First Monday
PEER-REVIEWED JOURNAL ON THE INTERNET

FLOSS developers as a social formation

by Frauke Lehmann

Abstract
Developers of Free/Libre and Open Source Software (FLOSS) are often referred to as a community or as a scene. But so far this seems mostly just a rough expression. This paper takes a closer look at FLOSS developers and their projects to find out how they work, what holds them together and how they interact. Community and social movement seem not to apply as descriptors. Looking at norms, values, structures, and conflict resolution, a hacker subculture appears which is compartmentalised into differently organised projects. By testing empirical findings against various theoretical approaches, ideas for further research are identified.

Contents

Introduction
Boundaries
Social processes
Processes of social cohesion
Summary of empirical findings
Theoretical explanations

Introduction

This paper is intended as a first contribution to the description of developers of Free/Libre and Open Source Software (FLOSS) as a social whole. Most studies have investigated internal phenomena of the field or of individual projects, while the social whole has been largely overlooked or was treated only superficially. Although many studies repeat words such as "community" or "movement," there is usually no follow–up discussion of whether and to what extent these terms actually correspond to the FLOSS phenomenon. This article aims at understanding FLOSS with a more holistic approach. This implies not only an analysis of the field as a whole, but also of the interaction between various internal phenomena. The characteristics determining and shaping a social formation which will be discussed in this paper are processes of boundary–drawing, internal structures, and the cohesion of the field.

The term *social formation* refers to the types of social connections between developers. This general term was chosen to avoid expressions such as *group* or *community* which have clear and distinct meanings in sociological theory. Using these terms would lead to misunderstandings and inaccuracies.

The subject *FLOSS developers* denotes, in the context of this paper, not only actual coders, but also people involved in beta testing, documentation, translation, infrastructure (*e.g.* servers, Web sites, mailing lists), and public relations. To include people who only use software seems inappropriate. They may belong to the FLOSS world, but do not contribute in any form to actual software development.

This study is based on data from interviews and conversations with developers from a variety of projects, and with other persons close to the field. Also, vital insights were provided additionally by the analysis of project Web sites and other online resources, a monitoring of selected mailing lists, observations on various fairs and conferences and, of course, examining the existing literature.

As this text is meant as a starting point for further study and discussion, its findings largely are hypothetical. For reliable findings, a much broader and intensive study is necessary — which would require the cooperation of a number of scholars.

Boundaries

Legal boundaries

FLOSS is clearly set apart from other software by the specific design of property rights. Property is a social relationship among people which regulates the disposal of physical or socially constructed things [1]. The claim to a given object is not the decisive element, but the recognition of that claim by others, *i.e.*, by society. Property rights have the form of a bundle of rights which can be divided into four types of rights: rights of use, rights of modification, rights of transferal, and the competences/competence [2]. While the licences agreements in proprietary software (*e.g.* Microsoft, 1995) grant the buyer only a small set of rights of use and exactly one right of transferal (re–sale under specific circumstances), copyright–owners of FLOSS grant nearly full rights of use and modification. How the competences/comptence is dealt with varies from licence to licence. Under the GNU General Public License (GPL) [3] the copyright–owner retains the full competences/competence through the copyleft effect, while Berkeley Software Distribution (BSD)–type licences [4] only prescribe that the original authors of the software must be named. In addition to these two licences (including the GNU Lesser

General Public License), there are a number of other FLOSS licences. But according to Freshmeat, these two are the most widespread (Open Source Technology Group, 2004).

Beyond FLOSS and proprietary software, there are other forms of licencing, *e.g.* public domain, freeware, and shareware. These also differ from FLOSS by their property rights design [5].

Cultural boundaries

In addition to these external legal boundaries, there are subjective, cultural boundaries embodied in the distinction of insiders and outsiders and in the collective self–image. The feeling of belonging is constructed through a collective identity. There is a canon of shared values with regard to FLOSS development which takes the form of quality standards and which is different from the principles underlying development of proprietary software. The goal is to write software and not — like in software companies — make a profit. Whether a product is ready for use at a specified time does not play an essential role. Rather, a new version should be released whenever it is fit for use. Software must not only work somehow, but should also be written well [6]. There is a shared understanding that code should be elegant, in the sense of "perfection [in design] is achieved not when there is nothing more to add, but rather when there is nothing more to take away" (Raymond, 2000a). Quick and dirty solutions can be acceptable, too, but only if something is needed now, and such solutions are always considered temporary. Other developers are not seen as competitors, but as helpful co–developers. Therefore, feedback, including bug reports, is expressly desired — if one’s work is improved by others, no drop in personal status follows (unlike in many corporate environments). On the contrary, this environment is understood as an opportunity for learning. Another important aspect is the necessity for self–reliance: if something does not work as you think it should, you will have to fix it yourself and cannot wait for somebody else to do it. Finally, the formal attributes and outward characteristics are quite unimportant; only coding ability and willingness count — as is shown by remarks such as "our best C coder was a 13–year old" [7]. And of course there is a consensus that software should be free.

The goal is to write software and not — like in software companies — make a profit.

To be able to become an insider, a certain level of knowledge is required. This is well epitomised by the t–shirt slogan "There are 10 types of people in the world: Those who understand binary and those who don’t." Here, a division of the world in two groups is made, with binary code as a symbol for computer knowledge as the distinguishing characteristic. Like the other points enumerated above, this is a closely work– or skill–related self–definition. Generally, FLOSS developers, in their self–image, tend to emphasise technical knowledge and a certain can–do–spirit to differentiate themselves from others. The fact that such knowledge and skill differentials are printed on t–shirts or coffee mugs underlines their role as elements of a lifestyle.

Finally, there are a number of clichés on the lifestyle of FLOSS developers. And while these are shared with the wider nerd culture outside FLOSS, they do strongly influence a self–image (and are quite often readily lived up to).

In addition to shared norms and values, the collective identity is underpinned by shared traditions. Common roots and role models are found in the early days of computing. The 1960s hackers at MIT with their spirit of exploration, or the early Unix users who developed a form of close and free cooperation are among them (*cf.* Levy, 2001; Salus, 1995). Individuals and stories also play a role. Richard Stallmann's dispute with Xerox about source codes [8], or the debate between Linus Torvalds and Andrew Tannenbaum (DiBona, *et al.*, 1999b) are examples.

Today, all these values and (his)stories have been canonised within the FLOSS field. Many activists reflect upon what they do, why they do it and what behaviour is dominant. Eric Raymond's essay "The Cathedral and the Bazaar" (Raymond, 2000a) is especially important in this respect. He came to the conclusion that the decisive innovation through Linux was not on the technical, but on the social level. He studied the form of cooperation in the Linux project and found a de–centralised bazaar model, which he set off against the traditional corporate approach of hierarchical, small–circle development which he dubbed the cathedral style. Raymond's analysis was widely read and generally accepted and now forms, together with a number of writings on the history of hacking and computers, a foundation for the self–image of the FLOSS field. Other such texts include autobiographies (*e.g.* Torvalds and Diamond, 2001) or historic reports (*e.g.* Levy, 2001). Together they contribute to and form the collective memory. Finally, gossip plays an important role, which indicates that there is an internal public sphere [9] based on magazines or Web sites (*e.g.* Slashdot [10]).

The common identity is strengthened further by the identification of common enemies. These enemies include anything and anyone which is perceived as prohibiting access, including copyrights, patents, and secret source codes, but also mechanisms that encourage dependence. In addition, Bill Gates and Microsoft are identified as common enemies, along with the SCO Group [11]. It is interesting to note that in both cases, former insiders have become enemies. Bill Gates broke away in 1976, when he wrote his "Open Letter to Hobbyists" [12]. In these cases, the problems are not just despised practices, but feelings of betrayal by insiders.

Social processes

Geographical factors

Actual development work is usually localised at home or in the work place, which means at a location physically apart from other developers. Most social activity, therefore, has to take place online, in virtual space. These interactions have a narrower bandwidth than face–to–face interaction, as Internet communication is generally restricted to text. The advantage, though, is that virtual communication works across time zones and vast distances. All of this communication is stored in archives, creating a memory of events. However, a close study revealed that there are indeed physical personal contacts as well: some of the projects organise meetings, while others meet at conferences and fairs. Obviously, these spatial characteristics have an impact on the possible forms of social relations.

Places of production

Places of production are generally projects consisting of volunteers [13]. The thematic range of these projects is very wide, reaching from full operating systems and their elements, to applications, small device drivers, and computer games. Most projects are started by individuals who are interested in solving a specific problem or want to develop a particular piece of software. Projects grow when others get interested and start to contribute — sometimes from the social environment of the originator, but more often when a project has been made public, people start using the software and provide feedback. Sometimes companies originate projects by making their software available under a FLOSS license, *e.g.* Mozilla [14]. The size of projects also varies widely; there are many one–person efforts and some with over 1,000 contributors.

The inner structure of project organisations can also take a plethora of forms. Key aspects here are the political structure and the degree of organisation, which are both closely related to each other: there are projects (such as Debian [15]) which has a clearly laid out organogram. The Debian project leader is elected in a formalised process. Such a structure can be described as a form of representative democracy.

Other projects, such as KDE [16] (a desktop environment) or Linux [17] have a consolidated structure, but a far smaller degree of differentiation and codification. Their organisation follows from the problems that need to solved. Holders of important positions are not elected and therefore have no formal legitimacy. Their influence is based on reputation, and they might quickly lose it, when other developers withdraw their approval [18]. As political systems, these projects can be described as Big Men societies or chiefdoms.

A Big Man is the speaker of an egalitarian group. His personality qualifies him for the job; his energy, valour, capability, rhetorical and organisational talent guarantees his position. The position is not hereditary, and even a sitting Big Man must permanently reassure his position, and he has no means to enforce his standing. This is much unlike the chief, who holds a permanent, clearly defined and hereditary position.

Chiefdoms are hierarchically ordered. Myths usually legitimise the special position of the chief. But he can be unseated by the elders. A chief can, unlike a Big Man, exact tribute from his subjects [19].

The organisational structures of FLOSS projects are not designed at the drawing board; they are the result of happenstance, conventions, and negotiation.

A further category of projects, *e.g.* Worldforge [20] (a multi–user role–playing game) or Eisfair [21] (an Internet server), only exhibit a rudimentary structure. Eisfair, for example, has one project leader, but the coders can fairly autonomously pursue their own tasks. The Eisfair project leader (also the founder) is more a small–scale headman than a chief [22]. Worldforge did have a project leader, but since he has failed to return from a holiday, everything is done informally. The project could be described as an acephalous [23] horde, in which there may be some person with more influence than others, but where everything is agreed in direct communication [24]. It is telling that Worldforge developers communicate via IRC rather than via mailing lists.

Projects also differ in whom they consider members, and the degree of membership within a given project can vary as well. Apart from officially assigned functions, such as being a member of the coreteam or a maintainer, writing access to Source Code Management Systems (SCM) is a distinguishing feature, as it allows contributors to work autonomously. Projects handle the granting of such rights very differently. Debian demands the successful completion of a series of tests to prove technical ability but also to show adherence to the Debian Social Contract [25] — a kind of constitutional charter of the project which has a lot to say about freedom of software. Only when these tests have been passed satisfactorily — which can take a month or more than a year — is one assigned the official status of a Debian Developer. This form of admission — which is bordering on a formal initiation process — seems to be rather unique (*cf.* van Gennep, 1999). Official developers in KDE or NetBSD [26] will propose to grant SCM writing access to someone who has submitted some good contributions (patches). There are many developers without their own SCM writing access who send their code to a developer with access who will check contributions after a quality check. Other projects (*e.g.* Mozilla), rely on competent users for beta testing who look for errors and correct them — a task often done by newbies as their first contribution. But not only coders can be project members: the extreme case is Worldforge where sympathetic bystanders who neither contribute to nor use the software can be members, solely on the grounds that they like the idea behind the project and regularly appear on its IRC channel.

It is widely assumed that the allocation and distribution of positions is based on reputation (*cf.* Raymond, 2000b). Such reputation, though, is not only acquired meritocratically by writing good code; the idea of elders (where the project founder is assigned in some fashion the role of leader) is also quite important. The organisational structures of FLOSS projects are not designed at the drawing board; they are the result of happenstance, conventions ("that's what is done in FLOSS projects"), and negotiation. But there is a clearly observable link between organisational structure on the one hand and the type, complexity and size of the project, *i.e.* there seems to be a form–follows–function principle.

Networks in the field

In addition to the internal structures within projects, the relation between participants in projects and to other important players in the field are relevant. Some projects foster social relations between each other, the KDE developers, for example, invited their Gnome [27] (a desktop environment) and Debian counterparts to their social event at the 2004 LinuxTag [28]. Being a distribution project, Debian has a frequent exchange with other developers to discuss a variety of topics, but especially licences. And there are, of course, developers who are part of more than one project.

It seems doubtful whether the entire field is linked by a coherent network structure. Among the bigger projects many connections can be observed. But it is unclear whether smaller projects are equally well connected.

As mentioned earlier, projects are places of production — the politics of FLOSS seem to be kept separate to a high degree. While there is lively discussion on legal and political issues within Debian, this takes place on a mailing list (debianlegal@lists.debian.org) which is separate from the lists for technical issues. Most other projects have no such special lists and these issues are only rarely debated. Many of the bigger projects now have separate organisations (foundations) which deal with legal and financial issues.

The wider politics of FLOSS seem to be tracked by specialised organisations, such as the Free Software Foundation (FSF) [29] or the Open Source Initiative (OSI) [30]. There are a wide range of societies which go beyond FLOSS which have an interest in a variety of FLOSS issues, such as the Electronic Frontier Foundation (EFF, focusing on free speech online) [31], Foundation for a Free Information Infrastructure (FFII, mainly opposing software patents) [32] or the German Chaos Computer Club (CCC, specialising in data protection issues) [33].

Processes of social cohesion

One key aspect of a social formation is its ability to persist. Individual members, who take part on a voluntary basis, need to be bound to the projects sufficiently tightly, so they will not quit in the face of minor problems or conflicts. This bond rests to some extent on the will to contribute (individual motivation), which will vary in degree from member to member, but also on a shared feeling of belonging. The project structures, too, must have a degree of mid–term stability to ensure a sufficiently stable work environment. In this context, the mechanisms for conflict resolution play a major part.

Motivation

Two types of motivation need to be distinguished: the motivation for joining, and the motivation to stay on, which to some extent needs to be supplied by the project and its structures (systemic motivation).

The motivation for joining is twofold: motivation for taking part in FLOSS development at all, and motivation to join a particular project. The reasons of individuals vary widely: to fight for the spread and growth of FLOSS; to give something back for the software you use (reciprocity); solving a specific problem; having fun (the FLOSS playground); improving your own programming skills; and, a service to others [34].

The systemic motivation relates to the reasons for continuing to participate. In addition to the above reasons, a set of motives emanating from work done can be observed: acquiring reputation; the personal bond and a feeling of responsibility to other members of the project or the software product itself; and, work ethic considerations ("getting the job done").

All these motivations are in evidence, and there seems to be no hierarchy between them and no root motivation [35].

Conflicts

Types of conflicts

Four types of conflicts can be identified in the FLOSS field: ideological, technical, personal and cultural conflicts. Of course, most actually existing conflicts are a mixture of these types.

Ideological conflicts are usually about differing ideas about and approaches to the meaning of freedom of software. The Gnome project, for example, was founded as a result of a conflict with KDE who used the then proprietary Qt library from Trolltech. It could be argued that the term Open Source also emerged as the result of such a conflict. A group of key people in the field — such as Eric Raymond — introduced the term to give the phenomenon a new name. They wanted to distance themselves from the FSF and its allegedly anti–commercial attitude. They hoped to popularise Open Source Software, as they now called Free Software, among wider groups of users, who, they felt, might have been alienated by the FSF’s value–conscious approach [36]. Technical conflicts are, evidently, about different views on the best solution for a given technical problem. Bearing in mind the strong role of technological perfection and code elegance, they have a huge conflict potential. Like the famous debate about macro– vs. micro–kernels, these conflicts are disputes about the best way to achieve a specified target.

Personal conflicts usually result from the behaviour of individuals who commit a breach of (social) norms. All discussions about how people should behave towards each other fall into this category. Also in this category was a case when a developer unjustifiedly claimed authorship.

The fourth and final type of conflict is best described as cultural. These are less about breaches of norms, and have more to do with the non–acceptance of norms. One such conflict has developed over the past few years following the wider spread of FLOSS; today, there are many users with only a very limited knowledge of computers. But it is not only their level of skills (which is lower), their attitude to computers is also quite different. They take a consumer's view of FLOSS and have very different expectations *vis-à-vis* developers. They wait (or ask impatiently) for a specific feature instead of contributing to its development. They ask questions clearly answered in the documentation — both demands which are very similar to those made by purchasers of commercial products. This results in many conflicts between developers and users who are not part of the FLOSS (sub) culture, and to mutual alienation.

Types of conflict resolutions

Given the spatial status of the places of production and the voluntary character of all contributions, the options for conflict resolution strategies are rather limited. How conflicts are played out also depends on the political structure. Many of the usual methods and processes of conflict resolution in social formations are rendered futile by the general lack of physical face–to–face situations (except at the rather rare physical meetings). The chances to sanction misbehaviour are also small, as there is no formal way to enforce the FLOSS or project compact — which is unlike, for example, working contracts in commercial software development environments. So to force a person to follow a specific path of behaviour is very difficult.

Formal procedures provide one way of handling conflict resolution. These require a codfied organisational structure. Debian developers, for example, have the option of elections if no consensus can be found on important issues.

In projects led by recognised authorities, someone can act as a sort of judge, *e.g.* the project leader or a maintainer. This individual can either put their foot down or choose to ignore a particular person and her contributions. They can also elect to spontaneously call for a vote. The chance for success will vary with project structure. Anyone with this sort of power will have to keep in mind the legitimacy of her position. A purely charismatic leader (*e.g.* project founder) can easily put her own standing at risk.

In projects without central authorities, however, conflicts have to be settled by negotiation. This can range from a rational, facts–based debate to so–called flame wars (personal attacks launched in e–mail). The assignation of shame, *i.e.* of negative reputation, is an instrument in such conflicts. Negotiaton processes can end in a consensus, by one side giving up their point, or by forking the project, *i.e.* splitting it in two. While judges do not appear in such projects, someone may act as a arbitrator between the conflicting parties.

Independently of these sorts of battles, there is always the option of a split, or fork. The terms of FLOSS licences make forks possible; both halves will be able to continue their work with only a loss in numbers and skills, but without having to rewrite any code. This forking option is staunchly defended [37]. Temporary forks are also a way out of major conflicts, giving both parties the chance to implement their ideas, and allowing for the project to be reunited under a more successful course of action. Evidently, this is a solution for severe technical conflicts.

Forms of sanctions

The toughest sanction, which is only employed in cases of severe social or cultural conflicts, is exclusion: the person concerned will lose his SCM rights, will be removed from project mailing list and be otherwise boycotted. This option is only used in special cases and very rare. A more common choice is shaming and blaming, leading to a negative reputation for the sanctioned person. Given the key role of reputation, this is a severe sanction. Finally, somebody can simply be ignored, which is a limited form of exclusion.

Generally, in all four types of conflict, it is easier to make people stop doing something than force them to do something else. The FLOSS field simply lacks enforcement (or enticement) options.

Stabilisation

In addition to mechanisms for conflict resolution there are other factors which foster the stability of and adherence to projects.

A negative determinant is the lack of resources, especially development time. Therefore, forking a project is usually seen as a very undesirable step as it is perceived to be a waste of resources. Given the spirit of wanting to get things done and the problem–solving approach, there is a strong incentive for cooperation which can help to overcome this scarcity (*cf.* Elwert, 1980). This interpretation differs from Raymond’s (Raymond, 2000b) assumption about FLOSS as an economy of abundance.

Most projects also develop a distinct collective identity which is often expressed in symbols, such as soft toys, mascots, and t–shirts, that define the project boundary for insiders as well as outsiders. Project rituals, such as Debian’s New Maintainer Process or KDE’s regular Social Event at the annual LinuxTag in Karlsruhe, are important. Friendly competition between projects also helps to sustain the project identity, as is well observable in the case of KDE and Gnome.

Potential conflicts over basic legal issues are externalised, with supreme jurisdiction left to a given state's legal system. Another important point is the enculturation (a slow absorption of norms and cultural practices) of newcomers. Many projects offer introductory texts about how the project works technically and culturally. In addition there are some general texts on online behaviour, *e.g.* standards of etiquette in IRC channels and mailing lists. Newcomers can also count on the help of experienced developers, and when they violate some project norm, they are usually told how they should behave in future.

Finally, nearly all proceedings are public, which imposes a certain degree of social pressure on all involved. This applies not only at project level: there are more universal fora which serve the same purpose across projects.

Summary of empirical findings

FLOSS is first of all defined by a specific structure of property rights, which gives — unlike proprietary licences — users far–reaching rights. FLOSS development is located in individual projects focusing on one thematic field. Projects are voluntary associations. Their structure varies widely and is the result of happenstance, convention and negotiation. The project structure seems to be shaped by the size and the task, *i.e.* a form–follows–function principle can be discerned. The form of interconnection between projects is less clear. For larger projects, it can be said that they are in regular contact — not only extending to cooperative work, but also to organising social, group–formatory events. Networking can also develop around individual programmers who are members of various projects and form a kind of bridge. To what extent they actually fill that role varies.

Social exchange usually takes place online — and during fairly rare physical meetings. Nevertheless, there is a collective self–image, based on shared values, most of which concern development work itself, but which also include political and social values such as free access to information and open cooperation. This self–image also forms the boundary line, separating developers and FLOSS as a whole from the rest of the world. This boundary is supported by a set of symbols, some of which are in in–group code and cannot be understood by outsiders (especially those lacking knowledge of computers and Unix). A second type of boundary are common enemies, mostly traditional software companies. The formation of a collective FLOSS identity has by now progressed further, including shared traditions, canonised values, and an endogenous written history — all of which are the subject of lively discussion among FLOSS developers. Finally, there seems to be a common public sphere which serves not only to discuss individual projects, but also other projects and wider issues. Individual developers are bound to their projects not only by the voluntary act of joining (which can be based on a wide variety of motivations), but also by a project identity which is formed in friendly competition with other projects and symbolised in mascots and logos, t–shirts and rituals.

As a final point, the importance of actual work itself for the association of developers should be emphasised again. Not only is the structure of projects shaped by this work, but it also heavily influences collective identity.

Theoretical explanations

I will now introduce six theoretical concepts which represent different approaches to the study of social formations and will apply them to my empirical observations. These concepts are community, social group, organisation, social movement, subculture, and social world. Each of these concepts can contribute to a better understanding of FLOSS as a social formation. I do not assume that one model will perfectly describe the FLOSS environment, but aim to point to specific characteristics of the field and to gain new insights for further study.

The theory of social groups and organisations describes the inner structures of social formations. Social groups are marked by the direct personal nature of relationships and informal structures in groups, based on interaction. Each group member is usually assigned a role and can gain a certain status. The stability of groups is based on shared values, norms, and goals as well as on a feeling of belonging together. A sense of separateness and a hostile environment further such a feeling (*cf.* Schimank, 2001). As groups are defined by their personal and informal character, their structures, as well as their norms and values, are never entirely fixed, but subject to a permanent process of negotiation (Neidhardt, 1983). Organisations are instruments to efficiently achieve a set of specified aims. Their structure is only planned after the aims and the best way to achieve them have been defined. The result is a formal structure, in which humans are perceived only as fulfilling functions within their defined remit (Schimank, 2001). The differentiation between function and person is crucial (Weber, 1980).

FLOSS projects are about creating software, not about social change.

The shape of individual FLOSS projects usually lies somewhere between the informal group and the formalised organisation. As voluntary associations, all projects can be initially defined as groups. Some projects, especially the bigger ones, have developed strongly organisational characteristics. An extreme example is Debian which is marked by a high degree of codification and clearly defined procedures. As a result of indirect communication and the lack of face–to–face exchange, though, some key phenomena of social groups cannot be easily observed. On the other hand the degree of formalisation created by the specific medium of communication, might wrongly imply organisational characteristics. So to clearly distinguish between groups and organisations in this virtual environment, the focus should probably be on the questions of roles (as understood in group theory) and the person–function (which is so central to organisation theory) distinction. This will allow us to understand the importance of personalities and issues such as leadership.

The central feature of the theory of social worlds is the idea of a core activity around which a social formation develops (*cf.* Strauss, 1978; Strauss, 1982). This approach considers actions, rather than structures. The cohesion and delimitation of a social world depends on the sense of authenticity which members assign to the execution of the core activity. This authenticity serves to set the social world off against others, and also to legitimise the development of hierarchies within the social world. The core activity of software development is shaping the collective identity as well as the concrete structures of social relations among developers. The issue of authenticity appears in the form of notions of code quality. The closer a developer comes to this ideal, the higher her reputation will be — and the more influence she will be able to wield. The problem with the theory of social worlds is that it is too universal — it can be applied to nearly any phenomenon and is not so helpful in studying a phenomenon in all its aspects.

The idea of community (in the sense of *Gemeinschaft* as described by Ferdinand Tönnies) emphasises the non–rational sense of belonging among human beings, which is not bound to any specific purpose. This bond between individuals is based on sympathy, adaption or familiarisation and memory, depending on the exact nature of the relationship (Tönnies, 1991). While Tönnies focusses on organic groups (such as families), a wider interpretation of such bonds would include constructed relations of "natural" belonging, usually based on a common memory or history (*cf.* Anderson, 1983, Elwert, 1989). Some of the basic assumptions of community theory — close non–rational ties in a homogeneous, spatially delimited environment — seem at first glance not to fit FLOSS developers very well. On the other hand, though, a number of typical community formation processes are in progress, *e.g.* the writing of an endogenous history and the canonisation of values. So there is a case for studying the phenomenon as a community, but the theory would have to be adapted to virtual relations without over–stretching the concept.

A (sub)culture serves the integration of a social group, in that it regulates collective life by defining categories for perception and assessment (*cf.* Sack, 1971; Yinger, 1960). Subcultures form around a set of values which partially differ from mainstream society; behaviour not regulated may well remain within the limits of mainstream society’s norms. Subcultural values are actualised in a system of norms, a particular lifestyle, rituals, and discourses. These cultural processes serve to stabilise and reproduce the subculture. There is a large overlap between this theoretical model and the actual cultural dimension of the FLOSS phenomenon: a set of values and norms which deviate to an extent from mainstream society, and specific practices can be observed. Also, mechanisms for reproducing the culture and maintaining its boundaries are in place. But the centrality of actual software production is not quite grasped by this theoretical approach. FLOSS is not only and not primarily about being a subculture. When FLOSS is studied as a subculture, one important aspect should be kept in mind: as a global phenomenon, it deviates not from one mainstream society, but from many, and in possibly different ways. This is a question which seems not yet very well covered by existing theories of subcultures.

Social movements are marked by their focus on starting (or stopping) a defined development in society. They are amorphous constructs consisting of individuals and groupings which are united by their motivation to start/halt a change. Voluntarism and strong self–motivation are key characteristics of social movements (*cf.* Rucht and Neidhardt, 2001). On the counts of voluntarism and motivation, FLOSS exhibits many characteristics of a social movement. The relevance of political and legal issues (free access) also seems to point in this direction. But FLOSS projects are about creating software, not about social change. Political activities are mainly the task of other organisations, which may have close links to FLOSS projects and even the same members, but which are set apart from development projects precisely by their political activist character. Therefore FLOSS as a whole is not a social movement. To (nearly all) FLOSS developers, writing software is the focal point, not the political aspects. The structure of projects is shaped by this, not by the achievement of some social change. There is however, a set of organisations (such as the FSF or the OSI) which are more movement–like in character. But they belong to a social movement concerned with issues like free speech, free access, and data protection — not with writing good code.

About the author

Frauke Lehmann will graduate in late 2004 with a Master’s Degree from Free University Berlin, Germany, in Social Anthropology (Ethnologie), Sociology, and Economics. Other research interests include development studies, political ritual, political movements, and African politics. She plans a PhD on the logic and culture of the FLOSS knowledge markets.

Notes

1. *Cf.* Elwert, 1999, p. 1143; Hann, 1998, p. 2.

2. Richter and Furubotn, 1996, p. 82; Richter and Furubotn do not discuss the idea of a competences/competence. The term originates in public law and refers to the competence (or right) to create new or re–assign existing competences (rights); Streinz, 2001, p. 49 [No. 121].

3. http://www.gnu.org/licenses/.

4. http://www.xfree86.org/3.3.6/COPYRIGHT2.html#6.

5. I will not discuss licences of these types. A good overview can be found in Free Software Foundation (FSF), 2002.

6. Due to the accessibility of source code, slovenly coding is easily detected and attributed to an individual author.

7. A member of *Worldfordge,* interview 12 July 2003, Karlsruhe/Germany.

8. *Cf.* Williams, 2002, p. 1; Grassmuck, 2002, p. 222.

9. At the annual LinuxTag conference in Karlsruhe, there are usually a few issues which dominate discussions and which preoccupy most of those present, such as changes in the Debian Social Contract.

10. http://slashdot.org.

11. The Santa Cruz Operation Group (SCO Group) brought a lawsuit against IBM, claiming that IBM employees had illegally integrated proprietary AT&T Unix code (which SCO Group owns) into IBM's Linux projects, hence leading to a demand for royalties (Kuri, 2004).

12. Available on many sites on the Web, including http://www.blinkenlights.com/classiccmp/gateswhine.html.

13. There are a handful of companies which are involved in the development of FLOSS, but these will not be considered in this paper.

14. http://www.mozilla.org.

15. http://www.debian.org.

16. http://www.kde.org.

17. http://www.kernel.org/.

18. On KDE, *cf.* Brandt, p. 27.

19. Kohl, 1993, p. 58.

20. http://worldforge.org.

21. http://eisfair.org.

22. *Cf.* Kohl, 1993, p. 61.

23. Acephalous means "without head." In an anthropological context, the term refers to societies without a central authority.

24. Kohl, 1993, p. 53.

25. http://www.debian.org/social_contract.html.

26. http://www.netbsd.org.

27. http://www.gnome.org.

28. Linuxtag in Karlsruhe is said to be Europe's biggest Linux convention/fair.

29. http://www.fsf.org.

30. http://opensource.org.

31. http://eff.org.

32. http://www.ffii.org.

33. http://www.ccc.de.

34. The reason giving by Mattias Ettrich for starting KDE; see Moody, 2001, p. 358.

35. *Cf.* Raymond, 2000b, p. 11; Stallman, 1999, p. 55; and, Torvalds, 2001.

36. DiBona, *et al.*, 1999a, p. 3.

37. *Cf.* contributions to the Linux kernel mailing list: for example, see
http://marc.theaimsgroup.com/?l=linuxkernel&m=108757316101769&w=2,
http://marc.theaimsgroup.com/?l=linuxkernel&m=108757316229168&w=2.

References

Benedict Anderson, 1983. *Imagined communities: Reflections on the origin and spread of nationalism.* London: Verso.

Andreas Brandt, no date. "Fallstudie horizontales elektronisches Arbeitsnetz/Open Source–Projekt." Unpublished MS.

Chris DiBona, Sam Ockman and Mark Stone, Mark, 1999a. "Introduction," In: Chris DiBona, Sam Ockman and Mark Stone (editors). *Open source: Voices from the open source revolution.* Sebastopol, Calif.: O'Reilly & Associates, pp. 1–17.

Chris DiBona, Sam Ockman and Mark Stone, Mark, 1999b. "The Tannenbaum–Torvalds debate," In: Chris DiBona, Sam Ockman and Mark Stone (editors). *Open source: Voices from the open source revolution.* Sebastopol, Calif.: O'Reilly & Associates, pp. 221–251.

Georg Elwert, 1999. "Eigentum," In: Hans Dieter Betz, Don S. Browning, Bernd Janowski and Eberhard Jüngel (editors). *Religion in Geschichte und Gegenwart.* Tübingen: Mohr–Siebeck, p. 1143.

Georg Elwert, 1989. "Nationalismus und Ethnizität: Über die Bildung von Wir–Gruppen," *Kölner Zeitschrift für Soziologie und Sozialpsychologie,* volume 41, number 3, pp. 440–464.

Georg Elwert, 1980. "Die Elemente der traditionellen Solidarität: Eine Fallstudie in Westafrika," *Kölner Zeitschrift für Soziologie und Sozialpsychologie,* volume 32, number 4, pp. 681–704.

Free Software Foundation, 2002, "Categories of free and non–free software," at http://www.gnu.org/philosophy/categories.html, accessed 13 August 2004.

Arnold van Gennep, 1999. *Übergangsriten.* Frankfurt/Main: Campus.

Volker Grassmuck, 2002. *Freie software: Zwischen privat– und gemeineigentum.* Bonn: Bundeszentrale für politische Bildung.

C.M. Hann, 1998. "Introduction: The embeddedness of property," In: C.M. Hann (editor). *Property relations: Renewing the anthropological tradition.* Cambridge: Cambridge University Press, pp. 1–47.

Karl–Heinz Kohl, 1993.*Ethnologie — die Wissenschaft vom kulturellen Fremden: Eine Einführung.* München: C.H. Beck Verlag.

Jürgen Kuri, 2004, "SCO vs. Linux: Die unendliche Geschichte. c't aktuell vom 10.02.2004," at http://www.heise.de/ct/aktuell/meldung/44492, accessed 1 August 2004.

Steven Levy, 2001. *Hackers: Heros of the computer revolution.* London: Penguin.

Microsoft, 1995. "Word 1997: Endnutzer–Lizenzvertrag für Microsoft–Software," Included in software package.

Glyn Moody, 2001. *Die software rebellen: Die erfolgsstory von Linus Torvalds und Linux.* Landsberg/Lech: Verlag Moderne Industrie.

Friedhelm Neidhardt, 1983. "Themen und Thesen zur Gruppensoziologie," In: Rene König, Friedhelm Neidhardt and Rainer M. Lepsius (editors). *Gruppensoziologie: Perspektiven und Materialien.* Opladen: Westdeutscher Verlag, pp. 12–34.

Open Source Technology Group, 2004. "Freshmeat.net: Browse project tree — licence: OSI approved," at http://freshmeat.net/browse/14/, accessed 18 August 2004.

Eric Raymond, 2000a. "The Cathedral and the Bazaar." version 3.0.2, at http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/, accessed 1 August 2004.

Eric Raymond,2000b. "Homesteading the Noosphere," version 3.0, at http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading, accessed 1 August 2004.

Rudolf Richter and Eirik G. Furubotn, 1996. *Neue institutionenökonomik.* Tübingen: Mohr.

Dieter Rucht and Friedhelm Neidhardt, 2001. "Soziale Bewegungen und kollektive Aktionen," In: Hans Joas (editor). *Lehrbuch der Soziologie.* Frankfurt/Main: Campus, pp. 533–556.

Fritz Sack, 1971. "Die Idee der Subkultur: Eine Berührung zwischen Anthropologie und Soziologie," *Kölner Zeitschrift für Soziologie und Sozialpsychologie,* volume 23, number 2, pp. 261–282.

Peter H. Salus, 1995. *A quarter century of Unix.* Reading, Mass.: Addison-Wesley.

Uwe Schimank, 2001. "Gruppen und Organisationen," In: Hans Joas (editor). *Lehrbuch der Soziologie.* Frankfurt/Main: Campus, pp. 199–222.

Richard Stallman, 1999. "The GNU operating system and the free software movement," In: Chris DiBona, Sam Ockman and Mark Stone (editors). *Open source: Voices from the open source revolution.* Sebastopol, Calif.: O'Reilly & Associates, pp. 53–70.

Anselm Strauss, 1982. "Social worlds and legitimation processes," *Studies in Symbolic Interaction,* volume 4, pp. 171–190.

Anselm Strauss, 1978. "A social world perspective," *Studies in Symbolic Interaction,* volume 1, pp. 119–128.

Rudolf Streinz, 2001. *Europarecht.* Fifth edition. Heidelberg: C.F. Müller.

Ferdinand Tönnies, 1991. *Gemeinschaft und gesellschaft.* Third edition. Darmstadt: Wissenschaftliche Buchgesellschaft.

Linus Torvalds, 2001. "Was geht in Hackern vor? Oder: Das Linussche Gesetz," In: Pekka Himanen (editor). *Die Hacker–Ethik und der Geist des Informations–Zeitalters.* München: Riemann Verlag, pp. 13–18.

Linus Torvalds and David Diamond, 2001. *Just For Fun: Wie ein Freak die Computerwelt revolutionierte.* München: Carl Hanser Verlag.

Max Weber, 1980. *Wirtschaft und gesellschaft: Grundriss der verstehenden Soziologie.* Fifth edition. Tübingen: J.C.B. Mohr.

Sam Williams, 2002. *Free as in freedom: Richard Stallman's crusade for free software.* Sebastopol, Calif.: O'Reilly & Associates.

J. Milton Yinger, 1960. "Contraculture and subculture," *American Sociological Review,* volume 25, number 5, pp. 625–635.

Editorial history

Paper received 22 September 2004; accepted 19 October 2004.

Contents Index

FLOSS developers as a social formation by Frauke Lehmann
First Monday, volume 9, number 11 (November 2004),
URL: http://firstmonday.org/issues/issue9_11/lehmann/index.html