Computer Supported Cooperative Work (2005) 14:323–368
DOI 10.1007/s10606-005-9000-1

Socialization in an Open Source Software Community: A Socio-Technical Analysis

NICOLAS DUCHENEAUT
Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA, 94304, USA (E-mail: nicolas@parc.com)

Abstract. Open Source Software (OSS) development is often characterized as a fundamentally new way to develop software. Past analyses and discussions, however, have treated OSS projects and their organization mostly as a static phenomenon. Consequently, we do not know how these communities of software developers are sustained and reproduced over time through the progressive integration of new members. To shed light on this issue I report on my analyses of socialization in a particular OSS community. In particular, I document the relationships OSS newcomers develop over time with both the social and material aspects of a project. To do so, I combine two mutually informing activities: ethnography and the use of software specially designed to visualize and explore the interacting networks of human and material resources incorporated in the email and code databases of OSS. Socialization in this community is analyzed from two perspectives: as an individual learning process and as a political process. From these analyses it appears that successful participants progressively construct identities as software craftsmen, and that this process is punctuated by specific rites of passage. Successful participants also understand the political nature of software development and progressively enroll a network of human and material allies to support their efforts. I conclude by discussing how these results could inform the design of software to support socialization in OSS projects, as well as practical implications for the future of these projects.

Key words: actor-network, learning, Open Source, socialization, software development

1. Introduction

Software development, due to its highly collaborative nature, has been a long-standing topic of inquiry within the field of computer supported cooperative work (e.g. Button and Sharrock, 1996; Potts and Catledge, 1996). In particular, the unique challenges of geographically distributed development projects have attracted significant attention (e.g. Grinter et al., 1999; Herbsleb et al., 2000). Indeed, these projects are rich in opportunities to study the effects of computer-mediated interactions on collaboration and organization.

Recent developments have highlighted the need for further research in this area. The emergence and success of the Open Source Software (OSS) movement, often characterized as a fundamentally new way to develop software, poses a serious challenge to more traditional software engineering approaches (Mockus et al., 2000). In many OSS projects developers work in

geographically distributed locations, rarely or never meet face-to-face, and coordinate their activities almost exclusively over the Internet. They successfully collaborate using simple, already existing text-based communication media such as electronic mail (Yamauchi et al., 2000) as well as revision tracking systems (e.g. CVS – Concurrent Versioning System) to store the software code they produce (Fogel, 1999). Evangelists argue that the ''bazaar'' model of OSS (Raymond and Young, 2001), with its egalitarian network of developers free of hierarchical organization and centralized control, has virtues that the commercial software world will never be able to emulate.

In great part because of its newly acquired visibility, its revolutionary claims, and its unusual characteristics listed above, OSS has generated a significant amount of questioning and research among social scientists and software engineers alike. Both have examined the economic, organizational and institutional aspects of the OSS movement. Of great interest to many, for instance, was the question of individual motivation: for its most ardent proponents the ''free'' (meaning voluntary and often uncompensated) contribution of OSS participants represents a new alternative to traditional economic systems (Raymond and Young, 2001); others have argued that, while OSS differs from more traditional forms of organization, participation can still be explained using traditional economic mechanisms (e.g. contributors can be rewarded through means other than money, such as reputation – see Lerner and Tirole, 2002; Von Hippel, 2002). Others have been more concerned with social and organizational aspects, looking for example at issues such as coordination (Fielding, 1999; Yamauchi et al., 2000) or members' participation (Zhang and Storck, 2001; Maas, 2004) to get at the ''essence of distributed work'' (Moon and Sproull, 2000) contained within OSS. These studies typically seek to quantify aspects such as message traffic, code ownership, productivity, defect density, etc. either through case studies (Mockus et al., 2000; Robles-Martinez et al., 2003; Gonzalez-Barahona et al., 2004) or the analysis of similar data over a large number of projects (Ghosh and Prakash, 2000; Krishnamurthy, 2002; Madey et al., 2002). Finally, another segment of OSS research has investigated its institutional dimensions, for instance by replacing OSS into a larger ecology of community-centered practices (Tuomi, 2001), describing its political economy (Weber, 2000), or comparing it to other institutions such as science (Bezroukov, 1999; Kelty, 2001).

1.1. FROM STATIC TO DYNAMIC ACCOUNTS OF OSS

These past analyses and discussions, however, tend to have treated the organization of OSS projects mostly as a static phenomenon. Many studies have produced ''snapshot'' accounts using aggregate statistics to summarize, for instance, the position of a particular participant inside a project based on

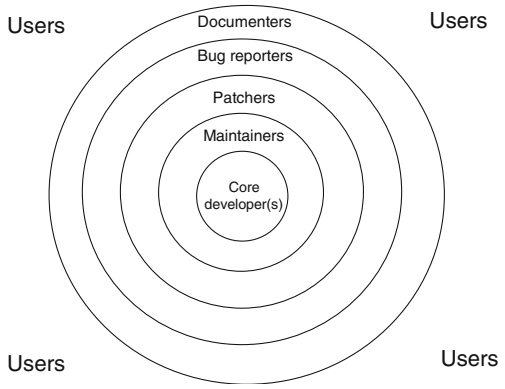

Figure 1. Current picture of OSS community organization.

the frequency of his message postings (Moon and Sproull, 2000; Zhang and Storck, 2001) or the distribution of CVS commits and downloads (German and Mockus, 2003). Such studies reveal a pattern of community organization that we could characterize as a series of concentric circles, each occupied by people playing a particular role in the development process (Moon and Sproull, 2000; Maas, 2004). In the center are core developers (frequently a single individual, like Linus Torvalds in the Linux project). Surrounding the core we find the maintainers, often responsible for one or more sub-components (modules) of a project. Around these we find patchers (who fix bugs), bug reporters, documenters... and finally the users of the software (see Figure 1). Here ''users'' is a slightly ambiguous term, as it usually conjures up visions of low-skilled or unskilled end-users of computer applications. In the Open Source world this is not necessarily the case since many Open Source productions are, in fact, developed by developers for other developers (e.g. computer languages: Feller and Fitzgerald, 2002) and the ''users'' can therefore be highly skilled. Hence the peripheral region of Figure 1 is composed of a nebulous arrangement of both skilled and unskilled individuals, in proportions that vary depending on the project studied.

This layered organization seems to be either imposed by developers to manage the design, implementation, and maintenance of OSS projects when they reach a certain size (see for instance the case of Apache in Fielding (1999)) or simply to emerge by itself over a project's lifetime, as it grows (Gonzalez-Barahona et al., 2004; Maas, 2004). Indeed several empirical studies have found that, in a large majority of cases, a small core group is often responsible for a large proportion of the work accomplished and a very large group of peripheral participants is responsible for the remainder (Ghosh and Prakash, 2000; Maas, 2004). This is reminiscent of earlier research on the organization of ''chief programmer teams'' (Kraft, 1977),

which described how software development projects tend to structure themselves as a role hierarchy – with a few number of important contributors at the top, and a much larger number of less important contributors surrounding them. In this respect at least, OSS projects may not be much different from more traditional software engineering teams.

However, while this characterization of OSS teams as role hierarchies seems to be generally well supported by empirical data, it neglects an important question. Indeed as I mentioned earlier, it is essentially static and does not explain how this software production system is sustained and reproduced over time. In particular, there is little research on how individuals external to the community are progressively socialized (or not) into a project (for a notable exception see Von Krogh et al., 2003). Instead, OSS research on newcomers has largely focused on one issue: motivation (that is, why participants choose to contribute despite potential costs to their entry – Lerner and Tirole, 2002; Von Hippel, 2002). The dynamic strategies and processes by which new people join an existing community of OSS developers and eventually contribute code are still under-investigated. Interestingly, while this issue of socialization has been explored in the context of traditional software engineering groups (e.g. Sim and Holt, 1998), there is much less data and analyses available from the Open Source world.

And yet, the socialization of newcomers is particularly crucial for the Open Source movement. First, the success of a project appears to be highly correlated with its size: projects that fail to attract and, more importantly, to retain new contributors rarely get beyond a name and a few lines of code stored on Sourceforge[1] (Ghosh and Prakash, 2000; Capiluppi et al., 2003). Second, as successful OSS projects mature, their technology grows more complex and only a few people who have been actively involved in their development fully understand their architecture (Von Krogh et al., 2003). If these key members were to leave (a probable event – pressures from their ''real'' job often forces OSS contributors to cut down or stop their activities), the project would eventually falter. This makes socializing new members an essential ingredient in the long-term survival of OSS projects – an important goal, considering the central role played by OSS productions in running public infrastructures such as the Internet.

Moreover, the question of socialization in OSS projects is also interesting from a theoretical standpoint. Indeed, the development of the individual Open Source participant appears to be both novel and paradoxical. Some accounts suggest that contributors move from peripheral participation toward mastery and full participation in a way that is strikingly similar to classical apprenticeship learning (Edwards, 2001; Von Krogh et al., 2003; Shaikh and Cornford, 2004). And yet, OSS being a fully electronic community, this apprenticeship lacks most of the concrete elements that are said to be essential to its success: the one-on-one connection with a master, the

hands-on observation and practice under observation, the physical nearness (Lave and Wenger, 1991). Current accounts of participation in projects also tend to gloss over the political maneuvering that often takes place around the integration of new members (Divitini et al., 2003). Instead, they describe an over-simplified and supposedly meritocratic process broadly inspired from the ideal of the scientific world (Kelty, 2001), where the best ideas and people get naturally selected based on their talent and scrupulous peer-review.

Therefore, it seems important to explain more precisely how and why OSS newcomers successfully move towards full participation and, perhaps ultimately, to the roles of core developer or maintainer. To achieve this, we need to adopt a more dynamic perspective: socialization is a process unfolding over time, not a simple act of matching a contributor to a role. We therefore need to look at the *trajectories* of participants from joining to contributing, not simply at the structure of a project (Inkeles, 1969).

1.2. THE HYBRID NATURE OF OPEN SOURCE DEVELOPMENT

It is worth mentioning at this point that a participant's progressive familiarization with the "tools of the trade," as well as the concrete productions that result from using these tools, are both central elements of traditional apprenticeship and socialization in communities of practice. Indeed "things" or "artifacts" are central in the construction of a practitioner's identity – practitioners of technology learn by doing, not through abstract means (Gordon, 1993). More generally, "things stabilize our sense of who we are; they give a permanent shape to our views of ourselves" (Csikszentmihalyi, 1993). In online communities, control over artifacts is a central source of power and influence (see the discussion of "wizards" in Cherny (1999)), and textual resources are manipulated to project a participant's "virtual" identity (e.g. Turkle, 1997). Contributing to the creation of an artifact is, therefore, as much a concrete activity of production as a social act of identity building.

This brings me to another dimension missing from the current OSS literature: despite its importance, few studies have paid close attention to the hybrid nature of OSS projects. Indeed email, code and databases constitute not simply the end products of OSS development efforts (as most of the literature would lead us to believe), but also material means that OSS participants interact with and through. As Mahendran (2002) described in one of the rare attempts to address this issue,

> "The multiple components [of an Open Source project] that at first seem to be hard material are in essence text [...]. This distributed network of people and things is constructed through the materialization of language. [...] There is a hybridism of dialogue and code, where the dialogue is directly embedded in the code" (Mahendran, 2002, pp. 13–14).

The importance of such an ecological view, articulating the relationships between people and "things" instead of focusing on one side of the equation alone, has been repeatedly emphasized by many scholars in the field of science and technology studies (e.g. Latour, 1987b; Star, 1995; Kling et al., 2003). However, despite its appropriateness for Open Source research (Divitini et al., 2003), few researchers so far have adopted such an analytical framework (exceptions can be found in the work of Tuomi (2001), Mahendran (2002), Osterlie (2004)). Instead, OSS research has generally focused either on the social side of the phenomenon (e.g. observing social networks across OSS projects, as in Madey et al., 2002) or the material side (e.g. inferring the structure of a project from CVS data, as in Gonzalez-Barahona et al. (2004)), independently of each other.

Instead, I think it is crucial to take the hybrid nature of OSS more seriously, and to *simultaneously* investigate the complex networks of people and material resources constitutive of OSS projects. This has consequences for the analysis of socialization in these projects: instead of a linear progression from role to role based purely on the quality of code contributions (a popular view, heavily criticized in Kelty (2001), Divitini et al. (2003)), I argue we need to analyze OSS socialization as the active creation and maintenance of strong links between individual participants and the socio-technical network of a project. As we will see in Section 3, this allows for a deeper and more nuanced analysis than what has been proposed so far: recognizing the hybrid nature of OSS projects helps reveal the complex political maneuvering taking place during the integration of new members (Divitini et al., 2003) and shows that an OSS participant's skills need to go far beyond purely technical knowledge.

1.3. A SOCIO-TECHNICAL INVESTIGATION OF SOCIALIZATION IN OSS PROJECTS

In summary, I have argued above that we need to examine how OSS communities are reproduced, transformed and extended *over time* through the progressive integration of new members, as these members interact with *both the social and material* components of a project. This particular framing of the issue of socialization in OSS is meant to bridge the two main research gaps identified earlier, namely: (1) that research on participation in OSS projects tends to be based on static accounts; (2) that this research also tends to separate the social and material side of a participant's activities.

Consistent with this view, I present in the remainder of this paper the results of a case study documenting the socialization of new members in a specific OSS project based on an analysis of their dynamic, hybrid interactions with and through computer code and electronic messages. To move away from limiting models of OSS organization such as the concentric circles of Figure 1, I have chosen instead to frame the problem in terms closer to

those of actor-network theory (ANT, see Latour, 1987b, 1996). To quote from Latour,

> "Instead of thinking in terms of surfaces [...] one is asked to think in terms of nodes that have as many dimensions as they have connections. [...] ANT claims that modern societies cannot be described without recognizing them as having a fibrous, thread-like, wiry, stringy, ropy, capillary character that is never captured by the notions of levels, layers, territories [...] Literally there is nothing but networks, there is nothing between them."

Open Source projects, with their complex network of interactions between people and people as well as between people and things, have such a "fibrous" quality. I will show below that, by reframing OSS projects as hybrid networks in lieu of a simple layered organization, it is possible to better understand the complex processes through which newcomers eventually become recognized code contributors, essential to the survival of their community.

2. Research methods: "computer-aided ethnography"

2.1. COMBINING ETHNOGRAPHY AND VISUALIZATION SOFTWARE

In the previous section, I highlighted several theoretical gaps currently limiting our understanding of socialization in OSS projects. To bridge this gap I proposed to conceptualize socialization as individual trajectories in a dynamic, hybrid network. But all of this would be of limited use if we did not have a way of using and exploring this new framework empirically. I will now describe the research methods used to address this issue.

As I mentioned earlier, OSS is an extreme case of geographically distributed software development. The members of a project coordinate their activities through simple tools over the Internet. The most central tools so far have been email (for communication between all the interested parties – from users to developers) and Concurrent Versions System (CVS – a database to store the code produced by the developers). Because of this particular infrastructure, the nature of the data generated by OSS projects is the source of two practical problems. First, and especially for those interested in using qualitative methods, it is extremely easy to fall prey to data overload. Indeed the number of messages exchanged in Open Source mailing lists is in the order of hundreds, frequently thousands of messages per week. It is quite difficult to keep track of such a constant influx of messages, let alone analyze it – and this is only one of the data sources available about each project. Second, OSS research material can be quite opaque. Despite their centrality in the Open Source development process, tools such as CVS databases, for example, produce few immediately analyzable outputs to the untrained eye.

It is possible to obtain logs of activity, but the researcher then has to face the first problem: sifting through page after page of coded text somehow representing the participants' contribution to a project. Researchers therefore have to face a difficult tension between abandoning the richness of the material available in favor of a more high-level, quantitative approach or focusing in depth on rich episodes, without any guarantee of their place in the big picture.

These problems are not specific to Open Source research, and illustrate the more general challenge of conducting ethnographic observations of online environments (e.g. Lyman and Wakeford, 1999; Hine, 2000; Rutter and Smith, 2002). In trying to find ways to circumvent these obstacles, I have adopted an approach comprising two mutually informing activities: ethnography and the construction and use of software to visualize and explore the hybrid, dynamic networks of humans and non-humans incorporated in the email, code and databases of OSS. This software facilitates the observations that are essential to ethnography, is in itself a form of ethnographic inscription, and extends a burgeoning tradition of "software-as-theory" (Dumit and Sack, 2000; Sack, 2000b, 2001) that addresses many of the difficulties an ethnographer has to face in studies of online environments.

The software developed as part of this approach is, therefore, intended to serve as a new mode for interrogating online relations. In adopting this approach, I want to promote the view that software technology can be purposefully built as a theoretical artifact incorporating, extending, and reflecting on ethnographic insights (Sack and Dumit, 1999; Dumit and Sack, 2000; Sack, 2000b, 2001). In this I follow the aforementioned authors' suggestion that computational tools can function both as analytic and generative devices. Analytic devices are used to test hypotheses, while generative devices are designed to be used in exploratory data analysis and as a means to formulate hypotheses. In other words, "such devices are generative insofar as they are evocative objects meant to engender discussion" (Sack, 2000b, p. 2). They are precisely what would be needed to facilitate the ethnographic analysis of online environments since they would offer a set of possible observational foci, drawing the attention of the observer to common and potentially interesting patterns of activity that could then be analyzed in depth and qualitatively.

Inspired by this research tradition, my analyses of OSS can therefore be seen as a form of "computer-aided ethnography" (a characterization I borrow from Teil and Latour (1995)'s notion of "computer-aided sociology"). My understanding of participation in OSS is based on a constant movement between the design of software to visualize the practices of interest (namely, participation in a project as manifested through code and email contributions) and qualitative observations using this software.

2.2. THE OSS PROJECT BROWSER

2.2.1. *Requirements*

Several attributes are required for a piece of software to assist in the ethnographic analysis of Open Source participation. From a theoretical standpoint, it should be generative by embodying a strong theory about the phenomenon being observed. This ''software-as-theory'' component will be used as an ''evocative object meant to engender discussion'' (Sack, 2000b) – or more precisely in my case, it will provide the observational foci needed for the ethnographic analysis. In the case of OSS, two theoretical attributes are required:

(1) The software must make the hybrid nature of a project visible by showing the connections not only between people, but also between people and material artifacts.
(2) The software must offer a dynamic perspective on activities and allow observations over time.

The above is meant to reflect what I consider to be a crucial change of perspective articulated earlier in this paper: namely, that we should move from the currently static, ''concentric circles'' model of OSS development to a dynamic, hybrid and network-like picture of OSS instead (Latour, 1987b, 1996). My first contribution with this software, therefore, will be to visualize how these hybrid networks evolve over time. This complements the efforts of scholars in science and technology studies, who have been trying to incorporate their theoretical ideas into software that supports actor-network analysis and ANT (e.g. Callon et al., 1986; Teil and Latour, 1995).

Furthermore and from a more practical standpoint, the software should facilitate the ethnographic observation of a project. Consequently:

(3) It must offer both aggregate views (to avoid data overload and facilitate the selection of interesting episodes of activity) and at the same time preserve access to the raw, untouched research material for qualitative analysis.
(4) Since my interest here is in the socialization of newcomers, there must be ways to track a participant's trajectory easily.

Requirement 3 above is not only meant to provide direct access to the research material – an essential component of qualitative research – but also to consciously avoid a very common pitfall in the design and use of computer visualizations that Sack rightfully emphasizes (Sack, 2000b). Indeed, scientific images sometimes become dangerously ''untethered from the data used to produce them'' (Sack, 2000b, p. 8). When this happens, these images become too easy to manipulate and misinterpret. We should therefore strive to keep high-level representations of a phenomenon like OSS ''tethered'' to the supporting data.

The fourth requirement will allow the ethnographer to engage in two forms of observations: a holistic appreciation of the social milieu through the aggregate views (observation at a distance), but also a form of "virtual shadowing" or focal follow (Bernard, 1998) to track a particular participant, or even an artifact (Marcus, 1995), over time. This is the second novel software contribution I am proposing here: currently, tracking the trajectories of individuals over time in online environments is, at best, difficult to achieve. Hopefully, the software I describe below mitigates this problem and can be used fruitfully to conduct online ethnographies.

2.2.2. *Architecture Of The OSS Browser*

The Conversation Map system (Sack, 2001), as it stands today, provides a strong foundation one could build upon and extend to satisfy the above requirements. The Conversation Map is much more than a simple social network browser: it includes social networks, semantic networks and, most importantly, sociolinguistic networks (the articulation that connect the social and semantic networks together). It is, therefore, already a hybrid representation. For the purpose of studying Open Source projects however, a different kind of hybridism needs to be highlighted; namely, one that encompasses social networks, software networks and, most importantly, *socio-technical* networks (the networks that connect social and software networks together). We also need ways to track the evolution of these networks over time.

It is worth mentioning here that, while the usefulness of network-based analysis has not been lost on the OSS research community, no software has been developed yet to simultaneously track social *and* technical networks. For instance, the work of Gonzalez-Barahona et al. (2004) uses only technical networks (namely, patterns of CVS contributions) to infer the organization of the Apache project. At the other end of the spectrum, Madey et al. (2002) consider only social relations (expressed through common membership in different projects) to analyze the community structure of the OSS world at a more ecological level. My goal in this paper is to merge these perspectives through an analysis of the OSS phenomenon in its totality: that is, by considering the hybrid, socio-technical networks emblematic of OSS projects.

Therefore and following the requirements outlined in Section 2.2.1., I produced over the course of several development iterations a standalone extension of the Conversation Map tailored to the analysis of OSS projects. The architecture of this new Open Source Project Browser is outlined below. The modules inherited from the Conversation Map will not be described in detail – the interested reader is referred instead to Sack's extensive description of the technical aspects of his system (Sack, 2000a).

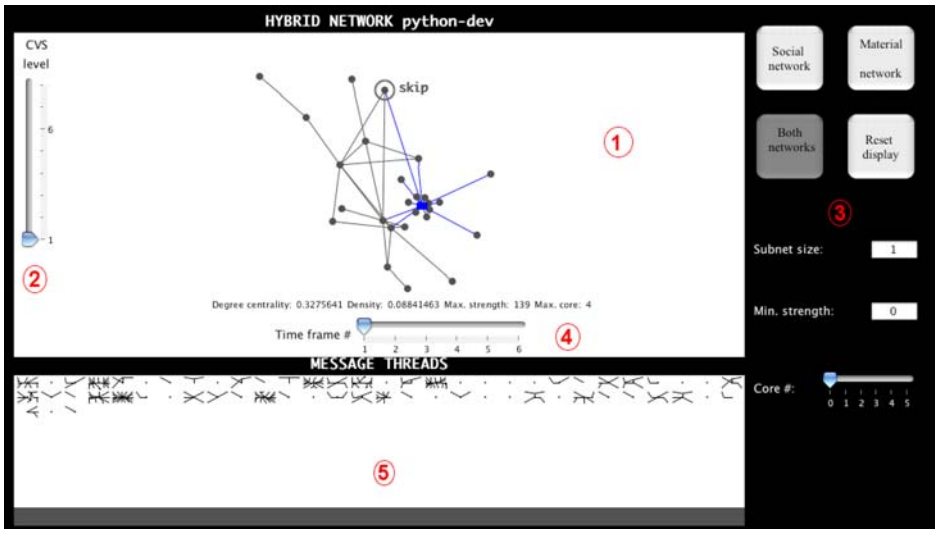

Figure 2. The Open Source Project Browser.

Before being displayed in the user interface, the data obtained from an Open Source project has to be processed. The ''backend'' of the system, developed in Perl (Wall et al., 2000), first retrieves the email messages and CVS logs for a given project. This data is then organized in a format suitable for display and analysis in the browser's user interface. This ''front end'' of the system is implemented in Java and runs as an applet so that it can be viewed from almost any Web browser. The user interface is composed of three parts: two visualization panels and a series of control buttons (Figure 2).

The topmost pane (1) is a graphical representation of (borrowing Latour's (1987b) terminology) the hybrid network for a particular OSS project. Black dots are individual participants in the project. A black line connects participants if they have both responded to each other over email. The more they reciprocated, the shorter the line (as in Sack, 2001). Another important part of this representation consists of the artifacts for this project, namely software code. Code is represented as blue rectangles. When an individual contributes code to the project, he is connected to the corresponding artifact with a blue line. The more he contributed to this particular artifact, the shorter the line. As such, this panel offers a new visualization for understanding the collaborative authoring practices of software development teams that employ email and CVS, such as those of OSS. It shares similarities with previous works on scientific citation analysis (Garfield, 1979; Callon et al., 1986), in which texts are linked via people and people are linked via text. The ''text'' of an Open Source project is represented here by the software code it produces.

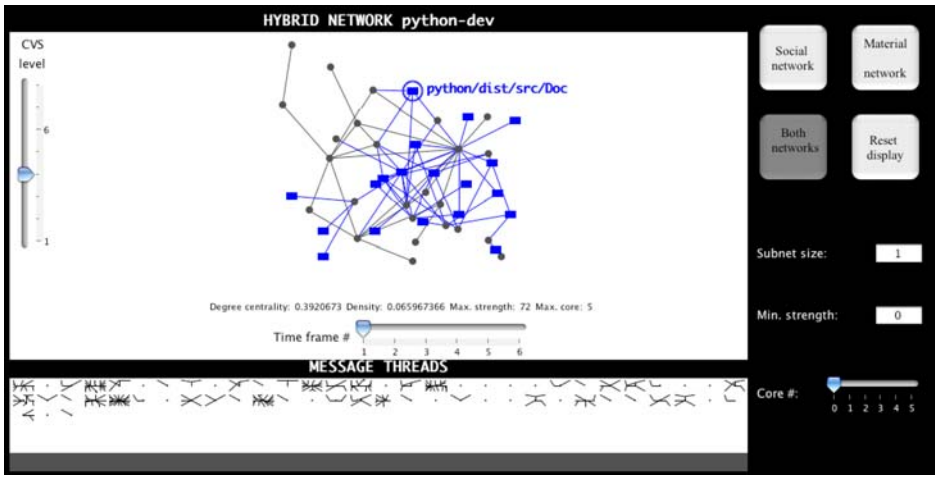

Figure 3. A view of the hybrid network showing artifacts at the 4th level of the CVS tree.

Artifacts and the participants' relationships to them can be analyzed at different levels of granularity. Figure 2 showed a high-level overview by agglomerating all the artifacts as one blue rectangle representing the entire project. It is possible, however, to analyze the participants' contributions in more detail. Indeed, CVS data is stored as a tree of code modules and we can decompose the first level, agglomerated view into its sub-components using the slider on the left (2). In Figure 3, for instance, the user has drilled down the CVS records to display artifacts at the 4th level in the database hierarchy. This way we see that participants do not contribute to the same modules of the project: for instance, some are actively working on the Mac source tree, others are writing documentation, etc.

In order to understand the dynamics of this hybrid network, it is also possible to see how it evolved over time using the timeframe slider (4) at the bottom. The numbers represent months in the life of the project – Figures 2 and 3 contain data from January 2002. If we move the slider to 3 we can see the network changing as we get into March. Since this is animated the impression one gets is hard to convey in writing, but the nodes in the network progressively move to new positions as the strength of their connections to each other changes with time. New participants and artifacts fade in slowly from the background so that one can see where they enter the network.

As I mentioned earlier, an important part of computer-aided ethnography is to preserve the raw, original research data for further qualitative analysis (Sack, 2000b). Again, there are two main sources of such material in OSS: email messages, and CVS logs. The purpose of the second panel (labeled (5) in Figure 2) is to provide an abstraction of the former, and to serve as an

entry point to access the original messages. This panel is a graphical representation of all of the messages that have been exchanged over a given period of time in the mailing list or newsgroup for a particular OSS project. It extends Sack's (2001) original module with a slightly improved representation of time. The messages are still organized into threads, i.e., groups of messages that are responses, responses to responses, etc. to some given initial message. Each thread is represented as a "glyph", in which the original message appears as a dot in the center and responses branch out from it in a "spider web" layout (see Figure 4 for an expanded view of a simple thread with three messages).

Threads are organized chronologically, from upper-left to lower-right – the oldest conversations can be found in the upper left-hand corner of pane (5). This pane is also tied to the time frame slider: the glyphs are progressively updated to reflect the addition of new messages as time move forwards. This way it is possible to distinguish between "static" and "dynamic" threads: the glyphs for the latter become progressively darker and darker as new nodes and edges are added, whereas the former do not evolve. The denser the glyph, the more messages have been exchanged, and this way episodes of intense conversational activity can be quickly isolated.

In the expanded view of a thread, each node represents a message sent by one of the participants to this conversation. Double clicking on any of these nodes gives immediate access to the original email message that was sent to the project (this part of the system is a direct re-implementation of the work of Sack, 2000a).

Selecting a particular thread in pane (5) highlights its participants in the hybrid network. Since most of the data in the system is cross-referenced the reverse is also possible: clicking on a participant in the hybrid network highlights the threads that he contributed to by surrounding them with a black oval in pane (5) (Figure 4). This way the analyst can get an idea of the kind of discussions this participant engaged in, as well as the extent of his participation.

Selecting a participant has two additional effects. First, it isolates the subnetwork that he is a part of, fading the rest of the network to the background. This way the relative position of the participant as well as his most direct connections are more easily visible. Second, it allows the user to access another source of raw research data: CVS logs. When a participant is selected, clicking on the artifacts that he is connected to will open a separate window containing the logs of his activity on each artifact (Figure 4). These text entries, written by the participants when they commit changes to the database, frequently provide a rationale for their technical decisions as well as a summary of the discussions or events that led to changes in the software.

Figure 4 below illustrates with a concrete and simple example how the system can be used. In this example, the selected participant P1 (the red dot)

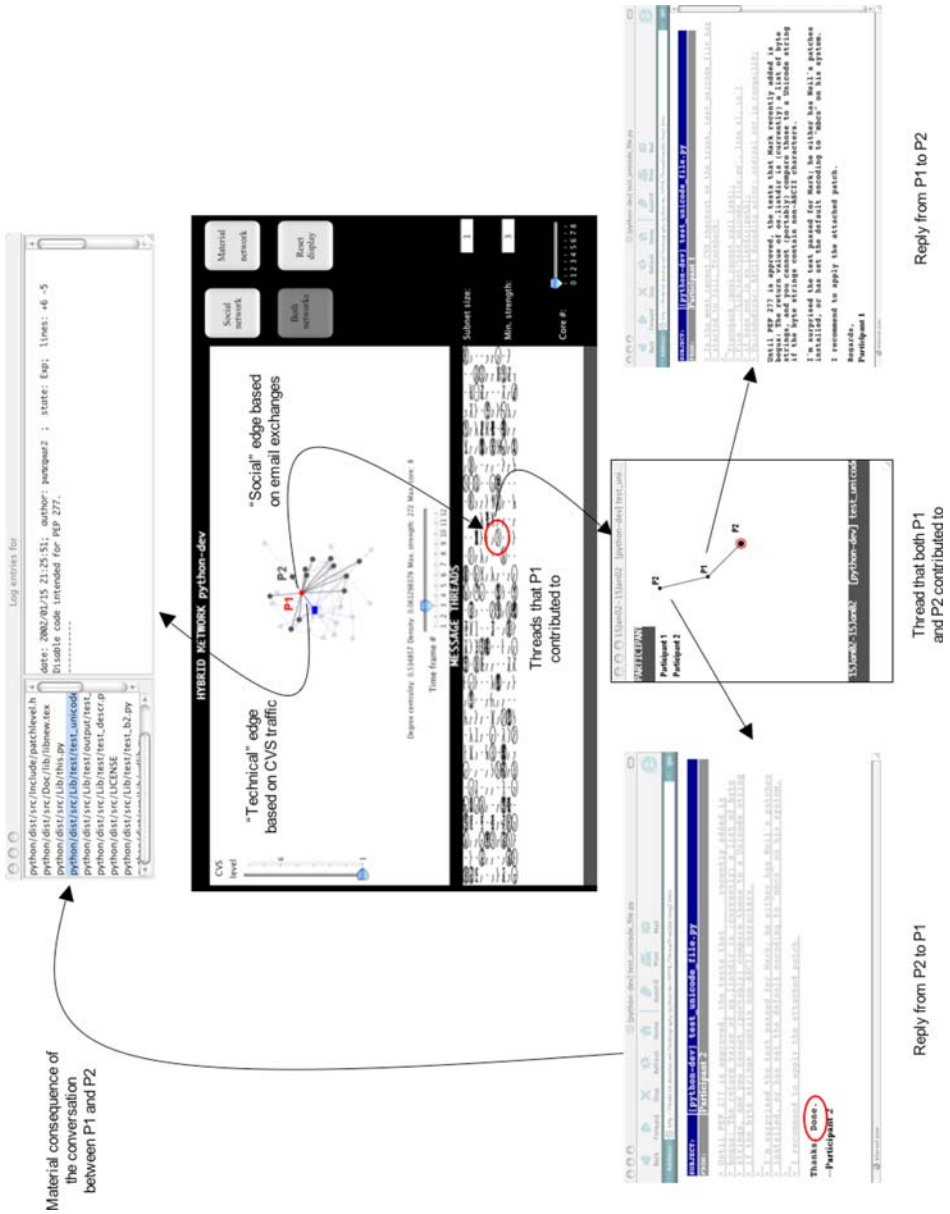

Figure 4. Analyzing the socio-technical links for a particular participant and displaying the corresponding source data.

has exchanged 143 email messages in March 2002 with another participant P2. As this is much more than most other members of the project (see Section 3.3.1.), both are connected by a fairly short edge. In the same time span P1 has committed 79 code changes to the project – also more than most of the

project's members. He is therefore connected to the Python artifact by a short blue edge.

The messages P1 and P2 have exchanged are parts of several different threads, highlighted in the lower pane (the black ovals). Here one of these threads has been expanded for deeper analysis.[2] The nodes in this view point to individual messages in this particular conversation. When read, they reveal that the team members were discussing a problem with a specific part of the project (namely, a defective test). Simultaneously, it is possible to track how P1's suggestion affected P2's coding activity by clicking on the artifact in the upper pane. This produces a list of all the commit messages P2 entered, showing that he simply disabled the offending code.

With these features, the software can be used to look for interesting patterns of activity: it is generative (Sack, 2000b), embodies a strong theory of the phenomenon analyzed (OSS projects are dynamic, hybrid networks), and therefore offers a series of starting points for the ethnographic observation of this online space. With such an interface, the researcher can now follow an individual's trajectory inside a project, qualitatively assess the nature of this trajectory by accessing the raw data, and simultaneously observe how this trajectory affects and is affected by the evolution of the hybrid network for the entire project.

I now turn to initial observations I have made using this software for "computer-aided ethnography."

3. Analyses: the case of Python

In the following section, I present the results of my analyses of participation patterns in a particular OSS project, used as a case study: Python. More precisely, I use the Open Source Project Browser to qualitatively track and analyze the trajectories of several project members who evolved (or not) into full-fledged participants. This allows me to later discuss how socialization proceeds in an OSS community such as Python, thereby allowing the project to sustain and reproduce itself over time.

I chose to observe Python for a variety of reasons. OSS projects vary across a number of dimensions (e.g. Krishnamurthy, 2002): age (number of years in existence), maturity ("1.0" and above software versus beta or alpha versions), number of developers, type of software developed, successful or failed projects, etc. All of these dimensions are likely to affect participation and socialization in the project. By studying Python, I am able to analyze community reproduction at one end of the spectrum: indeed, Python is a fairly old, large, mature and successful project. It is therefore representative of one of the classes of OSS projects that stands to benefit from an improved understanding of socialization: as discussed in the introduction to this paper, these mature and successful projects may be too dependent on the knowledge

of a few key team members (Von Krogh et al., 2003). As such, they need to make sure new participants are willing and able to extend the core members' work, should they have to leave. While Python constitutes an interesting entry point into the world of OSS, I hope to complement this analysis in future work by observing projects with different characteristics.

More pragmatically, Python also offers a very rich and easily accessible archive of data: all of its mailing-list archives are publicly available, as is its CVS repository. Python's Web site also contains extremely valuable information about its developers' community. Finally, Mahendran (2002) has already conducted ethnographic studies of one of Python's newsgroups (comp.lang.python, or c.l.py), which I used both as a background and to cross-check some of my analyses.

3.1. HISTORY OF PYTHON

In his thesis, Mahendran (2002) describes in rich detail the history of Python and its members – the following is a condensed version, augmented by information gathered from the project's Web site (Python, 2004). Python is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java. The Python implementation is portable: it runs on many variants of UNIX, on Windows, and the Macintosh. Right from its inception, the goal of Python's developers has been to produce a computer language that would be both easy to read and learn thanks to its clear and concise syntax.

The history of Python is, in many ways, connected to its founder's – Guido Van Rossum, who is now a prominent figure in the Open Source world. Guido (as he is simply referred to by most Open Source participants) created the first incarnation of Python in 1990 while working as a researcher for the Stichting Mathematisch Centrum (CWI) in the Netherlands. CWI did not officially sanction the development of Python,[3] which Guido developed mostly as a side project. During his tenure at CWI, Guido released newer versions of his software up to version 1.2.

When Guido moved to the United States in 1995 to work for the Corporation for National Research Initiatives (CNRI) in Reston, Virginia, he brought his project along with him. Development of Python continued in the same fashion as at CWI: as a side project not officially sanctioned by his employer. While working for CNRI, Guido was joined by a team of developers (such as Barry Warsaw, Jeremy Hylton, and Fred Drake). All worked heavily on Python with Guido, and this team would progressively solidify as the initial core of the project.

In 2000 Guido and the core development team left CNRI to work at BeOpen, a software startup in Silicon Valley. They did not leave Virginia, however, and all chose to telecommute. While BeOpen soon became another

victim of the dot-com crash, it financed Guido and his team's work on Python full time – allowing them to release version 2.0 of their software in a relatively short time. It is also during this time that Guido decided to stop licensing Python under its own terms and adopt the GNU Public License (GPL) instead.

After the demise of BeOpen, the entire Python team moved once more to another institution – Digital Creations, now called Zope Corporation. Here Guido and his team spend about half their time working on Python, and the rest developing the corporation's software (a web application server written in Python).

3.2. ORGANIZATION OF THE PROJECT

Python's organization and norms of participation are very explicit and spelled out in a tutorial on the project's web page. Here it is said that:

> "The most important skill Python can teach is the delicate skill of working in a diverse group. There's a core group of around 40 developers, roughly 10 of whom are very active and make the bulk of actual check-ins, and the rest of whom make occasional checkins and provide opinions and advice. Lots of people outside this core group contribute significantly, too; bug reports and patches come from core developers, well-known Python users, and complete strangers. The list of active members is always shifting because developers have differing free time, availability, and interests. To work with this large and dispersed group, *you'll have to learn who's the right person to answer a question, how to convince the other developers of the usefulness of a patch, how to offer helpful criticism, and how to take criticism*" (emphasis added).

Write access to the Python CVS tree is not automatically granted, though there is apparently no formal process to obtain it. The Web page mentions:

> "*If the python-dev team knows who you are*, whether through mailing list discussion, having submitted patches, or some other interaction, then you can ask for full CVS access" (emphasis added).

There are five "admins" listed on the project's page. These members are the only ones who can grant full CVS access. They are: Barry Warsaw;[4] Fred Drake; Guido Van Rossum; Jeremy Hylton; Tim Peters. At the time of my analyses, the Web site listed 47 other developers who had also been granted CVS access.

Over time, the members of Python have come up with a formal way to ensure that changes are carefully considered. Significant changes must be described in a Python Enhancement Proposal, or PEP. PEPs are modeled on the Request For Comments (RFC) documents used by the Internet

Engineering Task Force, and describe a proposed change by giving a fairly complete documentation for it and a design rationale. PEPs also record the community's consensus about a feature, because the PEP's author must take note of people's comments and incorporate their feedback.

The mailing list python-dev@python.org is the heart of Python's development. Practically everyone with CVS write privileges is on python-dev, and first drafts of PEPs are posted here for initial review and rewriting before their more public appearance on python-announce@python.org (another mailing list). Anyone can subscribe to python-dev, though subscriptions have to be approved by one of the "admins". The list address does accept e-mail from non-members, and the archives are public.

As I mentioned earlier, many aspects of the Python development process have been made explicitly available to both its members and would-be participants. Regarding the latter, a regular Python contributor (Raymond Hettinger) suggested in a message to python-dev some steps that newcomers should take if they want to progress quickly to the status of developer. He calls this approach the "school of hard knocks", and it apparently reflects the norms of the community enough that it has been reproduced on Python's Web site:

> "When learning a guitar, it helps to develop calluses on the fingers. *Writing a PEP is the fastest way to develop the calluses; contradicting Guido is the second fastest way; submitting a great idea is third fastest* (bad ideas either get ignored or are slammed so quickly that the scar tissue doesn't have time to develop). *Experience the politics of bug resolution*. If a developer proposed it, then it should not be dismissed lightly. If someone had a grandiose scheme in mind when they submitted the report, be prepared for wrath when you apply a simple solution. Realize that, in some cases, someone, somewhere is relying on the undocumented buggy behavior and your fixing it is breaking their code" (emphasis added).

3.3. SOCIALIZATION IN PYTHON

With this background in mind we can start to examine socialization in Python. How do participants come to play important roles in Python? Which steps do they have to take to evolve within a project such as this? Conversely, how is the project transformed by its participants' evolution?

My observations progressed as follows. Over the course of 2002, I progressively retrieved the entire email archive of python-dev (the developers' mailing list) and the CVS source tree for the project. I used this data as it accumulated both to refine earlier versions of the Open Source Project Browser and for preliminary observations of participation patterns. At the end of 2002, I used the entire archive in the final version of the Browser to refine my analyses. During all these observations I tried to uncover recurring

patterns of participation as manifested in the Browser, just like an ethnographer would look for patterns in his field notes (Emerson et al., 1995). I then analyzed in depth the participation of several project members who matched the most commonly observable patterns, reading the entirety of their message postings, CVS contributions, and any information they had made available on the Web (e.g. their home page).

3.3.1. *Four trajectories*

To give readers a sense of these recurring patterns of participation in Python, it is useful to briefly mention a few descriptive statistics. Over the course of 2002, I observed the activities of 284 unique participants in the project. Out of these, 136 posted a single message and never returned – they were subsequently excluded from the analyses since, by definition, they never evolved. Patterns of participation varied widely across the 148 remaining participants. The table below summarizes some of this data at the end of 2002 (Table 1).

It is easy to see that participation in Python is highly uneven (as in Kraft (1977), Ghosh and Prakash (2000), Maas (2004)). On average, participants converse with two other members and post about 20 messages, most of them part of relatively short threads. The standard deviations are high however – a small group of participants interacts with dozens of members, quite often during very long conversations spanning several weeks and hundreds of

Table 1. Summary of participation patterns in Python, for 2002

Social network		
Total population		148
Number of connections per participant	Median	1
	Mean	2
	Std. deviation	12.5
Technical network		
Number of active developers (at least one CVS commit)		28
Number of participants responsible for half the total number of commits		4
Total number of commits for 2002		7075
Number of CVS commits per participant	Median	55
	Mean	221.1
	Std. deviation	349.1
Conversational activity		
Total number of threads		571
Messages posted per participant	Median	3
	Mean	22.2
	Std. deviation	82.9
Number of messages per thread	Median	3
	Mean	10
	Std. deviation	24.1

Table 2. Number of CVS commits in 2002 from previously inactive participants

Participants	Number of CVS commits in 2002
Fred	385
George	78
Hector	27
Ike	21

messages. In a similar fashion, only a small sub-sample of the participants modifies the project's codebase (28, out of the 47 developers listed on the project's page). An even smaller sample (four participants) is responsible for more than half the code changes in 2002. This reflects the description of Python's organization written by the members themselves (see the quote on p. 17).

Since gaining access to the CVS database is not automatic (see Section Section 3.2.), I began my analyses by focusing on the participants who started contributing code changes in 2002. I extracted from the CVS archive a list of four participants who had not written any code in the two previous years (2000 and 2001) but began to do so in 2002. All posted messages to the mailing list before accessing the CVS database, with message postings starting between one to six months before coding began. The table above summarizes the extent of their contributions (Table 2).

As stated before, I read the entirety of the messages written by these participants and also looked at the software code they produced. There were striking similarities between their progressions over time, which I will describe shortly. Overall the trajectory of these participants reflects successful socialization in Python: an evolution from newcomer to developer.

To better understand the differences between these four participants and other members of the project I then focused on participants who were previously not contributing to the mailing list discussions and started doing so in 2002 – but who also never contributed code (56 individuals fit this profile). Six of these eventually became well connected to other project members, regularly contributing a large number of messages to significant discussions (they wrote much more than the average of 22 messages, often writing multiple responses to discussions extending far beyond the average chain of 10 messages). The remaining 50 of these contributed only sporadically, at most eight messages a year – far less than average. Moreover their messages garnered little attention: most were contained in very short threads, often only two or three messages long.

These two groups are representative of three other recurring trajectories. The first six participants illustrate "partial evolution" and the barriers some have to face before being able to participate fully and meaningfully. The 50 remaining participants illustrate how some remain at the periphery of the

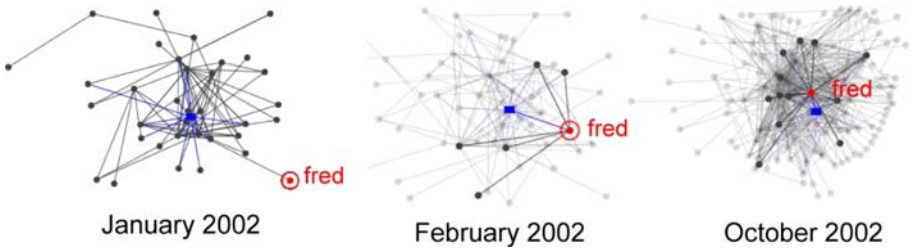

Figure 5. Fred's evolution in the hybrid network over 2002.

network and barely evolve – but for two very different reasons, as we shall see below.

Due to space constraints, I have chosen to describe in detail only the first trajectory in the remaining of this paper, since it is closest to the theoretical issues I want to address here. I will draw on selected observations from the other three trajectories to inform my discussion in Section 4.[5]

3.3.2. *A case study of successful socialization: Fred*

Let us now illustrate successful socialization in Python through a concrete example. The participant I selected is Fred,[6] a developer on a project based outside of the United States. Fred writes applications in Python for his employer, and also for fun. He has, for instance, developed a set of utilities to make programming in Python easier. The tight connection between Python and his work certainly provided a strong incentive for Fred to become a member of the community (this motive is, in fact, explicitly spelled out on his web site). Improvements in Python could make Fred's job easier; in using Python regularly he also encounters bugs and possibilities for improvements that he has to address no matter what. All of these factors are consistent with earlier research on individual motivation in OSS projects (Lerner and Tirole, 2002). While not a necessary and sufficient condition to successful socialization, this particular individual background is clearly useful. Out of the four participants I listed earlier, another one beside Fred had strong professional ties to Python.

A look at Fred's position in the hybrid network over time (Figure 5) shows that, starting in January of 2002, he progressively became very tightly enmeshed in a web of relationships with both the project's artifacts and some of its participants. To understand his evolution, however, we need to go beyond this aggregate view and look at the qualitative data in more details.

Fred's first contribution to python-dev in 2002 was to ask questions about possible changes in Python's architecture that could affect the application he was developing for his employer. His early messages show that he was monitoring the development mailing list for peripheral awareness, in order to keep abreast of any evolution of the language that could affect his work:[7]

‘

> ‘I have an application [...] that worked with Python 2.0 and 2.1. Now it fails under 2.2. Under 2.1. it appears that certain objects were unpickled first; under 2.2. that is no longer the case. Anyone got [an] explanation for [my] complaint?’’ (Fred, January 10, 2002)

In actor-network terms, Python appears like a ‘‘black-box’’ (Latour, 1987b; Star, 1995) to Fred at this stage. For him, the project looks like Figure 5 above: a tangled web of material and social relationships that has to be understood for his personal work to progress. By asking questions and making connections with some of the project’s participants, Fred is trying to make the structure of this network more visible to himself: he is ‘‘probing’’, discovering in the process which parts of the network relate to his work.

In the course of developing his applications, Fred progressively uncovered bugs in Python. He reported them to the mailing list, frequently adding a proposed patch (a few lines of software code, included at the bottom of his messages) to fix the problem, so that others could integrate it into the source code. As shown in Von Krogh et al. (2003), the practice of simultaneously identifying a technical problem and offering software code to resolve it is central to successfully joining an OSS project. Here it helped Fred build a reputation as a good bug fixer, which in turn resulted in his obtaining CVS commit rights so that he could integrate his patches immediately, without going through another developer’s approval:

> ‘‘I have just discovered a bug in [this library]. See [bug report URL]. The fix, as near as I can tell, is trivial:
>
> [Code segment]
>
> I’ll check this in and close the bug unless anyone complains.’’ (Fred, February 1, 2002).

From a theoretical standpoint however, bugs like the one above deserve particular attention. Indeed bugs are areas of weakness in the project’s hybrid network: they represent a part of the project where controversies still exist, where the usual ‘‘strong rhetoric’’ (Latour, 1987b) about the project becomes weaker. Bugs are, therefore, critical moments in the history of a technological project.

Dealing with such controversies is not a simple matter of ‘‘closing the bug.’’ A more complex process is involved that extends beyond posting a technical solution to the mailing list, unlike what earlier accounts of OSS participation could lead us to believe (e.g. Von Krogh et al., 2003). To reach the point where the controversy can be resolved, a network of allies has to be assembled first (Latour, 1987b). Indeed statements regarding a controversy are weak if they

are left alone. To make a statement stronger, it needs to be connected to what others have said beforehand. This way anybody opposed to the solution offered has to attack not only the solution and its provider (Fred), but also a string of other propositions and assertions made by others beforehand.

In the hybrid network for February 2002 (Figure 5) representing the moment when the bug-fixing episode above took place, it is clear that Fred is not alone when he attacks the problem: from this figure, we can see that Fred is connected (through previous conversations) to other actors in the network, some of them quite important ones (such as Barry, Guido, and Martin). Fred's activities are therefore backed by the weight of a significant number of ''allies'', just like a statement in a scientific paper is when it is accompanied by a large number of references and citations. It is only because of these connections that he can make offhand comments such as ''I'll close the bug unless anyone complains.'' Put differently, what he means is ''my solution to this bug is backed by previous discussions with Barry, Guido and Martin. If you don't agree with my proposal, you will have to demonstrate to all of us why I should proceed differently.'' Clearly, while proposing sound technical solutions to problems is an important aspect of Fred's successful participation, these solutions are not enough by themselves: establishing strategic links with key members of the project beforehand is what truly allows them to be respected and accepted.

After fixing several bugs like the one above, Fred was able to move from one position to another in Python: from a user simply monitoring activity he had become a bug fixer directly contributing changes (albeit small ones) to the project's architecture. It is interesting to remark that I found no evidence of coaching of any kind from the project's members: the acts of finding bugs, reporting them, and proposing a solution to them all stem from the participant's initiative. In this context the fact that Fred is already an experienced developer, intimately familiar with the software development process, certainly helped him know what, when and how to propose anything to the list members.

Moving to this stage of participation, therefore, requires skills. Some members, such as Fred, are able to identify important controversies and enroll a network of allies to attack the problem. This legitimates their contributions, and eventually results in them obtaining CVS access so that they can directly modify the project's source code without review. Through the connections they establish with both the artifacts and other members of the project, these participants become progressively more enmeshed in the project's hybrid network – which, in turn, puts them in an ideal position to influence the composition and structure of this network, as we will now see.

Indeed, having reached this new position in Python, Fred later proposed a more substantial contribution: to integrate his module (initially developed for his personal use) to the standard library:

> "Hi all,
>
> I would like to propose adding my module to the standard library. [...] Please take a look at [module URL] for the whole story, including all the documentation and code via CVS." (Fred, February 11, 2002).

At this point, the core members of the project stepped in more heavily than before and asked Fred to justify why his module should be integrated, and what made it better than the other available options:

> "No immediate objection, although there are some other fancy packages around, and IMO you have to explain why it is better." (Guido, February 11, 2002).

This generated a large volume of discussion on the mailing list, reflecting interest from the community. As a next step, Fred was therefore given control of a separate mailing list over which he could discuss the merits of his module in more detail with other interested parties. He was told to come back to python-dev later with a detailed rationale supporting the adoption of a refined version of his module.

Qualitatively, this proposal is quite different from simply fixing bugs. In proposing to add his module, Fred is not addressing a pre-existing controversy: he is creating one. To add his module, the very fabric of the project's hybrid network has to be challenged: relationships between people, between artifacts, and between people and artifacts would have to be changed to accommodate the addition of a new piece of software. And as was the case with bug fixes, Fred needs to build up a network of allies to support this effort – a fact that Guido makes plainly visible by saying that Fred needs to explain why his module is better. In other words, Fred's assertion about the usefulness of his module is not yet backed by a sufficient number of allies in the network. Indeed he is challenging a socio-technical order built upon "layers of assertions" and "fortified to resist a hostile environment" (Latour, 1987b). To transform the established order, a tangled web of social and material connections has to be understood, challenged and recomposed. As can be seen in Figure 6 below, one cannot just simply insert a newartifact in Python: a complex, stable network of inter-relations has to be strategically weakened first.

In suggesting the addition of his module openly though, Fred creates an opportunity for himself to start gathering support, which he obtains a short time thereafter: a separate mailing-list is created to discuss his project in more details. In actor-network terms, the charter of this mailing list is quite clear: to craft an alternative "strong narrative" (Latour, 1987b), a network of assertions and allies translating the current state of the network into something that can accommodate the addition of Fred's module.

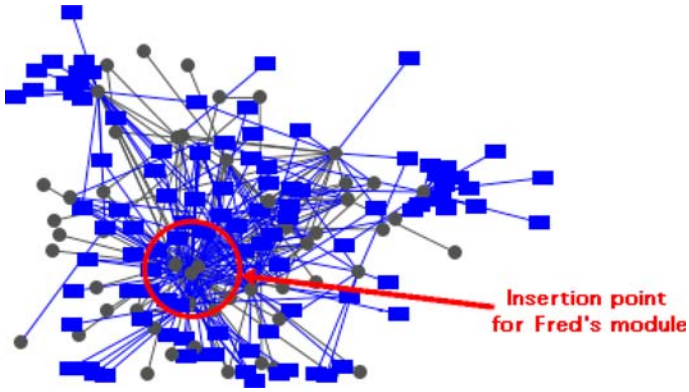

Figure 6. The hybrid network that Fred is challenging.

While discussion on this dedicated mailing list was going on, Fred did not disappear from python-dev: instead, he maintained his status by contributing to PEP discussions and debates around new features, suggesting additions to the architecture in the process. The next example message illustrates how Fred, in the process of polishing his module for adoption, regularly comes back to the list with suggestions for improvements that are directly connected to the work he is doing on his module:

> "Hidden away in [package name] is an exceedingly handy function [...]. Surprise surprise, [my module] uses this. I've never been terribly happy about importing it, and putting [my module] into the standard library is a great opportunity for putting it somewhere more sensible. [...] Proposal: a new standard library module [...]." (Fred, June 1, 2002).

On top of simply maintaining his visibility in python-dev, the kinds of activities Fred engages in above serve an additional purpose. Indeed the bugs he is fixing are not random: they deal with pieces of Python's material architecture that will be directly affected by his new module. In modifying these artifacts with his patches, Fred is already working on enrolling the network of allies that he needs to get his module accepted. With these modifications done, Fred has set the stage for the addition of his module. From this point on, each and every assertion that he will make about his new module will carry the weight of the work he did on these other, connected pieces of software. In other words he is strategically positioning himself so that he can "capture" (the term used by Latour (1987b) is "captation") a part of the network, and therefore make it seem like the changes he is suggesting with his new module are unavoidable.

After a few months spent working on improvements to his module, Fred eventually came back to python-dev with a summary of the activity that took

place on the mailing list dedicated to his project. He then proposed to discuss the specifics of how to integrate his module into Python:

> "I think it is time to declare the work of the [dedicated mailing list] finished: several competing proposals were put forward, but [my module] appears to be the only really complete, documented, field-tested (by someone other than its author) library. Not everyone will agree, but I think that's the broad consensus. [...] The main issues are [...] how the standard library should be rearranged to make this interface unobtrusive and efficient." (Fred, May 30, 2003).

In the message above, note how Fred immediately lists the network of allies behind his project: it is field-tested by someone other than its author, and a majority of the discussion group members agree that it is the most complete module. Criticizing Fred's module, therefore, is not only criticizing him but a larger number of participants who have become enrolled in Fred's effort. His use of "consensus" is also interesting: "not everyone will agree, but the broad consensus is..." Decision by consensus is done if and only if everyone agrees. Here, Fred redefines "consensus" to serve his own interests – in Latour's terms, he "translates" (Latour, 1987b) the interests of others to align them with his own.

Moreover, we know that Fred has also been actively preparing for the insertion of his module into the code database – he also has a network of material allies through his previous bug fixes. It should not be surprising, therefore, that the discussion Fred is starting at this point concerns "how the standard library should be rearranged to make [his module's] interface unobtrusive and efficient": from the position Fred has acquired in the hybrid network this looks like the only logical next step. In fact, the debate has now moved to the technicalities of how Fred's module should be inserted, and does not question the adequacy of this addition anymore. Unsurprisingly perhaps, Fred's efforts will eventually pay off in the message below, in which the project's leader proposes to integrate the new module into the next planned release of the language:

> "I want to start working on an alpha release [...]. One of the tasks is to adopt Fred's module [...] any comments?" (Guido, November 13, 2003).

At this point Fred has managed to directly affect the entire structure of the project's hybrid network, in which he is now tightly embedded (see October 2002 in Figure 5 above). All his previous actions, however, are hidden from the view of an outside observer: the process has been black-boxed. But with the software I am using here, it has been possible to "go back in time and space to the point where the black box was still a controversial topic" (Latour, 1987b) and understand how Fred progressively reached his privileged position.

4. Discussion

The example above illustrates quite clearly, I believe that the example above illustrates quite clearly, one of the trajectories that a diverse set of participants can follow in a project such as Python. Most importantly, it allows us to see how complex and nuanced socialization in an OSS project can be, extending earlier cross-sectional research (Von Krogh et al., 2003).

In this section I will revisit these trajectories from a more theoretical angle and propose several interpretations of socialization and community reproduction in the world of OSS. First, I describe socialization as an individual learning process whereby participants are progressively integrated into the project as they build their identities. Second, I adopt a more macroscopic, political perspective and analyze socialization as a series of ''trials of strength'' (Latour, 1987b), whereby OSS participants enroll material and human resources to align the project's hybrid network with their own objectives. These two perspectives are complementary, and each shed a different light on the reproduction of online communities such as those of OSS. Based on this analysis, I later discuss the possibility of developing software to better support socialization in OSS projects.

I start below with the first perspective socialization as a learning process.

4.1. OSS socialization as a learning process

From the observations I have made earlier, it appears that those who successfully evolve to reach the status of developer in Python have to go through a series of well-defined steps. These, of course, represent an ''ideal type'' trajectory in the Weberian sense (Weber, 1949) – not every successful Python participant follows this trajectory exactly, but most are reasonably close to it. The particular steps involved are: (1) peripheral monitoring of the development activity; (2) reporting of bugs and simultaneous suggestions for patches; (3) obtaining CVS access and directly fixing bugs; (4) taking charge of a ''module size'' project; (5) developing this project, gathering support for it, defending it publicly; (6) obtaining the approval of the core members and getting the module integrated into the project's architecture. Some of these steps had been recognized in previous research. Von Krogh et al. (2003), for instance, clearly describe how successful ''joiners'' spend a significant period of time ''lurking'' at the periphery (Nonnecke and Preece, 2003), simply observing activities (step 1). They also emphasize the crucial role of ''starting out humbly'' by contributing technical solutions to already existing problems (steps 2 and 3), before moving on to more significant accomplishments (step 4).

In many ways, therefore, such a trajectory supports the hypothesis that a process related to legitimate peripheral participation is at play in OSS projects. Through an initial period of observation, newcomers can assimilate

the norms and values of the community and analyze the activity of the experts. To evolve any further, they have to start building an identity for themselves and become more visible to the core members. Indeed, as Lave and Wenger (1991) proposed, learning involves the construction of identities and is itself an evolving form of membership.

This can be illustrated even more vividly by comparing Fred's trajectory above with David's (one of the 50 "static" participants I mentioned in Section 3.3.1. See also Ducheneaut, 2003). David posted his first message to python-dev in March 2002 – a simple bug report, without any suggestions how to fix it. One week later, he asked the developers community to consider a module of his creation for integration in the next release. In trying to go too fast however, he only received contempt from the core members of the project and his proposal was quickly rebuffed. David did not take any time to build an identity for himself; his later contributions were therefore limited to a couple of bug reports, after which he left definitely.

Establishing an identity for oneself, however, is not a guarantee of becoming a developer. Not only must the participants demonstrate that they have the necessary technical expertise, they also have to prove themselves as "artificers" by crafting software code publicly. Jeff, one of the 6 "partially integrated" participants from Section 3.3.1., illustrates quite well how one can be recognized as a technical expert but not a craftsman. Jeff slowly built up and demonstrated his expertise at the periphery of python-dev by answering questions from "newbies" in c.l.py. Once he had joined python-dev, however, the qualitative nature of his contributions did not change: he kept discussing the technical features of the language at a theoretical level, never proposing any new code by himself. As a consequence, he evolved to become a good source of advice and somewhat of a philosopher in the project, but not a developer. This suggests that expertise is not enough to become a core member in Python: one also has to create material artifacts.

Finally, another participant I observed (Boris, another of the group of 50) shows that peripheral participants are not necessarily novices to the project who are trying to make their way to the core. Long-time project members like Boris (he joined in 2000) also "retire" at the periphery when they don't have the necessary time or energy to contribute to the project anymore. This way they can contribute their historical perspective and insights to the contemporary problems faced by the development team.

From the above, it is clear that understanding the process of identity construction is primordial to analyze how participants evolve in OSS communities, and how they eventually become socialized into them. In Python, two complementary strategies can be used to become more visible and start establishing an identity for oneself. First, one can actively contribute to PEPs and features discussion. By contributing to the features review process, one can gain a reputation as a peer equipped with enough technical skills to make

meaningful suggestions. Second, one can submit bug reports and, simultaneously (this is important), a proposed solution to fix these bugs. If this is done frequently and the proposed solutions are deemed to be appropriate by the project members, a participant can be given CVS write access, thus becoming a patcher – the first step in which control over a material artifact is given.

Once a participant is given the right to craft material artifacts by himself, he then has to demonstrate a higher level of mastery by taking charge of a sub-module of the project if he wants to keep evolving (this, however, is not necessary – some participants are perfectly happy to remain patchers for the rest of their tenure in a project). Again, the parallels with traditional apprenticeship learning are striking; apprentices working on cathedrals in the Middle Ages, for instance, always started by working on small projects (e.g. small details on a sculpture) before undertaking a major project by themselves that would define them as a craftsman (e.g. an entire sculpture).

Moreover, the output of the work on this module will be evaluated during a rite of passage, where the entire community scrutinizes what has been produced and the core members finally deliver a verdict of acceptance or rejection. From a theoretical standpoint, it is interesting to remark that this progression is very close to what Turner (1969) describes as a ritual process. Indeed, Turner proposes that individuals moving from one social condition to another have to go through a liminal stage, during which they are stripped of their earlier attributes. During this stage ''their behavior is normally passive or humble; they must obey their instructors implicitly, and accept arbitrary punishment without complaint. [...] Among themselves, neophytes tend to develop an intense comradeship and egalitarianism.'' The first part of this description definitely fits what happens to OSS participants like Fred when they unveil the result of the work on their proposed additions to the project. There is, however, no evidence of an ''intense comradeship'' between OSS peripheral participants; in fact it is quite the opposite, as they are all competing for the attention of the core members in order to obtain CVS access rights.

In the process of becoming an ''insider'', a newcomer to Python also has to learn how to tell stories. Work in this project is not simply about crafting a material artifact: it is also about crafting and maintaining social relations (Orr, 1990). When making the transition from the periphery to more central roles inside the project, a participant has to embark on a story-telling procedure that will construct a coherent account of how the differences between his contributions and the current state of the project can be reconciled (see how Fred had to construct, over several months, an account of how his module would ''fit'' in Python). These stories are passed around publicly, and can be reused and modified for the purposes of other participants. This way a

participant's experience is socially constructed and distributed within the project (Seely Brown and Duguid, 1991). But more importantly, the ability to tell such a story is a mark of identity and membership, and an integral part of the rites of passage a participant has to go through during his evolution. Eventually, if the work of the participant is finally integrated into the project's architecture after the rite of passage above, another step has been taken: the participant is now a maintainer.

The above process illustrates how successfully contributing to an Open Source project depends much more on a complex socialization process than on a show of technical expertise. As one of Von Krogh et al. (2003)'s interviewees aptly described, ''There is the person who says, 'I am a Java engineer [...]. I have been working for five years, and I really would like to help. Give me something to do.' This person tends not to do anything.'' Python's would-be participants are generally highly skilled developers too; it appears clearly, however, that those who follow the process I described earlier stand a much better chance of becoming important actors. There is also little evidence of explicit coaching or teaching from established experts. Instead, the participants have to discover by themselves what the norms of participation are.

Python-dev, therefore, is not a place for novices to learn about computer science – this knowledge is assumed. What the newcomer has to learn is how to participate and how to build an identity that will help get his ideas accepted and integrated. When talking about legitimate peripheral participation in Python, therefore, one must not equate ''apprentices'' with the novices of traditional apprenticeship. This is surprising in view of Hars and Ou's (2000) finding that over 70% of the respondents to their survey reported the desire to improve their programming skills as being their primary motivation to contribute to OSS. From my analysis it looks as if participants come already equipped with good programming skills, and learn instead how to contribute meaningfully to a fairly large-scale project such as Python. Participation is more a demonstration of one's value as a developer and a way of learning the mechanics of distributed software development than an opportunity to acquire formal knowledge.

Interestingly, the rites of passage and the canonical trajectory I have uncovered above through empirical observation are partially documented on Python's Web site. The steps required to evolve from being a user to becoming a member of the outer layer of the project, for instance, are explicitly described in Raymond Hettinger's ''School of Hard Knocks'' document (that is: start by finding bugs and describing how to fix them; contribute to PEPs). From my observations it appears that these norms of participation are indeed enforced. Participation beyond this point, however, is governed by more implicit norms that I also uncovered above.

4.2. OSS SOCIALIZATION AS A POLITICAL PROCESS

It is important to remark at this point that very few people are as successful as Fred – as I mentioned earlier, only three other contributors had a similar progression. Many others (50 individuals in 2002) stopped at the bug reporter stage, or did not evolve at all. I think this remarkably low level of evolution can be explained, at least in part, by some of the more political aspect of OSS development, the consequences of which are visible in my observations.

Open Source is often described in the popular press as the panacea of distributed collaboration. It is supposed to be collegial and similar to academic research (Bezroukov, 1999). Yet, processes of cooperation in no way insulate their participants from considerations of power (Divitini et al., 2003). Supposedly meritocratic institutions, like science for instance, cannot be isolated from politics – some have even said that "science is politics by other means" (Latour, 1987a, p. 229). Indeed for Latour scientific research is akin to a "Machiavellian" process where scientists, to win support for "their" theory, engage in various power games to recruit allies and vanquish their foes. My analyses of participation in Python illustrate how socialization in this community is also, by nature, a political process. To paraphrase Latour, "Open Source Software (OSS) development is politics by other means." But, while the politics of software development projects have been recognized and discussed earlier (Block, 1983), the research literature on OSS tends to have glossed over the issue.

As we have seen, an OSS project is essentially a hybrid network – a heteroclite assemblage of human and non-human actors, entangled in specific configurations that may vary over time. The skill of Python's software engineers resides in their ability to create the most stable network of connections between these various pieces so that the project can withstand the test of time. Indeed when Latour (1987a) talks about "foes" that have to be "vanquished," this should not be interpreted in a literal sense. "Foes" are not akin to human adversaries in a political game, and vanquishing a foe during OSS development is not analogous to defeating an opponent during an election. It is the composition of the hybrid network, the strategic relationships between actants, which are used by stakeholders to control a project. In other words it is often impossible to point at an individual adversary: *the "foes" here are the entire network*, designed to resist change, which must be weakened in strategic areas and eventually reconfigured if a participant's contribution is to be accepted (or, in the case of Latour, if a new scientific theory is to gain legitimacy).

A "trick" used to fortify a hybrid network is to "black-box" the relationships between actants: the process through which connections were established in the first place is hidden from view, so that anyone who would like to challenge the current state of the project will have to uncover these

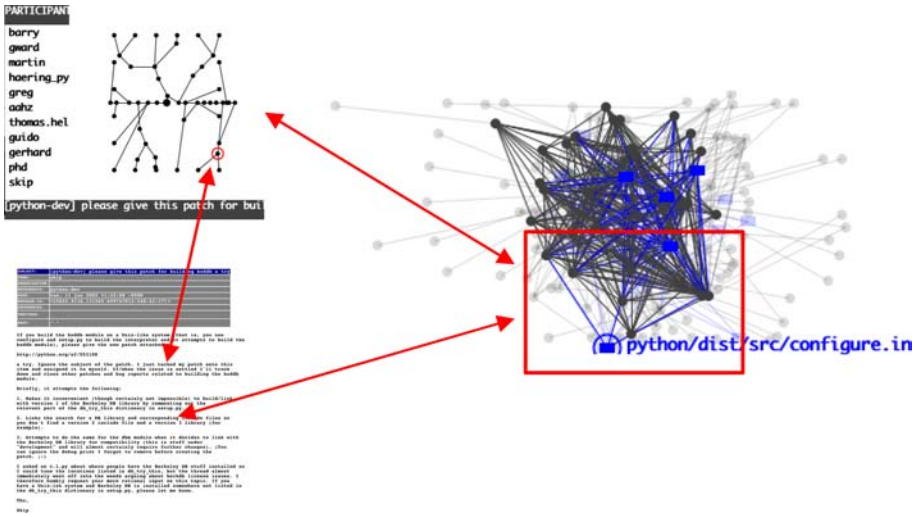

Figure 7. Controversial discussions reveal part of the hybrid network's structure.

relationships first. And indeed there is no information, no documentation to be found on Python's Web page that explicitly spells out the material and social structure of the project. Instead, it is said that "you'll have to learn who's the right person to answer a question, how to convince the other developers of the usefulness of a patch" – in short, you will have to understand the structure of the project's hybrid network by yourself.

A very important first step for newcomers is, therefore, to progressively open this "black-box." In Python, there are privileged places where this can be achieved. Complex strategic discussions regularly take place regarding the addition of new features to the system, either formally through a PEP or more informally when a participant publicly proposes a new idea. These discussions are highly controversial and, as such, make the structure of the project's hybrid network more visible to its participants.

Indeed, to justify adding a new piece to the project or to modify an already existing one, it is necessary to invoke "resources coming from other times and spaces" (Latour, 1987b). In other words, the champion of a new idea needs to articulate what the current relationships between artifacts, between participants, and between artifacts and participants are – and only then illustrate how this particular arrangement poses a problem, and what alternatives could replace it. An attentive observer can, therefore, piece together at least a partial image of the project's hybrid network from these exchanges (see Figure 7) – and later use it strategically.

Another approach, initially used by Fred, is to "probe" the network to reveal its structure. By asking simple questions about the current state of the project, a participant can see from the responses he obtains who is connected to a particular artifact, and what the nature of this connection is (Figure 8).

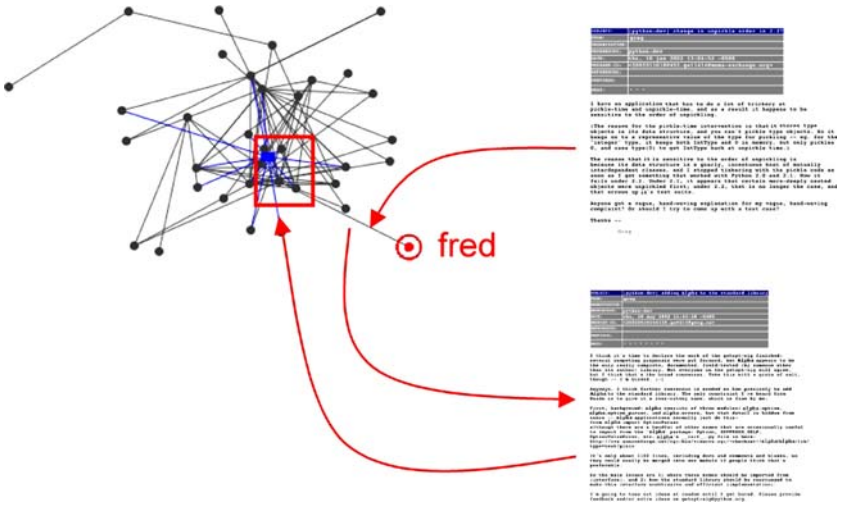

Figure 8. "Probing" the hybrid network help reveal some of its structure.

Note that the behavior and tone of the participant during this probing is key: as we have seen in Section 4.1. it is best to behave in a humble manner, to avoid claiming expertise without backing it up with code. A participant's behavior, the way he projects his identity, is part of the range of strategic interactions he can use to uncover the structure of a project's hybrid network.

Contributing to Python means, therefore, that a participant will have to figure out how to insert himself and his material contributions into this network – and this is where the process becomes political. Indeed, based on his understanding of the project's hybrid network, a participant can start to enroll allies (both human and material) to support his efforts. Participants like David, who do not take the time to understand the structure of the network and immediately try to modify it by inserting new material contributions, are bound to fail: the very purpose of the network is to resist such brutal changes. To achieve his objectives, a participant has to learn how to subtly manipulate and transform the relationships between actants instead.

To enroll allies, one has to align the interests of others with one's own – a process Latour calls translation (Latour, 1987b). Fred, for instance, progressively obtained the support of several of the core members with his detailed bug reports. Through his understanding of the hybrid network, he was able to identify areas of weakness in the project's technical infrastructure. This put him in a position to make an implicit proposal of the following form: "You need me to keep your system functioning properly." Once he had convinced the core members of the value of this proposition, he was granted CVS access to submit his patches directly.

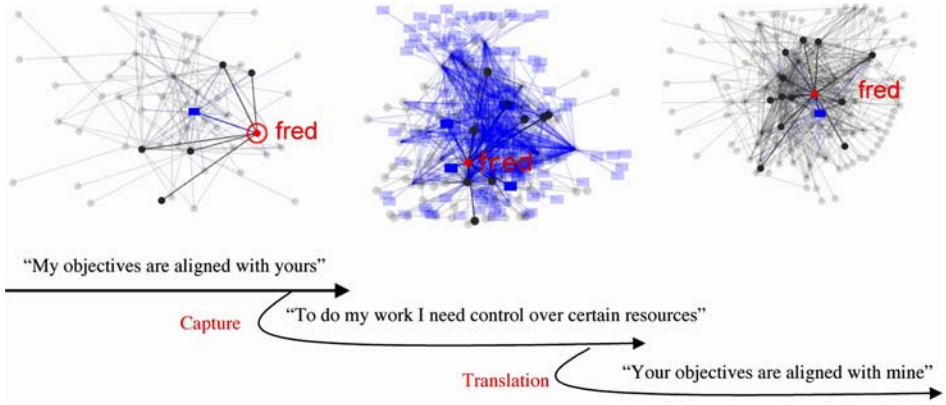

Figure 9. Capture and translation are essential to successful participation.

After he had been given control over technical artifacts, Fred was in a good position to propose another translation. This time, he suggested the addition of an entirely new software module to the project. This proposal was of the form: ''It would be beneficial to Python to have my module included.'' This is qualitatively different from the previous proposal: first, Fred demonstrated that his objectives were aligned with those of the project members. In this second proposition, he is subtly suggesting that the project needs to be aligned with his personal objectives instead.

To back up this statement, Fred relied on foundations he had set up earlier. Indeed, through strategically chosen interactions with material resources (software modules that he fixed) and other participants, he had started to ''capture'' a section of the hybrid network. From there, the changes he is suggesting look ''obvious'' or ''natural'', because they follow a path controlled by him or his material and human allies. All of this is illustrated in Figure 9.

Participation, socialization, and community reproduction in Python are therefore inherently political. Successful participants are those who can ''read'' the hybrid network of a project, identify areas of weaknesses and, based on this, recruit material and human allies to subtly align the interests of the project with their own. This requires skills, and only a few participants actually succeed in altering a project's hybrid network.

4.3. SUPPORTING SOCIALIZATION IN OSS PROJECTS

The above analyses have been, so far, entirely descriptive. I believe, however, that they have the potential to positively influence the way OSS projects are organized. To this end I would now like to discuss some implications of my analyses for the design of computer systems intended to support OSS development.

It is clear that successfully socializing new members is valuable to OSS projects; yet, as we have seen, few participants reach the end of their journey. Of course, this is not entirely problematic: by adopting a somewhat distant attitude, the project leaders make sure that they do not have to constantly "hold the hand" of newcomers and waste an inordinate amount of time introducing them to the subtleties of software development. A certain amount of selection is necessary, if only to allow the core members to focus on their tasks – this is clearly visible in the documents available on Python's web site (see for instance the quote on page 17 of this paper: "*You* will have to learn...", meaning, do not expect any significant coaching on our part). Moreover, the obstacles put in the path of newcomers function as trials and rites of passage that are important to ensure these individuals are a good fit for the project. They play an important role in the development of these participants' identities (Section 4.1).

To respect these constraints it seems, therefore, that computer tools could be potentially useful in two ways. First, they could help expedite the journey of newcomers who end up being important contributors. Indeed successful OSS participants often wait several months before feeling they can contribute to the technical discussion – a very significant delay (see Von Krogh et al., 2003). Given appropriate resources this delay could probably be reduced, accelerating the influx of new and useful ideas into the project without increasing the core members' workload. Second, computer tools could help would-be contributors who have the right technical skills but lack the necessary political acumen to promote their ideas. By making the project's socio-technical network more "readable," computer tools could facilitate the recruitment of allies that is central to successful socialization. Note that, with such tools, newcomers would still have to go through the steps required of them to be recognized as valuable members (e.g. behave humbly, contribute code related to important technical issues, etc.) – the goal here is to bootstrap the socialization process, not to bypass it. It is also worthwhile to remark that these two suggestions differ significantly from what is currently offered in tools designed to support OSS development (such as Matsushita et al., 2003), which focus essentially on the support of software engineering tasks.

While the OSS Browser was designed primarily for analytical purposes, it could also serve as a foundation to design a more user-oriented system in support of the above. Successful contribution does not happen instantaneously and clearly requires a significant amount of strategic planning and analysis from newcomers (however subconscious or unacknowledged these activities are). In fact, the activities of successful contributors are, in some sense, not too different from those of a researcher like myself: both aim at uncovering the hidden socio-technical fabric of a project. Therefore, some features of the OSS Browser could be particularly

useful to newcomers. Indeed, they could support and accelerate the following socialization steps:

(1) *A period of ''lurking'' to assimilate the project's culture and identify the areas in need of new contributions*. This peripheral participation could be facilitated in the online environment of an OSS project with an interface highlighting the most dynamic and controversial conversations: as I mentioned above, these usually reveal the underlying socio-technical structure of a project during ''trials of strength.'' If newcomers could easily focus on these particular conversations, they could potentially get up to speed more quickly. Yet the computer interfaces used by OSS participants to converse with each other are surprisingly crude, and often limited to ''off-the-shelf'' email and newsgroup clients. Instead, the lower pane of the OSS Browser could be a first approximation of an alternative interface facilitating the identification of current and past controversies. But much more could be gained by putting into practice current research on Persistent Conversations (Erickson, 1999) and the visualization of online discourse (Sack, 2001; Smith and Fiore, 2001). At a minimum, my analyses suggest that the following characteristics would be important to highlight: the total number of unique participants to the conversation, its relative proportion of core contributors versus other project members, its dynamism (e.g. average delay between replies), its tone and topic (e.g. by doing a thematic analysis as in Sack, 2001), its overall length (to separate active but short-lived issues from on-going debates), and finally its level of controversy (e.g. by looking at the number of branches in the conversation tree, as opposed to more simple, linear conversations).

(2) *Enrollment of key allies in support of future work*. As we have seen earlier, identifying areas in need of work is not sufficient to guarantee a successful contribution. On top of it, would-be contributors need to enroll human and material allies in support of their propositions. An interface similar to the hybrid network of the OSS Browser could be helpful in this respect. First, it would allow its user to go back in time and identify who contributed to the areas they are interested in. These contributions can be technical (who committed changes to this specific part of the software architecture) or more indirect (for instance, email messages suggesting important ideas about what to implement). More importantly the allies of these contributors could also be identified: whom they tend to talk to, and which part of the infrastructure they tend to work on. This would allow a would-be contributor to better ''translate'' the interests of this heterogeneous group and align them with his own. Again and respecting the constraints outlined earlier, contributors would still have to produce this translation on their own – but the tool could help them find the necessary information faster and more easily.

4.4. REFLECTIONS ON "COMPUTER-AIDED ETHNOGRAPHY"

The analyses presented in this paper are based on a novel methodology, inspired by the burgeoning tradition of "software-as-theory" (Dumit and Sack, 2000; Sack, 2000b, 2001) and earlier calls for the use of computer tools in sociological investigations (Teil and Latour, 1995). While I believe this approach offers many advantages, it is of course not without flaws. I would like to conclude this section by taking a critical look at some important limitations, both practical and theoretical.

4.4.1. *Theoretical limitations*

The OSS Browser was designed to embody a concrete representation of Latour's (1987b) concept of a hybrid network. This was accomplished by placing software code and individual participants on the same footing in the Browser's interface. However, it is worth noting that Latour's concept of an actant (an element in the hybrid network) encompasses much more than people and computer artifacts. Other actants could include, for instance, entire organizations (e.g. the Free Software Foundation), infrastructures (e.g. the Unix operating system, the Internet), other technical artifacts (e.g. bugs stored in tracking databases such as Bugzilla), etc. Therefore the OSS Browser is in essence partial, and analysts should carefully consider the influence of resources that are not immediately visible in the computer interface. I believe my analyses have focused on the aspects of Python's hybrid network that most directly influence socialization and, as such, the absence of these other actants is not too problematic. Still, the question of how to include a larger set of actants in analytical tools such as the OSS Browser remains an interesting avenue for future research.

The use of ANT as a framework for the analysis of socialization in OSS projects could be subjected to the same criticisms ANT itself received, and the limitations of this theoretical framework should therefore be acknowledged. In particular, ANT has been criticized for over-emphasizing goal-directedness, for having too much of a "managerial, engineering, machiavellian" character (Latour, 1999a) or for being "excessively strategic" (Law, 1999). Actants can appear "flat" (Latour, 1999a) and without much humanity: machinations are foregrounded, emotions backgrounded. Despite Latour's insistence that "we are not in command, we are slightly overtaken by action" (Latour, 1999b), there might indeed be a tendency in ANT to portray individuals as heartless, instrumental manipulator bent on accomplishing their goals.

It is in part to compensate this tendency to provide only "dry" accounts of strategic machinations that the OSS Browser allows easy access to the raw, qualitative data it processed. My analyses are not based on disembodied accounts of a person's activities: they draw directly on the text these

participants wrote "in the heat of the moment," so to speak. I believe this proximity to the research material reintroduces some of the humanity that some feel is missing from ANT. Still, it must be acknowledged that more could be done. For instance, in-depths interviews with Fred or similar participants could enrich our understanding of why and how they chose to participate in Python. This, in turn, might help nuance some of my analyses by framing each participant's actions in a larger socio-cultural context.

Another potential bias with ANT is the privileging of one perspective, most forcefully demonstrated by Star (1991). She argues that, despite ANT's capacity to account for heterogeneity and multivocality, there is often "only one kind of multiplicity, one kind of power, and one kind of network." In her critique Star reminds us not to focus on the existing networks only, but also to make visible the conditions necessary for alternative networks to emerge. Using the McDonald's fast-food chain as an example, she shows how ANT would usually focus on the heterogeneous network of technologies (frying pans, counters, etc.) and individual routines (scripted replies, workflow) "aligned" to produce a hamburger. She illustrates ANT's problem with the following question: what would it take to produce something that falls outside the actor-network such as, for instance, a hamburger with no onions that a customer is allergic to? This requires an alternative, competing network to be viable.

The analyses I have presented in this paper focus essentially on how Python's hybrid network is built to *resist* change, and how newcomers rely on translations to become members of the community. This focus on the reproduction of a pre-existing order is, therefore, directly subject to Star's criticism. What kind of conditions would be necessary for radically different socialization patterns to emerge? Exploring these alternative, competing hybrid networks would be a natural extension of my analyses.

4.4.2. *Practical limitations*

Earlier in this paper I described some of the practical problems OSS researchers have to face. In particular, I highlighted the difficulties posed by the heterogeneity, opacity, and volume of data available. While the OSS Browser was built to address these limitations, some progress remains to be made.

I believe the OSS Browser successfully provides a more readable overview of each participant's trajectory in the hybrid network. The graphical representations it produces certainly reduce the volume of text that would have to be manually coded and read using traditional qualitative research methods. Informal tests I conducted with other OSS researchers unfamiliar with the interface indicate that the Browser is reasonably easy to use, and that its representations are evocative and useful as starting points for deeper analyses. However it is important to note that, thus far, I have been the only

regular user of the tool. Conducting formal usability studies with its intended users (researchers and, perhaps, project members themselves) would certainly help reinforce the case for using "computer-aided ethnography" on a larger scale. It would also almost certainly result in many interesting suggestions for refining the tool's interface.

The amount of time and effort required to analyze and understand an OSS project with the Browser is perhaps the most important issue. This is partly by design: as I mentioned earlier it is important not to "untether" visual representations from the data used to generate them (Sack, 2000b). This would quickly lead to a formalist drift where the form of network patterns become more important than their content (Wellman, 1988). Using the Browser's interface, however, has not been the most time consuming part of my analyses – it is the development of the tool itself that required the most significant effort. Despite the availability of pre-existing software to build upon (Sack, 2001), we are far from a world where visualization interfaces can be quickly customized to reflect a particular theoretical orientation. Toolkits are currently being developed to mitigate this problem (e.g. Heer et al., 2005) and they might very well encourage further experiments in "computer-aided ethnography." For now however, the only way to really amortize the cost of developing a tool such as the OSS Browser would be to analyze a very large number of projects – and without modifying the interface along the way, which would be contrary to the approach I advocated. It is clear that a lot of work remains to be done to fluidly use computer interfaces as a form of ethnographic inscription.

5. Conclusions

Until now, the question of the dynamics of socialization and community reproduction in OSS projects had remained largely unanswered. Indeed, theoretical and practical limitations have constrained Open Source research in several ways:

- Open Source projects are *dynamic* entities, yet most of the current research has produced only static accounts of their activity.
- Open Source projects are *hybrid*, multi-sited environments composed of a network of human and material artifacts, yet these dimensions are often considered in isolation.
- The *massive amounts of research data* available tend to favor aggregate statistical analysis to the detriment of more qualitative, in-depth analysis of the activity in a project.
- Open Source productions are often *difficult to understand for non-developers*; accessing and processing some of this data (e.g. CVS records) requires technical knowledge.

In this paper, I have tried to go beyond these limitations both methodologically and analytically. Methodologically, I combined ethnography and the construction of software to visualize the hybrid, dynamic networks of OSS projects. This software facilitates the observations that are essential to ethnography, is in itself a form of ethnographic inscription, and extends a burgeoning tradition of "software-as-theory" (Dumit and Sack, 2000; Sack, 2000b, 2001) that addresses many of the difficulties an ethnographer has to face in the studies of online environments. Analytically, I have proposed two frameworks to help us understand socialization in Open Source projects: (1) as an individual learning process based on the construction of identities, and (2) as a political process involving the recruitment and transformation of human and material allies.

Based on this methodology, I analyzed in depth the activities of Open Source developers in a large, successful project (Python) over the course of a year and documented the various trajectories OSS participants can follow. It appeared clearly that being successfully integrated into an Open Source project is not as trivial as some would think: joining a project (and later evolving within it) requires one to go through a complex socialization process.

From an individual standpoint, successful contribution to an OSS project is much less about technical expertise than about the construction of identities. Despite the rhetoric surrounding Open Source, which basically argues that "anybody can contribute," it seems instead that only those few participants who have managed to define and present themselves as "software craftsmen" eventually reach the status of developer in a project. There are "ideal type" trajectories one can follow to reach this goal (Von Krogh et al., 2003), and in some ways these steps are reminiscent of the ones journeymen had to go trough in traditional apprenticeship learning (Lave and Wenger, 1991). Contributing to an open source project is as much a process of socialization as a show of technical expertise: many participants to Python are highly skilled developers, but those who follow the above trajectories apparently stand a better chance of becoming important actors. There is little evidence of explicit coaching or teaching from established experts. Instead, the participants have to discover what the norms of participation are by themselves.

Gaining influence inside a project such as Python, however, is not simply about creating and maintaining identities. Developing software is inherently a political process (Block, 1983; Divitini et al., 2003), and successful participants must also understand the nature of the game they have to play and then how to play it. Indeed, an Open Source project is essentially a hybrid network – a heteroclite assemblage of human and non-human actors, entangled in specific configurations that may vary over time. The skills of Python's software engineers reside in their ability to create the most stable

network of connections between these various pieces so that the project can withstand the test of time. These connections, however, are not visible to the naked eye: they are ''black-boxed,'' or hidden so that anyone who would like to challenge the current state of the project will have to uncover these relationships first. A very important first step for newcomers, therefore, is to progressively open this ''black-box.'' Based on his understanding of the project's hybrid network, a participant can then start to enroll allies (both human and material) to support his own efforts. Participants who do not take the time to understand the structure of the network and immediately try to modify it by inserting new material contributions are bound to fail: the very purpose of the network is to resist such brutal changes. To achieve his objectives, a participant has to learn how to subtly manipulate and transform the relationships between actants instead – hence my suggestion that ''OSS development is politics by other means'' (paraphrasing Latour, 1987a's famous ''science is politics by other means'').

The process outlined above must not be construed as limiting or negative. Politics often has negative connotations, yet it is clear here that the complex socialization mechanism I have described is essential to the good functioning of a project. It acts as a filter, ensuring that participants with the right set of skills and values are favored over a constant and potentially overwhelming stream of would-be contributors. Yet there would be ways to facilitate the process without diminishing this important function. I have proposed how some features of the OSS Browser could be repurposed to accelerate the integration of successful newcomers without bypassing the important stages and rites of passage characteristic of OSS socialization.

To conclude, it is important to mention some limitations that would have to be addressed in order to reinforce my analyses. First, time and resources constraints have allowed me to focus only on a single project as a case study. Python is reasonably representative of a class of large, successful OSS projects concerned with the development of computer languages (another example would be, for instance, Perl). But OSS projects vary across a number of dimensions: age, size (both in terms of the number of developers and the number of lines of code), type of software developed, etc. It would be interesting to analyze how much socialization in projects that differ along these dimensions resembles what I observed in Python – or not.

Second, my analyses have focused on the evolution of participants *within* a given project. This approach certainly leads to, I believe, interesting observations but it could also be that most of a participant's evolution occurs *across* projects. Rather than fighting their way in and go through the socialization process I have described, it could be that participants simply familiarize themselves with the world of Open Source by observing a first project, and then jump straight away to being developers in another one. To test this alternative hypothesis, we would need to consider community

reproduction at a more ecological level – a fascinating opportunity to develop another piece of software tailored to the kinds of observations one would have to make at this different level of granularity.

There are still many more questions to answer regarding participation in the Open Source movement – but hopefully, the results I am offering here will constitute a useful stepping-stone for further developments.

Acknowledgements

The author would like to express his gratitude to Professor Warren Sack (University of California, Santa Cruz) and Professor Peter Lyman (University of California, Berkeley), who both supervised much of this research. The author also gratefully acknowledges the contribution of three anonymous reviewers – their detailed and valuable comments were extremely helpful when refining early versions of this manuscript.

Notes

1. http://www.sourceforge.net is one of the main repositories of OSS projects.
2. To avoid cluttering the figure, I voluntarily selected a very short conversation.
3. Guido chose this name for his project because of his affection for the Monty Python show.
4. At this point it is useful to briefly discuss my approach to using a participant's real name versus disguising his identity. Throughout the rest of this document, I will be doing both. For those participants who can be considered to be "public figures", the real name will be used. It is widely known, for instance, that Guido Van Rossum is the founder and leader of Python – using a pseudonym to describe his activities would make little sense. When analyzing the activities of more peripheral participants, however, I have chosen to disguise their real identities. Indeed it would be far fetched to assume that, even though they are contributing to a public discussion, these participants are seeking any kind of notoriety – at least initially.
5. A complete description of these other individual trajectories is available in Ducheneaut (2003).
6. Fred is a pseudonym.
7. Space constraints prevent me from reproducing each email message in its entirety. In the remainder of this paper I will therefore present heavily edited versions of these messages, highlighting only the most relevant information.

References

Bernard, H.R. (ed.) (1998): *Handbook of Methods in Cultural Anthropology*. Walnut Creek, California: Alta Mira Press.

Bezroukov, N. (1999): Open Source Development as a Special Type of Academic Research. *First Monday* 4(10).

Block, R. (1983): *The Politics of Projects*. Yourdon Press.

Button, G. and W. Sharrock (1996): Project Work: The Organization of Collaborative Design and Development in Software Engineering. *Computer Supported Cooperative Work: The Journal of Collaborative computing*, vol. 5, no. 4, pp. 369–386.

Callon, M., J. Law and A. Rip (1986): *Mapping the Dynamics of Science and Technology: Sociology of Science in the Real World*. Houndmills, Basingstoke: Macmillan Press.

Capiluppi, A., P. Lago and M. Morisio (2003): Evidences in the Evolution of OS Projects through Change Log Analyses. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland OR, pp. 19–24.

Cherny, L. (1999): *Conversation and Community: Chat in a Virtual World*. Palo Alto, CA: CSLI Publications.

Csikszentmihalyi, M. (1993): Why We Need Things. In S. Lubar and W.D. Kingery (eds.): *History from Things: Essays on Material Culture*. London: Smithsonian institution press, pp. 20–29.

Divitini, M., L. Jaccheri, E. Monteiro and H. Traetteberg (2003): Open Source Process: No Place for Politics? In *Proceedings of the 3rd Workshop on Open Source Software Engineering*. Portland OR, pp. 39–44.

Ducheneaut, N. (2003): The Reproduction of Open Source Software Communities. Unpublished PhD dissertation. University of California, Berkeley.

Dumit, J. and W. Sack (2000): Artificial Participation: An Interview with Warren Sack. In G.E. Marcus (ed.): *Zeroing in on the Year 2000: The Final Edition (Late Editions, 8)*., Chicago: University of Chicago Press.

Edwards, K. (2001): Epistemic Communities, Situated Learning, and Open Source Software Development. In ''Epistemic Cultures and the Practice of Interdisciplinarity'' workshop (pp. 24). NTNU, Trondheim, June 11–12, 2001.

Emerson, R.M., R.I. Fretz and L.L. Shaw (1995): *Writing Ethnographic Fieldnotes*. Chicago, IL: The University of Chicago Press.

Erickson, T. (1999). Persistent Conversation: An Introduction. *Journal of Computer-Mediated Communication* 4(4) (http://www.ascusc.org/jcmc/vol4/issue4/ericksonintro.html).

Feller, J. and B. Fitzgerald (2002): *Understanding Open Source Software Development*. Addison-Wesley.

Fielding, R.T. (1999): Shared Leadership in the Apache Project. *Communications of the ACM* 42(4).

Fogel, K. (1999): *Open Source Development with CVS: Learn How to Work With Open Source Software*. The Coriolis Group.

Garfield, E. (1979): *Citation Indexing: Its Theory and Applications in Science, Technology and Humanities*. New York, NY: John Wiley.

German, D. and A. Mockus (2003): Automating the Measurement of Open Source Projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*. Portland OR, pp. 63–68.

Ghosh, R. and V.V. Prakash (2000): The Orbiten Free Software Survey. *First Monday* 5(7).

Gonzalez-Barahona, J.M., L. Lopez and G. Robles (2004): Community Structure of Modules in the Apache Project. In *Proceedings of the 4h International Workshop on Open Source Software Engineering*. Edinburgh Scotland, pp. 44–48.

Gordon, R.B. (1993): The Interpretation of Artifacts in the History of Technology. In S. Lubar and W.D. Kingery (eds.): *History from Things: Essays on Material Culture*. London: Smithsonian Institution Press, pp. 74–93.

Grinter, R.E., J. Herbsleb and P. Dewayne (1999): The Geography of Coordination: Dealing with Distance in R&D Work. In *Proceedings of the international ACM SIGGROUP Conference on Supporting Group Work*. New York: ACM, pp. 306–315.

Hars, A. and S. Ou (2000): Why is Open Source Viable? A Study of Intrinsic Motivation, Personal Needs and Future Returns. In M. Chung (ed.): *Proceedings of the 2000 Americas Conference on Information Systems*. Long Beach CA, pp. 486–490.

Heer, J., S.K. Card and J.A. Landay (2005): Prefuse: A Toolkit for Interactive Information Visualization. In *Proceedings of the Sigchi Conference on Human Factors in Computing*. New York: ACM, pp. 421–430.

Herbsleb, J., A. Mockus, T. Finholt and R.E. Grinter (2000): Distance, Dependencies, and Delay in a Global Collaboration. In *Proceedings of the ACM conference on computer supported cooperative work (CSCW 2000)*. New York: ACM.

Hine, C. (2000): *Virtual Ethnography*. Sage Publications.

Inkeles, A. (1969): Social Structure and Socialization. In D.A. Goslin (ed.): *Handbook of Socialization Theory and Research*. Chicago: Rand McNally, pp. 615–632.

Kelty, C.M. (2001): Free Software/Free Science. *First Monday* 6(12).

Kling, R., G. Kim and A. King (2003): A Bit More to IT: Scholarly Communication Forums as Socio-technical Interaction Networks. *Journal of the American Society for Information Science and Technology*, vol. 54, no. 1, pp. 47–67.

Kraft, P. (1977): *Programmers and Managers: The Routinization of Computer Programmers in the United States*. New York: Springer-Verlag.

Krishnamurthy, S. (2002): Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday* 7(6).

Latour, B. (1987a): *The Pasteurization of French Society, with Irreductions*. Cambridge, MA: Harvard Univeristy Press.

Latour, B. (1987b): *Science in Action: How to Follow Scientists and Engineers Through Society*. Cambridge, MA: Harvard University Press.

Latour, B. (1996): On Actor-Network Theory: A Few Clarifications. *Soziale Welt*, vol. 47, no. 4, pp. 369–381.

Latour, B. (1999): On Recalling Ant. In J. Law and J. Hassard (eds.): *Actor Network Theory and After*. Oxford: Blackwell, pp. 15–25.

Latour, B. (1999): *Pandora's Hope*. Cambridge, MA: Harvard University Press.

Lave, J. and E. Wenger (1991): *Situated Learning: Legitimate Peripheral Participation*. New York, NY: Cambridge University.

Law, J. (1999): After Ant: Complexity, Naming, Topology. In J. Law and J. Hassard (eds.): *Actor Network Theory and After*. Oxford: Blackwell, pp. 1–14.

Lerner, J. and J. Tirole (2002): Some Simple Economics of Open Source. *The Journal of Industrial Economics*, vol. L(2), 197–234.

Lyman, P. N. Wakeford (eds.) (1999): *Analyzing Virtual Societies: New Directions in Methodology*. Thousand Oaks: Sage.

Maas, W. (2004): Inside an Open Source Software Community: Epirical Analysis on Individual and Group Level. In *Proceedings of the 4th Workshop on Open Source Software Engineering*. Edinburgh Scotland, pp. 64–70.

Madey, G., V. Freeh and R. Tynan (2002): The Open Source Sofware Development Phenomenon: An Analysis based on Social Network Theory. In *Proceedings of the Americas Conference on Information Systems (AMCIS2002)*, Dallas TX, pp. 1806–1813.

Mahendran, D. (2002): Serpents and Primitives: An Ethnographic Excursion into an Open Source Community. Unpublished Masters thesis, University of California, Berkeley, Berkeley, CA.

Marcus, G.E. (1995): Ethnography in/of the World System: The Emergence of Multi-sited Ethnography. *Annual Review of Anthropology*, vol. 24, 95–117.

Matsushita, M., K. Sasaki, Y. Tahara, T. Ishikawa and K. Inoue (2003): Integrated Open-Source Software Development Activities Browser (CoxR). In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland OR, pp. 99–104.

Mockus, A., R.T. Fielding and J. Herbsleb (2000): A Case Study of Open Source Software Development: The Apache Server. In *Proceedings of the 22nd International Conference on Software Engineering*. Limerick, Ireland, pp. 263–272.

Moon, J.Y. and L. Sproull (2000): Essence of Distributed Work: The Case of the Linux Kernel. *First Monday* 5(11).

Nonnecke, B. and J. Preece (2003): Silent Participants: Getting to Know Lurkers Better. In D. Fisher and C. Lueg (eds), *From Usenet to Cowebs: Interacting with Social Information Spaces*, Springer Verlag.

Orr, J. (1990): Sharing Knowledge, Celebrating Identity: War Stories and Community Memory in a Service Culture. In D.S. Middleton and D. Edwards (eds.): *Collective Remembering: Memory in Society*., Beverly Hills, CA: Sage Publications.

Osterlie, T. (2004): In the Network: Distributed Control in Gentoo Linux. In *Proceedings of the 4th International Workshop on Open Source Software Engineering*. Edinburgh Scotland, pp. 76–81.

Potts, C. and L. Catledge (1996): Collaborative Conceptual Design: A Large Software Project Case Study. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, vol. 5, no. 4, pp. 415–445.

Python (2004). The Python Project's Web Site, available at: http://www.python.org.

Raymond, E.S. and B. Young (2001): *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates.

G. Robles-Martinez, J.M. Gonzalez-Barahona, J. Centeno-Gonzalez , V. Matellan-Oliveira and L. Rodero-Merino (2003): Studying the Evolution of Libre Software Projects using Publicly Available Data. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland OR, pp. 111–116.

Rutter, J. and G. Smith (2002): Ethnographic Presence in Nebulous Settings: A Case Study. Paper presented at the ESRC virtual methods seminar series, research relationships and online relationships, CRICT, Brunel University, 19 April 2002.

Sack, W. (2000a): Design for Very Large-scale Conversations. Unpublished Ph.D. thesis, MIT Media Laboratory, Cambridge, MA.

Sack, W. (2000b): Disourse Diagrams: Interface Design for very Large-scale Conversations. In *Proceedings of the 33rd Hawaii International Conference on System Sciences, Persistent Conversations Track*. Maui HI: IEEE Computer Society.

Sack, W. (2001): Conversation Map: An Interface for Very Large-Scale Conversations. *Journal of Management Information Systems*, vol. 17, no. 3, pp. 73–92.

Sack, W. and J. Dumit (1999): Very Large-scale Conversations and Illness-based Social Movements. In *Presented at the Conference Media in Transition*. Cambridge MA: MIT.

Seely Brown, J. and P. Duguid (1991): Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning, and Innovation. *Organization Science*, vol. 2, no. 1, pp. 40–57.

Shaikh, M. and T. Cornford (2004): Version Control Tools: A Collaborative Vehicle for Learning in F/OS. In *Proceedings of the 4th Workshop on Open Source Software Engineering*. Edinburgh Scotland, pp. 87–91.

Sim, S.E. and R.C. Holt (1998): The Ramp-up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize. In *Proceedings of the 20th International Conference on Software Engineering*. Kyoto Japan, pp. 361–370.

Smith, M.A. and A.T. Fiore (2001): Visualization Components for Persistent Conversations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Seattle WA NY: ACM Press, pp. 136–143.

Star, S.L. (1991): Power, Technologies and the Phenomenology of Convention: on being Allergic to Onions. In J. Law (ed.): *A Sociology of Monsters*. London: Routledge, pp. 26–56.

Star, S.L. (1995): *Ecologies of Knowledge: Work and Politics in Science and Technology*. State University of New York Press.

Teil, G. and B. Latour (1995): The Hume Machine: Can Association Networks do More than Formal Rules? *Stanford Humanities Review* vol. 4, no. 2, pp. 47–65.

Tuomi, I. (2001): Internet, Innovation, and Open Source: Actors in the Network. *First Monday* 6(1).

Turkle, S. (1997): *Life on the Screen: Identity in the Age of the Internet*. Touchstone Books.

Turner, V. (1969): *The Ritual Process: Structure and Anti-structure*. Chicago: Aldine Publishing Co.

Von Krogh, G., S. Spaeth and K. Lakhani (2003): Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. *Research Policy*, vol. 32, no. 7, pp. 1217–1241.

Von Hippel, E. (2002). Horizontal Innovation Networks: by and for Users (Working paper No. 4366-02). MIT.

Wall, L., T. Christiansen and J. Orwant (2000): *Programming Perl*. 3. San Francisco, CA: O'Reilly.

Weber, M. (1949): *The Methodology of the Social Sciences. (E. Schills & H. Finch, Trans.)*. New York: The Free Press.

Weber, S. (2000): *The Political Economy of Open Source Software (Working paper)*. Berkeley, CA: Berkeley Roundtable on the International Economy (BRIE).

Wellman, B. (1988): Structural Analysis: From Method and Metaphor to Theory and Substance. In B. Wellman and S.D. Berkowitz (eds.): *Social Structures: A Network Approach*. Cambridge: Cambridge University Press, pp. 19–61.

Yamauchi, Y., M. Yokozawa, T. Shinohara and T. Ishida (2000): Collaboration with Lean Media: How Open-source Software Succeeds. In *Proceeding of the ACM 2000 Conference on Computer Supported Cooperative work*. December 2-6, Philadelphia PA, pp. 329–338.

Zhang, W.and J. Storck (2001): Peripheral Members in Online Communities. In *Proceedings of AMCIS 2001 the Americas Conference on Information Systems*. Boston MA, p. 7.