# Flexible organizing and identity fragmentation in Free/Open Source Software development

Mehmet Gençer*, Bülent Özel*, V. Sinan Tunalıoğlu*, Beyza Oba**.
* İstanbul Bilgi University, Department of Computer Science
** İstanbul Bilgi University, Institute of Social Sciences

June 1, 2006

**Abstract**

Free/Open Source Software(FOSS) development is regarded as one of the most noteworthy contemporary collective innovation processes. Software produced in FOSS projects are available to everyone for all sorts of use and modification, only with some restrictions on how the modified versions can be licensed. This enables developer groups to start a new development branch for their own needs by choosing to develop separately from the parent project, which is called 'forking'. In this study, we have examined the histories of several project forks in FOSS. We have approached fragmentation of FOSS community and group identities through forking as a technology for problem solving and distributed innovation, attempted to identify factors that this fragmentation is contingent upon and understand construction of identities during the fork. Social power struggles, differences in interest, mobility and licensing terms were found to be the major reasons which trigger forking decisions. The variety of forking situations we have examined exhibits two levels of positioning with other actors(individuals, cliques, projects) in order to construct their identities: (1)higher distance with some actors for maintaining specialization, independence or competitive advantage, and (2)closer distance for maintaining compatibility and reciprocal benefit with some others. Norms and values in FOSS community level, which are partly reflected in license terms, serve as the constitutional basis which maintain coherence of this system through the fragmentation process.

# 1 Introduction

Success of Free/Open Source Software(FOSS) development in producing high quality software at high speed has attracted attention of an increasing number of researchers in recent years. The exchange of know-how, experiences, ideas, and the code itself within such project oriented clan organization is regarded as one of the most noteworthy contemporary collective innovation process[Hippel & Krogh 2003].

Software production activity, and FOSS development in particular, takes place in a digital medium where processes and outcomes are recorded carefully, and available for analysis. In FOSS projects, in addition to such information, the software produced is equally accessible to all parties in contract[Lehmann 2004]. Software licenses used in FOSS allow reuse of software under legal terms that are very relaxed compared to proprietary software licenses. This equality and free access enables all actors in FOSS to develop on their agenda appropriate for their needs, regulated by the FOSS software license terms and established community practices which constitute the identity of FOSS community. The term 'forking' is used for the process of initiating a separate branch of development by starting off with software taken from an existing project. Lack of legal barriers and low cost of forking in FOSS, as opposed to proprietary software World, is highly utilitarian in distributed innovation. Accessibility of forking enables actors equally to attempt exploiting "structural holes"[Burt 1995].

As each software project within FOSS develop their own manners and practices on top of FOSS practices, the group identity is incrementally fragmented but remains consistent with the larger community. In this study, we have examined the histories of several project forks in FOSS. We have approached identity fragmentation through forking as a technology for problem solving, increased flexibility and creativity, and attempted to identify factors that fragmentation is contingent upon. We have also looked at the FOSS licensing as a mechanism that drives coherence of the system.

Deriving upon the discussion of Schoemaker[Schoemaker, 2003], this study considers FOSS Projects as an extreme form of operating as a network. Additionally, the projects offer their developers, maintainers, and users an identification within FOSS Community. Software licenses as the governing actor of FOSS project oriented organizations offer a framework for the projects and social identity of the individuals. This relationship is examined in the sense it is discussed by Hogg[Hogg 1996] and Goia[Goia 1998] about identification of the

individuals with the clan organization.

The study follows qualitative analysis to study the forking phenomenon as a technology for flexibility within FOSS development processes. Exploiting various source of data from project Web sites, discussion mail lists, articles, project newsletters, and by conducting structured e-mail interviews with key figures, discursive patterns of forking is worked out.

The study further analyzes the distancing tendencies of the forked projects, in order to understand the discourse of forking decisions and identify contingencies for fragmentation. The variety of forking situations we have examined exhibits two levels of distancing with other actors(individuals, cliques, projects): (1)higher distance with some actors for maintaining specialization, independence or competitive advantage, and (2)closer distance for maintaining compatibility and reciprocal benefit with some others.

However, race for these opportunities may have several outcomes: (1)the forked or the forker may die and the other survives, (2)both survive either by co-evolving and staying compatible with its siblings, (3)differentiate and become incompatible with the other siblings, or (4)projects merge and the fork is unforked. While this process is increasing variation in the corpus of FOSS projects, contracting through FOSS license terms serves as a governance mechanism to maintain coherence of this system by mediating access to resources in a way that is stimulating equally the interoperability and diversity of actors inside the boundary of FOSS[Hippel & Krogh 2003].

Public licenses used in FOSS surfaces up as the essential code of conduct which manages the relationship between the open source movement as the organization and the contracting individuals. The very existence of several such licenses is the strongest indicator of FOSS' capacity for adapting. There are studies which indicate its success in doing so[Colazo et.al. 2005].

The next section presents an overview of FOSS. Section 3 lays out processes at different levels that stimulate diversity and innovation in FOSS World. A conceptualization of FOSS processes for organization and identity theories is attempted in section 4. A summary of data and research approach is presented in 5, followed by the results in section 6. A discussion of results in section 7 precedes summary of conclusions given in last section.

## 2 FOSS Background

Here 'Source', within the phrase "Open Source Software", refers to the source code of the software, which is the human-readable instructions that make up the software. In order to use software, source code has to be compiled into a machine-readable code, which a computer can interpret. And 'Open' refers to the notion that the source code is kept open and available to anyone.

> A relatively quiet, but potentially important phenomenon related to human collaboration occurred at the end of the 20th century in the field of computer science. The phenomenon, called open source (OS) software development, has the potential to change, perhaps dramatically, the way humans work together to solve complex problems in general, and specifically in areas of public policy and management[Schweik & Semenov 2003].

In fact, the free/open source software development as an activity of openly sharing source code in a collaborative environment is not new. The very idea of developing open source software has existed since the early days of programming. In those early days of programming, learning from each other, modifying and improving each others software was a prevalent practice amongst MIT students and scholars. Thus, it is crucial to note that this collaborative development style originated in the scientific and academic research organizations of the 1960's and 1970's[Weber 2003]

The first popular instance of this collaborative development process was realized in the construction and evolution of the Unix operating system. Unix operating system was developed by two programmers from Bell Labs, Ken Thompson and Dennis Ritchie. They had voluntarily worked on an earlier incomplete version of Unix called Multics (Multiplexed Information and Computing Service), when its development was no longer supported by its sponsors (MIT, Bell Labs and General Electric)[Weber 2003].

Later on, universities were allowed to use Unix free of charge. The source code was free and the universities contributed back to Unix[Edwards 2001]. Two highly important points were being epitomized within this sharing and collaborating process. First, Unix was made available in readable source code (written by C programming language) form rather than unreadable binary machine code. This meant that it was possible to understand the inner workings of Unix more easily [Raymond 1999]. Second, Unix was licensed as is and was not subject to

any type of warranty, service or support[Weber 2003].

In second half of 1970's and through the usage of this decentralized voluntarily collaborative development model, Unix transformed and forked into many variations (for example BSD fork in 1976)[Weber 2003].

By the late 1970s Unix were very popular and AT&T (parent corporation of Bell Labs where Thompson and Ritchie were employees) for the first time decided not to release its source code anymore to commercially exploit its popularity.

In reaction to such closing and proprietarizing acts on source code, Richard Stallman from MIT AI Lab, pioneered free software movement and founded Free Software Foundation (FSF). Free software, as defined by the FSF, is a program that grants various freedoms to its users [FSF 1991]:

- Freedom to run the program for any purpose
- Freedom to study and adapt the code for personal use
- Freedom to redistribute copies of the program, either gratis or for a fee
- Freedom to distribute improved or modified versions of the program to the public

This definition distinguishes F/OSS not only from proprietary(closed) software, which is distributed in binary(machine-code) form under a license that sets clear restrictions to its usage and requires payment of a fee, but also from freeware and shareware, that can both be downloaded free of charge but do not allow access to the source code.

One of the most important results of the FSF was its creation of the GNU[1] Public License (GPL). GPL is one of the most common software license used by F/OSS applications.

Apart from the GPL, there are many other types of F/OSS distribution licenses. Nevertheless, it is possible to name those licenses either liberal license type or viral license type [Weber 2003]. The GPL created by Richard Stallman is known as the viral license type. It aims to ensure that source code should always be open and modifiable in order to foster and simulate collaborative innovation. The very tenet of GPL is that it requires all subsequent developments on a piece of code distributed under the GPL must open its source code in all of its

[1] GNU stands for the recursive acronym 'GNU is Not Unix'

distributions and license them under the same terms. Therefore, the term viral is attributed to the GPL.

In liberal license type, for instance in Berkeley Software Distribution License (BSD), any derivative work based on a piece of BSD code has no further legal obligations. Hence the very tenet of this license type is that any derivative work based on BSD code has the freedom to show or hide its source when distributed. Thus it gives the option for the modified code to be used commercially[Weber 2003].

# 3 Diversity and Innovation in FOSS Development

There is an increasing level of activity within boundaries of FOSS. Most of these activities are related to software development. However, in recent years there is a proliferation of activity in other domains such as developing software documentation, educational materials, graphic arts, audio, etc., some of which are closely connected to FOSS community and most are released with intellectual licenses that resemble licenses used in FOSS.

A well known example in software side is SourceForge.net, which provides free services over the Internet to FOSS developer community. SourceForge currently hosts more than 100.000 projects, although a considerable number of these projects are inactive. There are more than one million users and developers registered to SourceForge. Another example is Debian project which produces an operating system that is an alternative to commercial systems such as Microsoft ©Windows. Debian project packages more than 20.000 pieces of FOSS software for use as a part of their system. It has more than 1.000 software developers all over the World, and has a key role in FOSS adoption.

These numbers demonstrate significance of FOSS activity. Aside from projects run by volunteers, there is now a considerable number of FOSS projects which are supported by private companies, either through financing or direct involvement in developing code. Once considered marginal, open source received much attention lately from private enterprises and public agencies as changing competitive dynamics of network industries, such as software, required different models for production[Garud & Kumaraswamy 1993].

The fact that open source development model and Copyleft licenses lower the barriers to involvement caused the open source model to be considered as

an effective governance mechanism for software production and its subsequent adoption by increasing number of actors in the field[West 2002]. It is suggested that production model of FOSS can be significant for domains other than software that require high level of networking and lowered barriers to information flow[Hippel & Krogh 2003]. FOSS emerges as a community-based alternative to knowledge creation which suits the software production practice quite well, as compared to firm-based models[Lee 2003, Hippel & Krogh 2003]. In this practice, written and unwritten rules encourage critical evaluation of existing knowledge, innovation, and rapid elimination of error. In such a community setting, the emergence of structure reflects the operation of organizing principles that act as "genetic rules"[Lee 2003].

## 3.1 Forking as a Community Practice

Organizing principles of FOSS community has its roots in the "hacker culture" (see Himmanen and Castells for an extensive review [Himmanen 2001, Castells 2001]) and in the times where software development practice took place in academic environment in which peer-review was an established form. Contrary to its widespread perception, the term 'hacker' refers to a talented programmer who reaches a goal by employing a series of modifications to exploit or extend existing software code and computing resources. A short glossary of these and other relevant terms in software development is presented in table 1 for convenience. For the purposes of our discussion it is sufficient to note that a common theme in hacker culture and academic software development is tradition of sharing software and openness to peer review of software code.

These organizing principles, which are manifested in open source licenses, allow everyone to modify and reuse software source code according to their own agenda. In fact the core of FOSS development is a private-collective innovation model in which each developer contribute to a software project in order to extend its capabilities for his/her own needs[Hippel & Krogh 2003]. Having many developers see source code of software additionally creates the potential of at least one to notice errors and fix them. As such although FOSS development may seem rather chaotic at first, quality of software produced in FOSS is highly recognized. The FOSS approach seems to avoid inefficiencies of a strong intellectual property regime and implement concurrent design and testing of software.

In some cases where various reasons lead to difficulties in accommodating

Table 1: A glossary of software development terminology

**source code** (commonly just source or code) series of statements written in some human-readable computer programming language, which after some possible processing produces the executable programs that are understood by the computer.

**developer** (or programmer) people who write source code of computer software.

**hacker** talented programmer who is capable of reaching a goal by employing a series of modifications to exploit or extend existing software code and computing resources.

**free/open source software** software that is (1) distributed with a license which allows copying and redistribution (after possible modification), and (2) whose source code is also freely accessible.

**GPL** General Public License. One of the oldest and most strict FOSS licenses.

**FSF** Free Software Foundation was formed by Richard Stallman, author of GPL, and is the center where many important elements of what is known as Linux system is produced under GPL license.

some contributions or implementing certain features, a separate project may be 'forked'. Wikipedia gives the following definition for forking:

> In software engineering, a project fork or branch happens when a developer (or a group of them) takes code from a project and starts to develop independently of the rest.

The availability of forking is seen as one of the primary merits in FOSS by developers. Through its employment it is possible for developers to create (sub)projects that have a focus which otherwise would not be possible if the same development effort would be forced to fit into another existing software project.

We should note that there is a hazard that forks may lead to inefficient allocation of developer workforce. Such things actually happen as the number of 'dead' projects in SourceForge.net indicates. On the other hand forking does not take place in an totally uncontrolled manner. Open source communities consist of governance structures that constitutionally minimize the danger of having hazardous forks[Kogut & Metiu 2001]. In many cases not only it is ad-

vantageous to all parties to avoid a fork but also codes of conduct may prevent them from doing so.

In many cases the forked project stays closely tied to its sibling project. Projects share their source code and empowered by each other. In other cases however competing projects are actually created. But even in this case it leads to a competition that improve software quality and present choices for users and other developers.

## 3.2 Modularity and Specialization Through Forking

Forking practice creates branches of software development which in turn allow a great diversity and number of projects to come into existence. Each project specializes on solving a certain problem. As such forking allows for a sensible fragmentation of the FOSS community towards specialization.

On one extreme, every update to software is a fork which creates a new 'version' of software. In this case it is a rather incremental development path which is suitable for fixing problems and adding new features[Jørgensen 2001]. However for the sake of simplicity we do not include these incremental steps of development in our definition of forking. We are rather interested in the cases where a new development branch is initiated.

Such 'real' forks may result in a new project which stays closely tied to its parent or sibling project in some cases. Projects share some source code, but specialize on different niches. It may even be the case that new project is a sub-project of the other. In some cases however competing projects are created, and there may even be a certain tone of hostility in this competition.

(The distance of the two projects is not necessarily same from both sides. For example although the new project tries to legitimize itself and the move of forking by declaring itself as close to its sibling, the other project may be disturbed by competition and hence the distancing is not reciprocal.)

Contrary to policy of openness in using software source code, FOSS developers maintain the general principle of information hiding when writing programs. The internals of program implementation is abstracted to increase modularity and re-usability of software. Juxtaposition of these features and practice of forking for specialization makes FOSS development a very modular process[Narduzzo & Rossi 2004]. FOSS community essentially practices a distributed division of labor for increased efficiency in innovation [Kogut & Metiu 2001] and in the process a high level of modularity is introduced into structure of FOSS

community. Governance mechanisms of community maintains coherence of the system through this modularization.

### 3.2.1 Alternatives to Forking

Forking is not the only way to create specialized groups in software projects. In situations where technical and political determinants allow, a project may develop a modular structure and work practice which can accommodate specialization and flexibility within the project boundaries. Debian and GNOME are such projects which have large team size. However, in general we may say that team specialization in such large projects create similar results with forking as far as modularity is concerned. Nevertheless access to such cases is relatively limited and not addressed in our study.

In general there is a certain amount of resistance to splitting the workforce which has obvious disadvantages. However such forces of negative reinforcement, where they exist, only sets the threshold for forking decisions in FOSS and does not eliminate the potential for them.

## 3.3 Levels of Diversification

For better elaboration of the matter it is useful to identify different levels of diversification in FOSS. Although these levels may overlap in rare cases, they are mostly clear.

First level is the FOSS itself. Open source community is quite sensitive in maintaining their values and practices which are usually manifested as opposition to commercial software development practices. When interest in open source started to spread beyond a small number of enthusiasts, there was a need to clarify what is considered open source and what is not[2]. Today this clarification process and its agents are well established and definition of FOSS is enlarged to accommodate needs of private enterprises involved in FOSS activity.

Second level is the FOSS licenses themselves. The licenses range from those that use a very restricting definition of public property which disallow FOSS products to be reused under different licenses, to those that use a rather loose

[2]GPL was used since late 80s by Free Software Foundation and others. However as this was found too strict by many there was a pressuring demand for redefinition. Bruce Perens wrote the first draft of open source definition as "The Debian Free Software Guidelines" in 1997. He removed the Debian-specific references from the document to create the "Open Source Definition" which is in use by the Open Source Initiative today.

definition which permit appropriation for profit making in order to to promote collaboration with private enterprises.

The third and bottom level are the individual open source projects. Borders in FOSS are quite permeable and as noted above there are certain cases where these levels may intermingle. For example when two similar versions of the same software product is available with different licenses(one FOSS and one commercial). In general, however, these cases are rare and projects adopt the contribution policy largely determined by terms of license chosen.

# 4 Flexible Organizing and Identity in FOSS

Tendency for diversification and modularity are important features of FOSS activity. However nexus of this community, which leads enactment of flexible modes of organizing while maintaining coherence of system and focusing on innovation, is to be clarified. In other words mechanisms which prevent FOSS from being too fragmented, and maintain coherence while allowing a sensible level of fragmentation necessary for distributed innovation and specialization needs to be identified.

There are limits to flexibility through fragmentation. Schoemaker notes that "organizations tend to become opportunity coalitions when identity is too fragmented or neglected"[Schoemaker, 2003]. He also gives the following definition of identity:

> Identity refers to the process by which meaning is assigned to the organization, the norms and values in the organization, the organizational culture and the relationship between the organization and its environment (Based on Weick, 2001, and Castells 1997). Identity (literally: own character) is, therefore, a rather intangible phenomenon. An organization has an identity, but that identity develops over time.
>
> *Schoemaker, 2003*

Some theories focus on the 'communities of practice', rather than the larger organization, as the space where identity formation is practiced[Wenger 1998].According to this the nexus of ties that constitutes the community is both the space of

production and the space where meaning is assigned to the organization through a reflexive process between individuals and the immediate .

We need to answer following questions to understand what identity is and how it is constructed in FOSS as a flexible organization: (1) how are labor relations managed, (2) how is talent managed, and (3) how is identity managed [Schoemaker, 2003].

In FOSS development software license sets the contract between actors involved. Thus answer to first question mostly regards first and second levels identified in section 3.3. FOSS licenses give access to all products of the development process in general. However at a lower level, different licenses may weaken the merits of this approach by allowing use of software products created through FOSS process in proprietary software products. In such cases, usually private enterprises who are involved in development provide financing for development[3] to resume contribution. In general FOSS licenses provide a contractual setting for a balanced private-collective production mode. Together with unwritten practices of FOSS culture, they provide the dominant logic of operation[Prahalad & Bettis 1986].

Talent is managed through a distributed process at the project level, as hinted above through practice of joining and forking by developers or developer groups according to their needs and capabilities(see also [Kogut & Metiu 2001]). Such discursive practices provide the means for alignment of development activity (however certain factors, such as power struggles, may lead to deficiencies in this alignment). In some cases projects may actively, rather than passively, seek talent from other developer groups, as one would see in commercial software projects. This is especially true when projects have some sort of financing, as it happens when loose licenses are used and private enterprises are involved in development. In a rather purely FOSS discourse however the general approach to this problem can be coined as "when the released version of the software doesn't work, someone has to make a working one, one way or the other"[4]. In this respect talent management is practiced in failure situations.

Management of identity, specifically at projects level, is the key issue addressed in the rest of this study. In general we observed that at each level of diversification, identities are constructed by positioning the new organization with respect to existing references. For example General Public License(GPL) manifests its identity through contrasting and differentiation with proprietary

[3]Some examples are Apache web server, and Linux kernel itself
[4]Quote from interviewee #9 we have conducted

licenses[FSF 1991]:

> The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.
>
> *first sentence of GPL version 2, in 1991*

On the other hand there are several licenses which claim to be compatible with GPL or OSI(Open Source Initiative) Guidelines. These licenses construct their identity via similarity to GPL rather than contrasting to something.

Similar patterns of identity construction by positioning with respect to reference identities is observed as elaborated in the following sections, along with factors that effect choice of patterns. Although one may not claim this is the only mechanism of identity construction, it seems to be the key mechanism used in alignment of identity fragmentation in FOSS. Whereas (1)contrasting and putting a high distance with some references causes a split and differentiation of identities, (2)emphasizing similarity and closeness with others is utilized for maintaining coherence through the fragmentation that is necessary to resolve differences which arise in the process and achieve a sensible division of labor as required.

# 5 Summary of Data and Method of Analysis

There are several resources available for analyzing forks in open source projects. Most fresh or forked projects put forth their identity in the form of a mission statement or through statement of the key problem they claim to solve. Since open source developers usually collaborate exclusively through Internet, many of the discussions that lead to forking decisions are archived in project mailing lists, and some are published in websites or can be found in electronic forums.

We have conducted an examination of such sources for 7 cases of forking, in order to identify common discursive themes. These were rather famous cases for which a variety of records were available. We have examined more than 50 messages in 5 forums and mailing lists, 18 web pages containing recollections of people involved in or observed these forking situations in this way. During

this examination we have classified: (1)what major reasons trigger forking decisions, (2)what happens to projects after forking in the future, and (3)how forked projects position themselves to each other. A brief summary of the cases examined is presented in table 2. The number and coverage of cases selected is insufficient to draw any statistical statements about the reasons for forking. It merely provides a basis for laying out the range of contingency factors for, and positioning patterns in forking.

To support our reading of discourse we have also conducted structured e-mail interviews with 11 developers, majority of whom were involved in one of the forking cases examined in the study. We have asked them whether they approve or disapprove several relatively well-known cases of forking presented to them, and their reasons for doing so. For the same cases we also asked them which of the forked or forker they associate and why. And finally they were asked to state their objections, if any, to these forking decisions. A list of interview questions can be found in appendix A. These interviews proved most valuable in evaluating community approval/disapproval of different types of forking decisions.

In addition to analysis at project level, 4 different FOSS licenses and several records of discussions regarding their differences were examined in order to understand identity fragmentation at licenses level. Reasons for differentiation and patterns of positioning along with their similarity to projects level were laid out, however no classification of destinies of licenses were attempted. A summary of features of these licenses is presented in table 3

# 6 Results of Analysis

The three legs of the analysis and summary of developer reactions are presented in the following sections. A layout of findings are briefly presented in table 4

## 6.1 Reasons of Forking Decisions

Even when the two identities after forking are friendly towards each other, forking is a result of friction. These frictions fall into following categories:

**Power Struggles** Even though FOSS organizations are usually cited for their lack of hierarchy, there are conflicts of interest and attitudes among individuals or groups within projects. Such frictions usually result in forks, some of which may be unfriendly.

Table 2: Summary of forking cases examined: Original → Forked

**XFree86 → XOrg :** XFree86 was the original window system of GNU/Linux. At the end of 2003 some lead developers of the project made a move to license the software in a way many others have not liked. As a result a big group has left to start XOrg project and most developers left the XFree86 project. XOrg now seems to be widely accepted as the reference implementation of the window system and the original project severely weakened.

**GCC → EGCS :** FSF's GCC was the flagship project of their GNU system. When the number of developers and developer groups who use it increased, especially with speed up of Linux project, there started to appear many bug-fixes and new features from them. FSF team was both unmotivated and slow in merging these contributions to their project. So EGCS project was forked in 1993. Despite some bitterness prior to forking decision, EGCS team maintained a good relationship with FSF and the two projects were occasionally synchronized, and finally merged in 1999.

**Emacs → Lucid Emacs :** When private company Lucid started to develop an enhanced version of FSF's Emacs editor they have approached FSF to merge the two development branches. However the effort failed due to FSFs insistence on maintaining control and differences in pace. Eventually the effort was discontinued and the two products went on their way.

**386BSD → FreeBSD :** Despite the fact that 386BSD was one of first efforts for creating a version of BSD operating system for Intel 386 family of microprocessors, it was slow to develop. As a result FreeBSD was started by a few enthusiasts. Their efforts to merge was not welcome by the maintainer of original project. FreeBSD succeeded and is one of the strongest variants of BSD.

**Debian → Ubuntu :** Ubuntu was based on Debian but created with an intend to be more appealing to novice users. They have started developing separate versions of software packages which was the core of Debian, as Debian was more focused on stability of software than user friendliness. They claim an effort to merge and stay compatible with Debian. Despite good relationships Debian folks are frustrated about possibility of maintaining compatibility.

**cdrecord → dvdrecord :** Maintainer of cdrecord wanted to provide DVD functionality only in commercial version of product. Thus although both projects had DVD support as technical target they were separated.

**FSFbinutils → binutils :** FSF's binutils was slow to match the needs of Linux development effort. As a result a group started to develop a separate version. The two branches are synchronized regularly.

Table 3: Features of FOSS licenses representing the spectrum

**General Public License(GPL)** GPL appeared in 1989 and was the first license to use free software definition. This is a very strict license which requires any software that uses the whole or parts of software code released with GPL, in original or modified form, to be also released with GPL license as a whole. In other words it does not allow any permeation to non-GPL domain.

**Lesser-GPL** This slightly looser version of GPL was promoted for use in library type of software which are not standalone programs but intended to be used in writing other programs. Lesser-GPL allows using software in proprietary products provided that those parts are licensed with GPL.

**BSD-license** Used by Berkeley Systems Development group in their products, this is a very loose license which allows all kinds of reuse which acknowledges original authors.

**Mozilla Public License(MPL)** When Netscape corporation decided to open source their web browser product(and name it Mozilla) they designed MPL which takes a middle stand between Lesser-GPL and BSD licenses. MPL allows reuse of software in proprietary products however requires modification to be also released with MPL. Thus it somewhat prevents free riding by competitors. This license has later found widespread use in other projects.

**Different activity interests** Contributors of open source projects have their own agenda, such as solving a technical problem for their work or study. Thus there may be conflicts regarding what the software will accomplish that cannot be resolved. In such conflicts projects are seen to go through a friendly fork, i.e. while actively sharing most of the software code separate projects are created for specializing on separate technical targets.

**Differences in organizational mobility** Some software projects operate on a rather long-term agenda and aim to produce code that is stable. On the other hand this creates a different window of opportunity and some of the developers may be interested in producing on a higher speed and exploiting the new niches by producing feature rich products(with the cost of reduced code stability). In most such situations the new project tries to maintain a close distance to and compatibility with its sibling, although this may not always be possible as the new project will be pressured to move away by its market niche of choosing.

**Differences in licensing terms** In cases when private enterprises or academic units that has close ties with them are involved in open source projects, their experience in the market is incompatible with some public licenses in terms of how appropriation of produced software is restricted, or vice verse. Thus choice of licenses becomes problematic and causes a partitioning in contributor base. Parties in such forks seem to take a pragmatic stand and retain close distance in order to take advantage of each other's code base, as technical targets remain the same despite the undergone fork.

There may be more than one one reason for forking, although one may seem to prevail in forking decision.

In two of the cases license issues were the sole reason for forking. Strong power struggles have appeared in two cases. In three cases there were differences in both mobility and interests of two groups. In one case mobility differences was the major reason.

Strong group identity, highly established norms in a project seems to be the key issue in understanding how fragmentation is triggered under tensions of overlapping frictions:

> "They[group membership, etc.] matter to the extent that you need a sizable group of like minded developers to make a fork work. The

personality factor comes into play here too–the more cliquish the original project, the more likely a fork is, because many developers may feel excluded or dis-empowered to make any sort of meaningful change within the original project.[interviewee #7]"

### 6.1.1 Licenses Level

At the licenses level essence of differentiation is in terms of the degree to which private appropriation of software produced with these licenses are permitted. This is rather obvious as rationale of these licenses is to govern exchange of products across the boundaries of projects and FOSS World. These differences were summarized in table 3.

In two of the project fork cases where license issues appeared as the major reason, it was the sole reason mentioned for forking. In this sense license level conflicts seem to be considered more vital than others when encountered.

## 6.2 Destiny of Projects After Forking

After a fork takes place the fate of siblings may depend on each other. The cases we have examined fall into four categories: (1) either the forked or the forker die and other survives, (2) both survive by co-evolving, (3) both survive but differentiate and become incompatible, and (4)project merge after a period of co-evolution and thus the fork is un-forked. The cases where both siblings die are not considered in our study. There are many forking cases where the forker has died without leaving much trace and is long forgotten. The cases we have examined in which one of the siblings die are rather fresh, well known cases, for which there were reliable records.

Success of a software project clearly depends on many factors that are beyond the extend of our analysis. Nevertheless, destiny of forked siblings are examined with respect to initial factors of forking, in hope to enable further probing of the problem. In addition the approval level of interviewed developers is used as an estimator for developer support of projects and considered in the examination.

In only one of the cases examined one of the sibling projects was a discontinued one, and it was the original project. As noted before, many cases goes unnoticed as not much of their records is retained if the forked process is the one that died. In this well known case we have examined, a foundational system library was forked due to license problems. The original project was using a license that was considered incompatible with FOSS by many developers, and

a FOSS-compatible branch was forked. After the fork the original project is dying. All interviewed developers who cited this case was in favor of the newly forked project.

In one of the cases, projects have merged after some period of co-existence. In five others projects continues separately. In two out of these five cases projects are trying to co-operate and stay close.

Method and coverage of our study is not suitable for drawing any conclusions regarding complete determinants of success after forking, in relation to the forking itself. Such success is dependent on many other things. Forking is important mostly due to the fact that it is a critical period in shaping the identity of newly created projects and as such it effects user or developer association/disassociation with the project through good or bad publicity.

## 6.3 Positioning of Identities After Forking

After the forking process (1)a project may show a friendly attitude and manifest close proximity to the other(close positioning), or (2) it may distance itself from the other(distant positioning). This distancing is not necessarily symmetric.

The creation of GPL and FOSS movement is itself an example of symmetric distant positioning. One side can be seen from the example of GPL manifestation in the license as quoted above. At the beginning private enterprises (the 'closed source world') has reciprocated this behavior. However in recent years an increasing number of actors in private sector are trying to manifest some degree of compatibility with FOSS(including Microsoft as seen in their move to reveal parts of their source code to major customers. See also [Garud & Kumaraswamy 1993] for the case of Sun Microsystems.), and there is some degree of reciprocity from FOSS side.

At the level of projects, there is a similar prevalence of reciprocity in positioning. We did observe in some cases an asymmetry in reciprocity of positioning. This asymmetry seems to be residual and not permanent.

In three of the cases examined, there was reciprocal close positioning. In one of these cases where differences in organizational mobility was the key reason for forking, the projects have merged after a period of co-existence. In the other two there were differences in both interests and mobility of two groups.

In two cases where legal issues triggered forking there was reciprocal distant positioning. This is an indicator that issues related to higher, more established norms of identity are more important in positioning.

In the other two cases where power struggles were the dominant factor, positioning was distant, although in one of them it was not reciprocal there was a residual attitude from the forked project to establish a friendly relationship with the original project.

It is interesting that in one of the cases the parent project was disenchanted with the forking as it was indeterminate whether it was towards a specialization or direct competition. In this case the forked project is putting and increasing effort into maintaining a close relation. This positioning have no apparent technical impact for the new project, but seems to be an attempt to establish legitimacy through confirmation of the positioning. From the perspective of original project an additional factor of alienation is exclusion from the forking decision. It appears that the forked project is willing to compromise more than the original to establish a close relationship.

## 6.4 Developer Identification with Forking

When developers were asked about their approval of forking decisions, almost all of them reacted in ways stressing that they approve all forking decisions. It is as if the "right to fork" is more fundamental than anything else. This seems to directly reflect the dominant logic of FOSS licenses at a higher level[Prahalad & Bettis 1986]. In general forking is perceived as an ordinary practice that potentially leads to innovation: "I approve of all forking efforts. It is very fundamental: People can spend their time on whatever they want to"[interviewee #2], "I can't think of an example of a fork I disapproved"[interviewee #4].

Aside from this initial reaction, forking decisions that appear to be "of value to the community" are well respected while those that can be avoided and happened primarily due to power conflicts are looked down: "I approve when the fork is for technical reasons and disapprove when it's for personality reasons"[interviewee #7]. Most developers favor only necessary level of forking: "I approve of [X], because it gave extra functionality ... for the community and there was no way to get this in without forking."[interviewee #4], "We have no desire to diverge in the long term, we simply focus on short term optimizations for specific use cases important to the [X] community"[interviewee #1]

Many developers have cited rather technical reasons for the forking decisions when asked directly. But when they are asked whether personal or group conflicts play a significant role in forking decision, they confirm importance of such role. Some developers suggested that projects forked due to such social conflicts

die quickly. Some think that many/most forks could be avoided if it was not for the social struggles that prevented resolution of differences: "I think technology is often the stated issue. But the fact that the fork occurs is really because socially the differences cannot be explored within the same project"[interviewee #1], "Most of the major forks I'm aware of ... were done more for personality reasons than technical reasons"[interviewee #7]

# 7 Discussion

Flexibility in innovation and organizational change towards reaching that goal has been a subject of increasing amount of research in literature. Of particular interest was formation of inter-firm alliances in knowledge intensive sectors, such as biotechnology and software, as a strategic choice of firms in order to conduct research and development at a level which they fail to do on their own [Gulati 1995, Van de Ven & Poole 1995, de Rond & Bouchikhi 2004]. Despite differences in terminology we have chosen for the context of software development practice, forking corresponds to formation and re-formation of such strategic alliances in FOSS community.

One of several approaches which are attempted in explaining dynamics of alliance formation is the dialectics approach. In this approach organizational change is informed by collision of existing but contradictory social forces to produce a new social order. Das and Teng suggest that[Das & Teng 2000] strategic alliances are sites in which conflicting forces develop and which can be viewed as being constituted by three key pairs of competing forces: cooperation versus competition, rigidity versus flexibility, and short-term versus long-term orientation.

This approach seems to have a certain level of parallelism with the findings about reasons for forking decisions and contingency of community fragmentation on certain a priori tensions. Among our classification of tensions on which forking decisions are contingent upon, power struggles correspond to cooperation/competition tension, and mobility differences correspond to short-term/long-term tension. Within broader but weaker terms, it is possible to suggest that license issues correspond to rigidity/flexibility dimension, as certain licenses promote wider base for contribution and openness, hence for flexibility. Differences in activity interests, however, have no correspondence in dialectics approach.

One interesting question to ask is whether there is any prevalence among these different tensions in terms of identity fragmentation. Our findings, although limited to few cases, suggest, for example, that power struggles and license differences lead to permanent separation of groups and corresponding identities, whereas mobility and interest differences lead to separate but cooperating groups.

# 8 Conclusions and Limitations

Major objective of this study was to understand process dynamics of organizational fragmentation in Free/Open Source Software community. Fragmentation was examined at licenses level as well as projects level below it, with main focus on the latter. Among four major reasons determined that triggered fragmentation, power struggles and licensing were found to be leading to increased differentiation of identities and competition, rather than cooperation, between groups. Other two, differences in organizational mobility and interests, lead to formation of groups with more compatible identities, which in turn enable close cooperation between groups.

Established norms of behavior in FOSS community provides constitutional means of maintaining coherence through this process. Although developers and developer groups are free to fork whenever they want and this freedom is well respected, forking decisions are expected to be well justified, usually on the basis of interest and mobility differences. Developer groups usually try hard to resolve power struggles to avoid inefficient workforce allocations. However, insistent exercise of power to exclude a potential contribution is strongly disapproved in the community. On the other hand license issues are never taken lightly as they reflect differences at more essential levels of identity.

We have seen that power struggles and license issues may alienate developers to the extent which results in decease of alienating project/group. Whereas mobility and interest differences tend to lead to cooperative existence, and even future merging, of groups.

We are aware of problems resulting from non-random choice and insufficient number of cases examined. In this sense our study is limited in its power to suggest relative importance of contingency factors for fragmentation and reasons for survival of projects. However, it was not our intent at the first place to develop a general theory of group and identity formation. Apart from these,

comparison with more stable environments and governance mechanisms other than FOSS may be necessary to frame general categories of factors which cause formation of new groups and identities. Nevertheless our study calls for future research on dynamics of flexible organizing through in-depth, longitudinal case studies.

# References


[Burt 1995] Burt,R. (1995) Structural Holes : The Social Structure of Competition. Harvard University Press, Cambridge

[Castells 2001] Castells, M. (2001) in his Epilogue to : Himmanen, P. *The hacker ethic*, Random House, New York.

[Colazo et.al. 2005] Colazo, J. A., Fang, Y., Neufeld, D. (2005) Development Success in Open Source Software Projects: Exploring the impact of Copylefted Licenses. *Proceedings of the Eleventh Americas Conference on Information Systems.*

[Das & Teng 2000] Das, T.K., Teng, B.S., Instabilities of strategic alliances: An intrnal tensions perspective. *Organization Science.* Jan/Feb 2000. Vol. 11, Iss. 1.

[Edwards 2001] Edwards, K. (2001). Epistemic communities, situated learning and Open Source Software development. @ http://www.its.dtu.dk/ansat/ke/ec_sl_oss.pdf

[FSF 1991] General Public License, version 2. (1991) Free Software Foundation. http://www.gnu.org/copyleft/gpl.html

[Garud & Kumaraswamy 1993] Garud, R., Kumaraswamy, A. (1993) Changing competitive dynamics in network industries: an exploration of Sun Microsystems' open systems strategy. *Strategic Management Journal.* Jul 1993. Vol. 14, Iss. 5.

[Goia 1998] Goia, D.A. (1998) From individual to organizational identity. In: Whetten, D.A. and Godfrey, P.C. (eds.), *Identity in organizations. Building theory through conversations.* Sage, Thousand Oaks.

[Gulati 1995] Gulati, R. (1995) Social structure and alliance formation patterns: a longitudinal analysis. *Administrative Science Quarterly*. Vol. 40, Iss. 4.

[Himmanen 2001] Himanen, P. (2001) *The Hacker Ethic*. Random House, NY.

[Hippel & Krogh 2003] Hippel, E.v., Krogh, G.v. Open source software and the private-collective innovation model: Issues for organization science. *Organization Science*. Mar/Apr 2003. Vol. 14, Iss. 2.

[Hogg 1996] Hogg, M. (1996) Social identity, self-categorization and the small group. In:Davis, J. and Witte, E. (eds.), *Understanding group behavior. Volume 2: Small group processes and interpersonal relations.* Hillsdale, NJ: Lawrence Erlbaum, Hillsdale NJ, pp. 22754.

[Jørgensen 2001] Jørgensen, N. (2001) Putting it all in the trunk: incremental software development in the FreeBSD open source project. *Information Systems Journal*. Vol 11.

[Kogut & Metiu 2001] Kogut, B., Metiu, A. (2001) Open source software development and distributed innovation. *Oxford Review of Economic Policy*, Vol. 17. No. 2.

[Lee 2003] Lee, G.K., Cole, R.E. (2003) From a firm-based to community based model of knowledge creation: the case of Linux kernel development. *Organization Science*. Nov/Dec 2003. Vol 14, Iss. 6.

[Lehmann 2004] Lehmann, F. (2004) FLOSS developers as a social formation. *First Monday*.

[Narduzzo & Rossi 2004] Narduzzo, A., Rozzi, A. (2004) The Role of Modularity in Free/Open Source Software Development. In: Koch, S. (ed.) *Free/Open Source Software Development*, Hershey, PA.

[Prahalad & Bettis 1986] Prahalad, C.K., Bettis, R.A. (1986) The dominant-logic: a new linkage between diversity and performance.

*Strategic Management Journal*. Nov/Dec 1986. Vol 7, No 6.

[de Rond & Bouchikhi 2004] de Rond, M., Bouchikhi, H. (2004) On the dialectics of strategic alliances. *Organization Science*. Jan/Feb 2004. Vol. 15, Iss. 1.

[Raymond 1999] Raymond, E.S. (1999). A brief history of hackerdom. *The Cathedral and the Bazaar: musings on Linux and Open Source by an accidental revolutionary*. Beijing: O'Reilly. @ http://catb.org/ esr/writings/cathedral-bazaar/hacker-history/

[Schoemaker, 2003] Schoemaker, M. (2003) Identity in Flexible Organizations: Experiences in Dutch Organizations. *Creativity and Innovation Management* Vol. 12, Iss. 4.

[Schweik & Semenov 2003] Schweik, C.M. and A. Semenov, (2003). The institutional design of Open Source programming: implications for addressing complex public policy and management problems. *First Monday* Vol.8, Iss.1.

[Van de Ven & Poole 1995] Van de Ven, A.H., Poole, M.S. (1995) Explaining development and change in organizations. *Academy of Management Review*. Vol. 20, Iss. 3.

[Weber 2003] Weber, S. (2003) *The Success of Open Source*. Harvard University Press, Cambridge, MA.

[Weick 2001] Weick, K.E. (2001) *Making Sense of the Organization*. Blackwell, Oxford.

[Wenger 1998] Wenger E. (1998) Community of practice: learning as a social system. *Systems Thinker*, Jun/Jul 1998. Vol 9, Iss. 5.

[West 2002] West, J. (2002) How open is open enough: melding proprietary and open source platform strategies. *Research Policy*, special issue on "Open source software development"

# A Interview questions

There are many cases in which a group of developers fork a new project and start developing on their own agenda. Some examples you may be familiar are Debian vs. Ubuntu, BSD vs. FreeBSD/OpenBSD/NetBSD variants, emacs vs xemacs, XFree86 vs Xorg, gcc vs. egcc, glibc vs libc, etc. In our research study we attempt to investigate reasons of forking and its relation to group identity of open source software projects. Please reflect on the following questions:

1. Can you describe the main reasons, in your terms, for one forking decision you know of.
2. You may approve some forking decisions, and disapprove some others. Please give one example of each and describe your reasons for approval/disapproval.
3. Do you think forking leads to good, innovative software projects. Or rather can you give examples when it is and it is not.
4. Do you think forking occurs mostly due to problems in the existing project(i.e. slowness of managerial decisions, technical limitations of code base, etc.) that prevents the team from exploiting new opportunities that arise? If not what do you think the main reasons that trigger forking?
5. Do you think groups(group membership, trusted peers, friendship networks, etc.) is more important in forking decisions or it is rather due to technical, practical reasons.
6. Do you think some social factors such as dominance, fame, personal conflicts leads or accelerates forking in projects.
7. Have you been involved in any forking? If yes please describe briefly.

Table 4: A summary of findings for forking cases examined

| Case | Important reasons for forking | Destiny of projects | Positioning(original to forked, forked to original) |
|---|---|---|---|
| XFree86 → XOrg | License issues | original is dying forked project is successful and has more support | distant, distant |
| GCC → EGCS | Mobility differences, mild power struggles | merged after several years | close(after initial hesitation), close |
| Emacs → Lucid Emacs | Different interests, power struggles, mobility differences | goes separate | distant,close |
| 386BSD → FreeBSD | Power struggles | original died, forked project survives | distant, close |
| Debian → Ubuntu | interest and mobility differences | both alive, goes separately | cautiously close, close |
| cdrecord → dvdrecord | License issues | goes separate | distant, distant |
| FSF binutils → Linux binutils | mobility and interest differences | both alive | close, close |