to collaborate and cooperate to achieve their common goal. In this paper we will try to look into some of the issues that explain the factors that contribute to success in FOSS projects.

1 Introduction

Free Software and Open Source Software have some significant differences in their own, especially related to the software licenses they are bound to and their underlying philosophies. Despite of these differences they share some common goals that makes it possible to coin them together to observe collaborative work practices among the contribtutors [1]. A major reason being the way these communities have evolved without any masterplan (or formal structure as such) as open communities and now taken an identity of their own in the industry. This also opens up new oppurtunites for research where methodologies like ethnography with techniques like *grounded theory* and *participant observation* can be practiced to understand and build theories about ogranizational structure and culture fostering open collaboration in FOSS community [ES]. In following sections we will look into some of the work practices, communication strategy, role strucutre, tools and coordination techniques seen in FOSS community.

[1] Thus, we will use the terms FOSS (Free/Open Source Software) and 'Open Source Software' interchangeably

2 The Bazaar style of Open Source model

No discussion about Open Source can escape Eric Raymond's insightful writing - *The Cathedral and The Bazaar* [Ray]. This article is a landmark in the Open Source movement, in particular responsible for attesting and exposing the success of Linus Trovald's model for distributed software development (that he used to develop the Linux kernel) to the rest of the world. Also, the first major industrial adoption of Open Source model [2] was influenced by it. [Ray] is also significant in a way that it actually points out the essence of open collaboration not even quite distinct in the eariler projects of Free Software Foundation (a kind of precursor to the open source movement) led by Richard Stallman. In this section we will see how Raymond describes Open Source model and the social ingredients necessary for it. These points differentiate the Open Source Model of software development as the *Bazaar* style from the *Cathedral* style of building closed propreitary software.

1. *Beyond Brook's Law:* Fred Brook's negative scaling law about software project management that *adding more developers late to a project makes it later* [Bro78] has been in a way defied by Raymond's claim that he says to be Linus's Law - *Given enough eyeballs, all bugs are shallow*. More formally, *Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone*. But this assertion is to be taken carefully, it holds true when other ingredients of Open Source model are there. Moreover the team strucutre as proposed by Raymond is similar to Brook's *surgical team* with a difference that it has a *star* like organization with a core developer or maintainer at the center communicating with many other contributors. So in this model success depends on whether the core maintainer has enough *bandwidth* to cope with the continuous interaction going on . While on the other hand, every contributor does not need to talk to every another one that reduces the number of communication path between the contributors, thus preventing negative scaling of productivity.
2. *Plausible Promise:* Open Source projects do not start from scratch and there needs to be a minimum level of design or code to get started with getting support from other contributors. The initial code or design need not be perfect or fully documented but it must be able to offer a *Plausible Promise* to others with a foreseeable amount of work that it can succeed. The initiator or the cooordinator must possess solid technical skills and should be a competent developer. (S)he does not need to be a genius extraordinaire as such but should have a sixth sense for good engineering, an ability to foresee a least effort path between alternatives and dependencies.
3. *People Skills and acting as a Brain Amplifier:* Raymond presents this remarkable quote by Kropotkin -

> Having been brought up in a serf-owner's family, I entered active life, like all young men of my time, with a great deal of confidence in the necessity of commanding, ordering, scolding, punishing and the like. But when, at an early stage, I had to manage serious enterprises and to deal with (free) men, and when each mistake would lead at once to heavy consequences, I began to appreciate the difference between acting on the principle of command and discipline and acting on the principle of common understanding. The former works admirably in a military parade, but it is worth nothing where real life is concerned, and the aim can be achieved only through the severe effort of many converging wills. [Kro]

Embedded in this quote is the deep meaning about how the organizational culture works

[2] Netscape releasing its soure code for the Netscape Communicator

in Open Source community. Common understanding, not command drives the collaboration. The leader/coordinator/core-developer whatever (s)he is called, should possess excellent *People Skills*.(S)he has to know how to recognize good ideas from other people, recruit people and use them as *brain amplifier* and further make them feel good about it. Instant gratification plays an important role. Contributors should be(are) rewarded instantly by following the *release early, release often and listen to your customers* philoshophy. Also, important to notice is how there is no difference between co-workers and customers. Developers are customers themselves.

4. *Gift Culture and Ego-Boo:* Raymond emphasizes how Open Source community works with *gift culture* that we get in conditions of extreme material abundance, where basically there are not any scarcities. In such cultures, instead of playing market games formed around scarcities, resource allocation and competition, people play for *reputation.* One gains status by giving things away. In Open Source communities contributors are giving away their valuable time and creativity. They exist because they want to share knowledge, skills and gain standing and reputation among peers. Going down to the individual level the major motivation for contribution boils down to the desire to fulfill ones own *ego-boo* (ego) to be called and regarded as *the hacker*.

5. *Harnessing the power of new information infrastructure:* Without the widespread accessibility of the internet Linux would not have been a success and we will see later how computer mediated communication is interwoven in the infrastructure that the internet provides for FOSS projects . Linus Trovalds was smart enough to figure out how to play with the new rules of the internet and act as a mediator between hundreds of increasing contributors.

So this whole analysis boils down to few things - *reinforcing people*, *communication* and *common understanding* are the key success factors in Open Source projects. With this background, we now take a look at the results from some research studies that will verify these propositions.

Stages in Standard SRE*	Open source software communities practice**	Remarks
Elicitation	Assertion of requirements	Requirements are asserted as system capabilities
Analysis	Requirements reading, sense-making, and accountability	Descriptions are not self-complete, readers make their own effort to understand and gather incomplete information
Specification and modeling	Continually emerging webs of software discourse	Functional and non-functional requirements continuously are refined via narrative descriptions supported with domain specific artifacts
Validation	Condensing discourse that hardens and concentrates system functionality and community development	Requirements co-evolve with design and implementation. No formal validations but validations through consensus among participants
Communication	Global access to open software webs	Hypertext style organization of requirements that is accessible globally via platforms like the World Wide Web

* SRE = Software Requirements Engineering, ** = Alternatives to the standard SRE practices

Table 1: Open Source Community's practices for Requirements Elicitation.

3 Work Practices

Requriements Gathering is the initial phase in almost all engineering projects. In all of the convential software process models too *Requirements Elicitation* plays a vital role, especially in resolving the *impedence mismatch* between the customers and the designers/developers. There exists interesting differences between requirements engineering practices in FOSS projects and the standard/classic software engineering process. Table 1 list out these differences. Work practices attributing these fundamental differences have been termed as *software informalisms* [Sca02] that *collectively describe the Web-based descriptions of all the functional and non-functional requirements for open source software systems*.

These software informalisms are the consequences of the communication mechanism available to and the organizational and cultural con-

ventions that has evolved in FOSS community. Eight such software informalisms identfied are briefly summarized below -

1. *Computer based communication tools for community communications:* The communication systems appearing in the form of - (a) messages placed in Web based bulletin board discussion forums; (b) email list servers; (c) network news groups; (d)Internet-based chat (instant messaging) fall under this.

   This communication infrastructure serve as the *place* where all the work pertaining to requirements is done. Messages written and read through these systems, together with references or links to other messages or software webs, help the participants make sense of them.

2. *HowTo guides:* Categorized into *formal* and *informal* HowTo's. These documents capture and condense *How To* perform some behavior , operation, or function with a system, they also serve as a semi-structured narrative that assert or imply end-user requirements. *Formal* HowTo's descriptions include explicit declarations of their purpose as a *HowTo* and may be identified as a system tutorial. Informal HowTo's may appear as a selection, composition, or recomposition of any of the proceedings. These informal HowTo guides may be labeled as a *FAQ*; that is, as a list of frequently asked questions about how a system operates, how to use it, where to find its development status, who developed what, known bugs and workarounds, etc.

3. *Software bug reporting and issue tracking:* This addresses the one of the most obvious and frequent types of discourse that appears with open software systems - discussion about operational problems with the current version of the system implementation. Bugs and other issues (missing functionality, incorrect calculation, etc). Communication media facilitating this are email, bug report bulletin boards, threaded email discussion lists and related bug/issue tracking mechanisms like like *Bugzilla*.

4. *External publications in the domain:* This includes external publications that describe open software available for consumption by the public or by community members such as - technical articles, books (less common), professional articles in trade publications (like the *Linux Journal* or *Game Developer* and academic articles that are refereed and appear in conference proceedings or scholarly journals.

5. *Traditional software system documentation:* This constitutes more traditional artifacts like online manuals and instruction sets.

6. *Software extension mechanism and architectures:* Documentation of the APIs and source code fall into this.

7. *Scenarios of usage as Linked web pages as scenarios of usage:* Artifacts created and linked together in a hypermedia like structure by the community members, like screenshots, guided tours, or navigational click-through sequences with supplementary narrative descriptions of how the system operates, or how it appears to a user when used, constitute this category.

8. *Open software web sites and source webs:* All of the open source projects use the world Wide Web as a global communication platform. The *contents* and *links* that constitues the elements of the various interlinked web sources (Web-sites) related to an open source project serve as a repertiore-at-large.

These software informalisms reveal that the work practices in FOSS community is a collaborative effort based on computer mediated communication that forms a complex social structure in the cyberspace. Open source software is built upon these informal and light weight or non rigorous software informalisms because of the fact that these informalisms are sufficient and extensible enough so that the contributors can use and create tools and artifacts according to their ease.

3.1 Genres of communication

Eight of the above software informalisms listed serves particular purposes for communication in the FOSS community, most of them can be distinctly identified by their specific forms and every one of them has specific purpose that leads to some discourse among the paricipants of the FOSS community. Each of these eight software informalisms can be identified either as *genres* of communication actions or a *repertoire-subset* [3] of the FOSS *genre repertoire* for communication. As a whole these eight informalisms constitute the entire *repertoire* for FOSS community. We take this notion of *genre* of organiational communication as defined by Orlikowski and Yates,

> a distinctive type of communication action, characterized by a socially recognized communicative *purpose* and common aspects of *form* ..Such communicative actions are typified responses to recurrent situations within organizations. [OY94]

A classification of these informalisms as a genre or a repertoire subset is given in Table 2. They are further classified as whether they can be identified with the distinct form they have or the purpose they serve. Sometimes the contributors also make sense of the communication genre/artifact depending on the context so context is also added to form and structure used to identify a particular genre or a repertoire-subset. These communiation tools are sufficient to aid all the *sensemaking* [Wei95] in FOSS community, making the participants aware of their common goals and shared understanding.

FOSS projects of all kind ranging from scientific to academic to entertainment (gaming) rely sufficiently on these informal and light weight or non rigorous software informalisms for community building and the product development process. [Sca02]

[3] By repertoire-subset we mean they further can constitute several genres inside them and they are just the subset of the genre repertoire not the entire repertoire.

Informalisms	genre (G)/ repertoire-subset(RS)	Identified by
computer based communication tools	RS	form, purpose, context
How-to guides	G	form, purpose
Software Bug Reports/Issue Tracking	G	form, purpose
External publications	RS	form, purpose
Traditional Software system Documentation	RS	purpose, form
Software Extension Mechanism/Architectures	G	form
Scenarios of usage//linked web pages	RS	purpose, context
open source web site and source webs	RS	purpose, context

Table 2: Classifying software informalisms to genre/repertoire-subset.

4 Role Structure

The basic role structure in an Open Source project is a central moderator or core developer with many contributors communicating with the central authority. This has been Linus Trovald's model for developing the Linux kernel. But Open Source projects have come a long way, these days we see big industry giants like IBM, Oracle and Sun Microsystems participating and contributing in Open Source projects. This brings in new economic and oraganizational implications. But, in any case the freedom vested upon the developers, so called hackers remain same. FOSS projects still follow the principle of meritocracy - the more one puts in, the more he gets in terms of value and respect and peers will take his/her decisions with great regard. Decison making still remains democratic where a critical design change or code modification issues are usually resolved with open discourse and techniques like e-mail voting [Fie99]. The higher level of organizational strucutre with roles titles like *Board of Directors, Steering Committee* are now common in Open Source projects. Mostly these higher level of structures exist to manage extramural issues like funding, legalities and industry ties. Whereas, in newer projects like Eclipse (`wwww.eclipse.org`), where the initiative have been from a huge industry player (IBM), the higher organizational authorities have more power to control and plan major decisions regarding design and the future of the product. A detail analysis of emerging role stuctures in

FOSS projects is out of scope of this document, so we conclude this section by enumerating the major role structre that exist in most of FOSS projects. This classification is based on the results from [NYN+02].

1. *Passive User:* Casual users of the system, same as most of the commercial software users. They seek quality and flexible changes as need arises.

2. *Reader:* Readers are active users of the system. They try to understand the system by reading the source code and have roles as of peer reviewers.

3. *Bug Reporter:* Acting as testers, responsible for discovering but not necessarily fixing the bugs.

4. *Bug Fixer:* They fix the bugs that are either discovered by themselves or reported by Bug Reporters. They have necessary knowledge about the source code related to the bug-fix they are making.

5. *Peripheral Developer:* They contribute towards new functionality or features to the existing system. However, their contributions are irregular and last shorter.

6. *Active Developer:* As one of the major development forces of FOSS systems active developers regularly contribute to new features and bug-fixes.

7. *Core Member/Maintainers:* They are responsible for the overall guidance and coordinating the development. Core Members are involved with the project relatively for a long period of time and make significant contributions to the development and evolution of the system.

8. *Project Leader:* Project Leader is often the person who initiates the project and is responsible for the vision and overall direction of the project.

It should be noted that not all of these roles structures are visible in all FOSS projects and not always go by the same name.

5 Not so Extreme Collaboration

It has been found that large [4] projects with many interdependencies and complex requirements often tend to benefit by practicing *Extreme Collaboration* [Mar02]. Bringing this notion to software development eXtreme Programming (XP) [Bec00] [chr03] makes a very close match. Besides relying on normal electronic information tracking tools like e-mail, project tracking tools, spreadsheet or webpage, XP advocates the use of physical artifacts and facilities like - (i) Story Cards - for customers to put down 'What should be done' , (ii) Task Cards - for developers to put down 'How should it be done' and (iii) The Bullpen (War Room). Such practice carry in itself the essence of Distributed Cognition [HHK00] and the practice of radical collocation for communication [TCKO00]. Similarly, the role structure in XP puts the *Customer* in constant communication with the *Developers* and additionaly suuplementary roles like of the *Coach* (mentor) and *Tracker* are defined. XP's philoshopy that a software project can be managed in terms of four variables *time, scope, resources* and *quality* (any one of these cannot be overemphasized without considering the effect on another) is also very similar to *timeboxing* [TCKO00].

It is interesting observation to make how some of the XP principles match with FOSS practices, and when people try to advocate XP practices in FOSS projects, they normalize some of the essential ethos of XP.

The core of XP lies in its 12 practices, (4 coding, 4 developer and 4 Business) given below -

1. Code and Design Simply
2. Refactor Mercilessly
3. Develop Coding Standards
4. Develop a Common Vocabulary (Metaphor)
5. Adopt Test Driven Development
6. Practice Pair Programming
7. Adopt Collective Code Ownership
8. Integrate Continually
9. Add a customer to the team
10. Play the planning game
11. Release Regularly
12. Work at a substantial Pace

[4] large does not necessarily imply large number of participants

XP Practices	DXP Solution
Pair Programming	Videoconferencing, application sharing and personal familiarity
Planning Game	Application/Desktop sharing
On-Site Customer	*Virtual* on-site customer via videoconferencing and application sharing
Continuos Integration	Remote access to integration machines

Table 3: DXP (Distributed eXtreme Programming) Solutions to XP Practices that require physical collocation [MKL01]

Almost all of these practices require constant *communication* and *rapid feedback*. Among above 12, four practices rely on physical collocation and face-face-face communication, that can be replaced to some degree using electronic commincation media as listed in Table 3. Table 3 is a generalization of alternatives sought out for communication media in extending XP to DXP (Distributed eXtreme Programming) in a globally distributed project [MKL01]. The findings regarding extending XP to DXP are [MKL01] -

1. In conditions that limit such physical collocation, combination of synchronous (videoconferencing) and asynchronous (e-mail) mode of electronic communication is effective.

2. Configuration Management (CM) Tools help to a great extent in maintaining code integrity and facilitates parallel development. Leveraging communication facilities beyond CM tools, like using a simple *e-mail token to serialize change access* while working on common code simplifies the task. This phenomenon is also very well described in [dRD03] where team members seek out solutions other than those provided by CM tools to resolve conflicts.

Although the solutions outlined in Table 3 cannot provide the same communication experience as in physically collocated environment, for some categories of projects that already has a *loose coupling between team members* such technologies are sufficient to create a complex web of socio-technical interaction among the participants [Sca02]. Arguments given in favour of the fact that XP can be practiced in large open source source communities more or less exploit this fact [KL01]. As already stated, among the 12 XP practices, four are the most demanding of physical collocation. Here are some of the arguments supporting the fact that they can be compromised in open-distributed projects -

1. *The Role of the customers are not explicitly visible in such open projects.*

   Most of the times, developers are customers and issues can be resolved by participating in communication through electronic media that furthermore, provides mechanism for persistence (aiding storage, retrieval and search facility) of on-going communication. This role of customers in FOSS has been attested by [Sca02] and also as,

   > *Hypothesis 6:* In successful open source developments, the developers will also be users of the software. [AMH02]

2. *Pair Programming is an issue that seems to have no better alternative in such projects.*

   Solutions like remote pair programming using collaborative editors or even practicing serial pair programming rather than parallel pair programming can be considered as alternatives. (The argument for this is, four eyes sees better than two, but do you need to see at the same time?). In FOSS projects pair programming and radical collocation has not been an issue of much importance, however software inspections with peer reviews remains an important activity, they just are not necessarily conducted syncronously with peers being collocated.

3. *Moderator based software builds and reliance in modern CM tools*

   Mechanisms for *continuos integration* are manifested as a set of Release Management policies characterized by release authority, versioning, prerelease testing, approval of releases, distribution, and formats. [Ere]

4. *High standard coding guidelines with a proper gatekeeper process (norms that all participants follow) and well-mainained user groups with fast responses for good planning.*

> Again the usage of specific tools like BugZilla for bug-reports, well managed howt-to guides and design documents that are all available publicly helps to a great extent in planning.

Besides these difficult to practice principles in case the team is distributed, other are easier to adopt in open-distributed projects, for instance the notion of metaphor is very well captured by using *Design-Patterns* [GHJV95] as a medium of communication as well as foundation for development [KL01]. Also, arguments supporting mechanisms to make XP scale to large projects [Cro01] very well matches the way FOSS community works as we see below where we compare the extension mechanisms given in [Cro01] to their similarities in FOSS projects by relating them to some of the main hypotheses about FOSS projects given in [AMH02].

1. *Loosely Coupled Teams* - Large teams are to be broken into a collection of loosely coupled collaborating teams. The 'prime' team should not dictate of impose a process on the 'subcontracting' teams. The 'prime' team should impose the feature roll-out plan on the teams.

   This matches *Hypotheses 2* and *3*,

   > *Hypothesis 2:* For projects that are so large that 10 to 15 developers cannot write 80% of the code in a reasonable time frame, a strict code ownership policy will have to be adopted to separate the work of additional groups, creating, in effect, several related OSS projects. [AMH02]

   > *Hypothesis 3:* In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems. [AMH02]

2. *Team Co-ordination Layer* - Additional layer, outside the XP, to support team interaction. This replaces metaphor, pair programming, collective code ownership and on-site customer practices with the following - (i)*Up-Front Architectural Lite:* An incomplete but do-able architectural description of the project that ensures that the design knowledge permeates the team (ii)*Liaison:* Members of various teams joining forces to develop the architectural-lite. (iii)*Team-Wise Planning Game (TPG):* Allowing each sub-team to practice XP and facilitating planning across sub-teams. This includes concepts like *proxy customer*, where a sub-team on behalf of a customer can participate in TPG.

   This is supported by *Hypothesis 1* from [AMH02],

   > Open source developments will have a core of developers who control the code base. This core will be no larger than 10 to 15 people, and will create approximately 80% or more of the new functionality.

The conclusion we can draw from these studies is that Radical Collocation is not an absoulute necessity for FOSS projects. However almost all the charateristics of extreme collaboration is seen in FOSS participants, for instance,

> *Hypothesis 7:* OSS developments exhibit very rapid responses to customer problems. [AMH02]

matches the central theme in XP - *communication and rapid feedback*. The only difference is, mostly the communication is asynchronous and participants are distributed geographically. Besides these all the elements of discourse are quite similar that takes place through the web based artifacts for communication. It is not the case that FOSS community do not need Extreme Collaboration because they are dealing with less complex projects. No one can refute that building an operating systems kernel or software for deep x-ray astronomy [Sca02] lack interdependencies but the contributors somewhat have managed to get used to mostly asynchronous mode of computer mediated communication and live by their rules that donot hinder collaboration. Also the fact that FOSS development is not driven by strict deadlines and mangerial pressures makes

it sustainable in the world of asynchronous distributed communication.

Empirical studies show global dispersion and distance among participants introduce delay in collaborative work [HMFG00]. The solutions that are outlined to cope with such delays highlight the necessity for ease in finding *expert* help in need, support for *awareness* and *richer interaction*. The web-based discourse in FOSS is in itself embodiment of these concepts. For example, all projects have threaded discussion boards where sub-sections led by area experts are distinct to all contributors and people know where to go in case of problems. Interactions with internet based tools whether they are synchronous (IRC chat), asynchronous (email, messageboards), private(point to point) or public can go beyond anything required as we will see in next section how people participate in philosophical debates and resolves issues related to culture and organizational values using such computer mediated communication tools.

6 Conflicts and Negotiations

Often times, in FOSS teams we get to see a different picture of conflicts, debates and negotiations taking place. Especially how computer mediated communication (CMC) provide the platform for initiating discussions as well as resolving conflicts fostering collaboration.

[ES03] presents two examples of such articulation among participants of GNUe, an open source project targeting development of a freely available ERP (Enterprise Resource Planning) solution. [ES] extends this example to refute and support some of the earlier assertions made about conflicts and negotiation in collborative work practices. Both of the debates took place in the GNUe IRC channel. All the discussions going on in the IRC channel are stored persistently and anyone can read previod message logs and start a new discussion later in the IRC channel itself or mailing lists.

The first debate was initiated by an outsider to GNUe indicating his concern about *use of a non-free graphics tool* to produce a graphic on the GNUe website. The second debate, also of similar nature started where a frequent contributor put forward his complaint about needing to install a non-free software to edit the documentation for the project. The settlement in the first case was made after a series of a day long discussion with the user of the non-free graphics tool accepting to produce the graphics later using a free software. Settlement in the second case took more time (3 days) and also the conflict delayed the work of some contributors. The instigator, an insider backed down due to peer pressure. However it was decided that a completely free version of the tool will be used in future when available and, the documentation distributed would not require the users to install non free software.*There appeared to be less resistance to change regarding free versus non-free tools when the change involved an impact to one persons work and was not of a procedural nature.*

Following points are inferred from the conflicts and negotiation techniques in above examples -

- Immediate acceptance of outside contributors
- Belief in free software, value in community and value in cooperative work
- Involvement in detailed philoshopical discussions not taken as interruption to the work but as a reinforcement of values and beliefs

Drawing conclusion from these results [ES] presents an analysis of few assertions about conflicts and coordination in FOSS cooperative work -

1. *Anonymity and physical separation does not always contribute to conflict.*
   Although conflicts occur in daily work practices, strong cultural ties diminish the effects of anonymity and distance separation. In addition, the frequent contributors are not really anonymous as their credentials are available on public web sites. Authors agree with findings by Lea and Spears,

   > They suggest that the social context influences the occurrence of de-individuation. If de-individuation occurs with immersion in a group, then this enhances the salience of the

group and strengthens its norms. However, if the group identity is not already established, then de-individuation only serves to strengthen ones sense of individuality, and weakens group norms.[LS91]

2. *It is not always true that conflicts are unlikely to be resolved if participants argue from entrenched positions.*
   This assertion refutes an earlier claim made by Easterbrook et al [SMEC93],[Pac]. It highlights the fact that the recommendation for handling *competitive conflict* by separating people from their positions to help support creativity in resolving the conflict is not always possible or desirable especially in a virtual community. In such communities the values in community and cooperative work motivates people to try and resolve differences without a third party/manager.

3. *Articulating conflicts helps in its resolution*
   With this assertion the authors in [ES] agree with previous studies like that in [SMEC93] and [Pac]. They agree *differentiation* - a group process of identifying and understanding the dimensions of a conflict by making the conflict explicit, identifying issues involved, and recognition of individual views by other members of the group play an important role in conflict resolution as illustrated by the two examples given above.

With all our discussions so far it is of no surprise that strong beliefs in values and culture of organization and strong commitment to contribution exist absolutely persistently among members of FOSS teams and it is this common belief that leads them to pass over any conflicts or debates.

7 Tools in Practice

A very important necessity to succeed in a project is the availability of appropriate tools. Richard Stallman knew this very well and that is the reason he started out GNU with quintessential tools like the GNU C compiler and the legendary Emacs editor. This tradition follows in FOSS community and now it is full of all categories of tools required for dvelopment. *To a large extent, the open source culture and methodology are conveyed to new developers via the toolset itself, and through the demonstrated usage of these tools on existing projects* [Rob]. For a FOSS development project today there exists almost all kinds of software engineering tools needed such as those for - Version Control, Issue Tracking and Technical Support, Technical Discussions and Rationale, Build Systems, Design and Code Generation, Quality Assurance and Collaborative Development Environments. [Rob]

8 Conclusions

In this paper we have seen how the thesis laid out by Raymond [Ray] before the massive proliferation of FOSS community holds out to be true as we saw results from some other studies in successful FOSS projects.

Having talked all good things about FOSS projects, success stories in these communities are to be taken with caution. Not always FOSS model leads to success. There are 82,883 registered Open Source projects and 866,021 registerd users in Sourceforge (`www.sourceforge.net`, an online repository of FOSS projects) alone to this day of writing. Definitely not all of them are guaranteed success or even completion and, most have been or will be defunct within a short period of incubation. Furthermore, not all FOSS projects have hundreds or thousands of contributors, [Kri] showed that *the vast majority of mature OSS programs are developed by a small number of individuals*. In order for a FOSS project to succeed all the necessary ingredients necessary has to be there. As Jamie Zawinski talks about reasons for his failure in the Mozilla project,

> People only really contribute when they get something out of it. When someone is first beginning to contribute, they especially need to see some kind of payback, some kind of positive reinforcement, right away... ...you can't take a dying project, sprinkle it with the magic pixie dust of "open source," and have everything magically work out. [Zaw99]

Zawinski's words are justified with his own reasons, but yet Mozilla as such cannot be taken as a failure as it has definitely given lot of things, one of

the significant contributions being to be able to break Microsoft's monopoly on the browser market.

Another important requirement that we have been discussing from the begining ,getting large user base is important -

> *Hypothesis 4:* Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects. [AMH02]

Seeing how the role of a user fits in FOSS software we cannot say FOSS software is a panacea for all kinds software need today. Usability is still an issue. This side of FOSS is still improving. It is not that FOSS is of low quality, in most of the successful cases it is the opposite. But, one has to invest some effort to be used to FOSS tools, one has to learn to play by the rules of the community. How many of users in a need of an operating system will be willing to receive a patch for a kernel module and recompile his system for a performance gain or a bug fix ? No matter what we claim the truth is, there still seems to be a divide, a divide between *normal users* and the so called *geeks*. It might be this fact that proprietary software vendors understand and know how to exploit to make people pay for what might be freely available. It seems it will take time for FOSS software to reach an ordinary user's reach and it is not yet clear where non-technical people or normal software users fit in the realm of FOSS. Can they fit in?

Also is important how FOSS community offer new interdisciplinary research oppurtunities where techniques like ethnography and empirical methods following theories from antropology and social sciences can be used to come up with theories that explain the complex social and technical embeddings of collaborative practices in FOSS community. As Bowker gives his argument for the need of such research endeavors,

> The open source movement has seen as running counter to the dominance of large centralized industries - the argument goes that it puts power over media back into the hands of the people in a way that might truly transform capitalist society. This promise of cyberdemocracy is integrally social, political and technical. While there is much talk of an 'information revolution', there is not enough the ways in which people and communities are constituted by the new infrastructure. [Bow]

Consequently such research efforts will help discover new grounds for software development that would try to reach places convential software engineering and management have not reached so far.

References

[AMH02] Roy T Fielding Audris Mockus and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, 2002.

[Bec00] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[Bow] Geoffrey Bowker. The new knowledge economy and science and technology policy. *University of California, San Diego*.

[Bro78] Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Softw*. Addison-Wesley Longman Publishing Co., Inc., 1978.

[chr03] chromatic. *Extreme Programming - Pocket Guide*. O'reilly And Associates, first edition, July 2003.

[Cro01] R. Crocker. The 5 reasons xp cant scale and what to do about them. *XP2001, Italy*, May 21-23 2001.

[dRD03] Cleidson R. B. de Souza, David Redmiles, and Paul Dourish. "breaking the code", moving between private and public work in collaborative software development. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 105–114. ACM Press, 2003.

[Ere] Justin R. Erenkrantz. Release management within open source projects. *Institute for Software Research, University of California, Irvine*.

[ES] Margaret S. Elliott and Walt Scacchi. Free software: A case study of software development in virtual organizational culture. *ISR Technical Report UCI-ISR-03-6*.

[ES03] Margaret S. Elliott and Walt Scacchi. Free software developers as an occupational community: resolving conflicts and fostering collaboration. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 21–30. ACM Press, 2003.

[Fie99] Roy T. Fielding. Shared leadership in the apache project. *Commun. ACM*, 42(4):42–43, 1999.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[HHK00] James Hollan, Edwin Hutchins, and David Kirsh. Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.*, 7(2):174–196, 2000.

[HMFG00] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. Distance, dependencies, and delay in a global collaboration. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 319–328. ACM Press, 2000.

[KL01] Michael Kircher and David L. Levine. The xp of tao: extreme programming of large, open-source frameworks. pages 463–485, 2001.

[Kri] Sandeep Krishnamurthy. Cave or community?: An empirical examination of 100 mature open source projects. *First Monday*.

[Kro] Peter Kropotkin. *The Memoirs of a Revolutionist*. Fredonia Books (NL); (April 1, 2002).

[LS91] M. Lea and R Spears. Computer-mediated communication, de-individuation and group decision-making. *International Journal of Man-Machine Studies*, 34:282–301, 1991.

[Mar02] Gloria Mark. Extreme collaboration. *Commun. ACM*, 45(6):89–93, 2002.

[MKL01] A. Corsaro M. Kircher, P. Jain and D. Levine. Distributed extreme programming. *XP2001, Italy*, May 21-23 2001.

[NYN+02] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*, pages 76–85. ACM Press, 2002.

[OY94] Wanda J. Orlikowski and JoAnne Yates. Genre rrepertoire: The structuring of communicative practices in organizations. *Administrative Science Quarterly*, 39(4):541–574, December 1994.

[Pac] R. C. Pace. Personalized and depersonalized conflict in small group discussions: An examination of differentiation. *Small Group Research, 21(1), 79-96*.

[Ray] Eric S Raymond. The catheral and the bazaar. *http://www.catb.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/*.

[Rob] Jason E. Robbins. Adopting open source software engineering (osse) practices by adopting osse tools. *Institute of Software Research,University of California, Irvine*.

[Sca02] Walt Scacchi. Understanding the requirements for developing open source software systems. *IEEE Proceedings - Software*, 149(1):24–39, 2002.

[SMEC93] J. S. Goodlet M. Plowman M. Sharples S. M. Easterbrook, E. E. Beck and C. C.Wood. A survey of empirical studies of conflict, cscw: Cooperation or conflict? *(pp. 1–68). London: Springer-Verlag*, 1993.

[TCKO00] Stephanie Teasley, Lisa Covi, M. S. Krishnan, and Judith S. Olson. How does radical collocation help a team succeed? In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 339–346. ACM Press, 2000.

[Wei95] Karl E. Weick. *Sensemaking in organizations*. Thousand Oaks. CA: Sage, 1995.

[Zaw99] J Zawinski. Resignation and postmortem. *URL: http://www.jwz.org/gruntle/nomo.html*, 1999.