

On the Uncertainty Principle and Social Constructivism: the case of Free and Open-Source Software

By Shay David, sd256@cornell.edu

Abstract

In this paper I propose and use an enhanced version of SCOT—the Social Construction of Technology—to look at the socio-technological development of free and open source (1969-2003). The paper analyzes the relevant social groups that attributed meanings to free operating systems and used their interpretive flexibilities when shaping them, the attempts of these groups to reach closure using both rhetorical and conceptual means, and the development of several technological frames. I use the historical case study of free and open source software in order to alert the reader to several methodological weaknesses in the original SCOT framework and to suggest and apply potential remedies. Specifically I, find that SCOT in its initial version is susceptible to what I call the ‘uncertainty principle of social constructivism’ which states that the social position and technological momentum of an artifact cannot be analytically recorded with precision at one and the same time. I find that SCOT does not properly account for the activities of individuals and corporations which are dominant in Capitalism, and that it lacks temporal dimensions and commitment to iteration that would enable us to properly understand the intricate workings of technology at moments of controversy. I suggest several practical solutions and new metaphors that enable us to overcome these challenges, while telling the story of an ongoing technological revolution in social constructivist terms.

Keywords

free software-history of, Linux-history of, open source-history of, Technology-social aspects, Technology-social construction of

Introduction – The Uncertainty Principle of Social Constructivism

I believe that the existence of the classical ‘path’ can be pregnantly formulated as follows:

The ‘path’ comes into existence only when we observe it.

Warner Heisenberg, 1927

Heisenberg’s uncertainty principle implies that atomic-scale physical particles can only be described with a limited degree of accuracy because momentum and position-in-space, which initially seem to be independent properties, are shown to be intricately connected by the quantum principles that underwrite physics. (Heisenberg 1927:174-5) When measuring the properties of, say, a photon the physicist has to choose which of these two properties she is interested in measuring with higher accuracy but she cannot perfectly record a particle’s momentum and position at one and the same time. This microscopic choice is comparable to macroscopic decisions a sport photographer has to make when working on a photo-finish image of a 100-yard dash. If she wants to remain true to the spirit of the event and give her audience a sense of the momentum of the sprinters she has to keep the shutter open for a while to allow a certain degree of fuzziness to show up in the image; this, of course, comes at the cost of losing the direct perception of who crossed the line first, which is the very idea that triggers the photo-finish concept to begin with. A corollary of the uncertainty principle discusses the implications for elementary particles when they are inspected separately from one another. The inability to precisely know the position and momentum simultaneously implies that the behavior of a single particle, its ‘path’ is unpredictable, or simply put—uncertain. Consequently the uncertainty principle has profound implications for our fundamental notions of causality and the determinism from the atomic level and up.

By analogy, we can say that historians of technology are faced with similar uncertainties when trying to understand how technology operates. They, therefore, willy-nilly must make the same type of analytical decisions regarding the study of technological systems’ properties as the physicist and photographer do in

their work, but sometimes they fail to admit it. As a response to histories of technology that treat technology as if it is deterministic, certain, and sure to follow some trajectory, some two decades ago a valuable framework called the Social Construction of Technology (SCOT) was proposed by Pinch and Bijker (1987) and was soon developed by them and others to explain the social construction of technologies and artifacts as varied as Bakelite, (Bijker 1987) Automobiles (Pinch and Kline 1996), computer users (Bardini and Horvath 1995) and the Internet (Kim and Watanabe 2002), to name just a few of the many works that were written in this spirit. SCOT represents a significant departure from earlier history of technology methodologies, in that according to it the historian of technology should approach technological innovation with a social constructivist mindset and reconstruct the heat of the moment of technological change instead of offering simplistic linear models of how technology develops. The main setback with SCOT, however, is that in its quest for analytic accuracy and flight from concepts like ‘trajectory’ and ‘momentum’ it resorts to capturing short-lived snapshots of socio-technological positions. It tries to contain in two dimensional sociograms processes which are systematic, procedural and at least four dimensional¹ in essence. SCOT works very well for simple technological artifacts, but what many social constructivists fail to acknowledge is that when looking at examples which are more complex than the socially constructed bicycle—SCOT’s mascot—we find that by describing the system discretely and not continually, and by focusing on social groups exclusively and not leaving room for firms and individuals, SCOT misses out on important economical drivers, social interactions, political forces, and power-plays that force technology to advance in certain ways and not others, and which understanding should be part-and-parcel of any reflective attempt that aims to recognize the social aspects of the development of a given technology. Nonetheless, I find SCOT to be an exceptionally apt method for an enquiry of technological innovation in a rapidly changing field, and I find its imperfections correctable.

¹ Counting time as a dimension.

In this short essay, therefore, I want to propose an improved version of SCOT that takes the uncertainty principle into consideration and to use this improved model to look at the social construction of free and open source software (F/OSS). F/OSS is a technology that is socio-political in its nature; more than a need for technological enhancements, its originators envisaged a need for acute social and political change. As such F/OSS offers a rare glimpse into its own social construction. In what follows I look at the relevant social groups, the core sets of people, the interpretive flexibilities these groups have (or have not) as they pertain to operating systems, and the problems of achieving closure in an open environment. This analysis will work towards two ends. First, it will allow me to tell the story of the free and open source software revolution in what I hope will be an innovative and revealing manner. Second, it will allow me to accentuate my critique of the original SCOT framework and clarify my suggestions for improvement to the analytical framework itself. I hope that my treatment of this case can generalize into an enhanced SCOT model that can serve the social study of technology in a robust and fruitful manner.

The genesis of free software

By the late 1970's and the early 1980's developments in the nascent computer industry were underscored by two trends—the rise of Unix workstations and the development of the first personal computers. Both trends were motivated by the desire to move away from central processing and into more distributed computing systems (Ceruzzi 1988:chapters 7,8). But just at the time that hardware prices were dropping and truly personal computers seemed within reach, a new obstacle appeared: Software. How should the development and distribution of software for these new, distributed computers be managed? How tightly should the software be coupled with the hardware? Who should pay for it? And, more important to our case, who should write code and who should be authorized to tinker with it?

The history of free software begins with these questions and with the story of an extraordinary individual called Richard Stallman, the last of the true hackers, as he refers to himself (Levy 1984:415). In

1985 Stallman founded the Free Software Foundation (FSF) in order to try and answer these burning questions, and in order to affect first-handedly the future of software in a very innovative and controversial manner. FSF's mission is to promote computer users' right to use, study, copy, modify, and redistribute computer programs, and the way to achieve this mission, Stallman believed, is by writing, and giving away quality software, with its source code—for free.

Following the SCOT framework, in order to see how this whole story started, and how eventually the answers to these questions were negotiated, we should identify the relevant social groups and see what problems software posed for them. A relevant social group becomes relevant when the artifact at hand has some meaning to that group. A 'problem' is a 'problem' only inasmuch as there is a social group that perceives it as such (Pinch and Bijker 1987:30). Focusing on the problems an artifact creates is supposedly useful to the analysis of technology since it highlights the controversies and points of contention among the groups. It also highlights the degree to which different groups have interpretive flexibility regarding the actual meaning of an artifact². Bridging these competing interpretations, according to SCOT, is the process by which a technology reaches a state of stabilization and closure.

At the time of the FSF's founding we can identify at least two existing relevant social groups that attributed meaning to the concept of computer operating systems: computer software manufacturers and academic computer science research centers. Stallman had intimate acquaintance with both of these groups. He graduated with a physics degree from Harvard (Magna Cum Laude) and later worked for MIT's AI (artificial intelligence) lab where he and his hacker peers worked on early computer time-sharing systems. This experience and acquaintance with state of the art computers developed by the Digital Equipment

² Note that here I extend the notion of artifact to include software in general, and operating system software specifically. I think this extension is warranted although it does bring up burning questions regarding the nature of digital artifacts and their place in an information society (specifically it calls for an investigation into the nonrivalrous nature of software and its ability to be duplicated which is not exhibited by other artifacts), unfortunately, these further questions are beyond the scope of this paper.

Corporation (DEC) illustrated to Stallman just how powerful computers can be when put in the hands of individuals, without the mediation of an 'operator' who had to feed the computer with instructions that were prepared beforehand (like in the older IBM mainframe machines).

Up until that period, most software, by default, was free. It wasn't really free, but its development cost was amortized in the price the hardware manufacturers charged for setting up their systems that included software and hardware. Computers were expensive, programs were machine specific and institutionally software wasn't really separate from hardware. When an institution bought a computer it actually paid for a bundle of hardware, software, and set-up and maintenance services. But operating systems like Unix and personal computers were about to change all this. In places like MIT's AI lab, AT&T Bell labs, Stanford, Berkeley, and Xerox PARC people were starting to write portable software that would be independent of a specific machine and that could network together cheap hardware and support many processes and many users in parallel. When these concepts started to mature, a score of private firms were spun-off off these research centers in order to financially exploit the new market which was being created.

Three such software firms had significant influence on Stallman's work. First there was AT&T's Bell Labs who developed the Unix operating system. Then there were LMI (Lisp Machines Inc.) and Symbolics Corporation, both of which were start-up firms that were monetizing on work done at MIT and other academic centers and were trying to commercialize computer systems based on cutting edge research results. In their quest for profit, however, all these companies had violated the hackers' longstanding principle of information sharing that promoted the invention of said software to begin with. In addition, in an attempt to recruit the crème-of-the-crop Symbolics and LMI hired away most of Stallman's AI Lab peers. (Levy 1984). Moreover, the two companies were not collaborating with one another and used the AI lab as an extended battleground in their wars. The main meaning that software has attained in the minds of the young managers of these software companies was as a money-making tool that should be protected by law and business strategy. In his history of hackerism Steven Levy quotes Russ Noftsker, the president of Symbolics: "We

develop a program or an advancement to our operating system and make it work. [We give it to MIT] and then Stallman ...reimplements it. He calls it reverse engineering. We call it theft of trade secrets.” (Levy 1984:426).

And indeed, Stallman, who by that time had developed an extreme version of a hacker ethic and didn't want to work for a commercial for-profit entity, was caught in the crossfire (Levy 1984:422). He was dismayed by the commercialization process, and by the ease in which the principles of knowledge sharing that were predominant in his hacker environment were compromised by license agreements and copy protections. His reaction was to eventually quit his job at the AI lab and start the FSF to which he devoted his life ever since. In an early manifesto Stallman writes:

The Free Software Foundation is dedicated to eliminating restrictions on copying, redistribution, understanding and modification of software. The word "free" in our name does not refer to price; it refers to freedom. First, the freedom to copy a program and redistribute it to your neighbors, so that they can use it as well as you. Second, the freedom to change a program, so that you can control it instead of it controlling you; for this, the source code must be made available to you. The Foundation works to give you these freedoms by developing free compatible replacements for proprietary software. Specifically, we are putting together a complete, integrated software system 'GNU' that is upward-compatible with Unix. When it is released, everyone will be permitted to copy it and distribute it to others; in addition, it will be distributed with source code, so you will be able to learn about operating systems by reading it, to port it to your own machine, to improve it, and to exchange the changes with others. (GNU Bulletin 1987)

Revealingly, as a name for his operating Stallman chose the recursive acronym "GNU" which stands for "GNU's Not Unix." This highlights his connection to the other software group, AT&T. Stallman wanted his innovative system to be 'not-Unix' because Unix represented for him lost-freedom in software. He understood, however, that if he wants it to succeed he needs to have (a) people who use his new software and (b) some existing system as scaffolding during the time he builds it. To gain these advantages, GNU had to be designed to be fully compatible with some other operating system, which ironically was not other than Unix itself. Being Unix-like meant that programs written for Unix would run on GNU and vice-versa. This, of course, begs the questions of exactly how is GNU "not Unix" and "Unix-compatible" at the same time, and why to collaborate so closely with an enemy:

Why a Unix-Like System? It is necessary to be compatible with some widely used system to give our system an immediate base of trained users who could switch to it easily and an immediate base of application software that can run on it. (Eventually we will provide free replacements for proprietary application software as well, but that is some years in the future.) We chose Unix because it is a fairly clean design which is already known to be portable, yet whose popularity is still rising. The disadvantages of Unix seem to be things we can fix without removing what is good in Unix (Ibid)

As soon as Stallman felt he had a grasp on the technical issues involved in producing GNU the FSF took a step beyond software development and focused on championing free software and open licensing as a form of social activism. This transition is not surprising if we remember that the main perceived problem was social and not technical. In addition, in a brilliant legal maneuver the FSF turned the copyright system against itself and created the GNU General Public License (GPL) as an alternative to traditional copyrighting. The process of publishing software under the GPL, which became known as ‘copylefting’, ensures that the software will always remain free, and avoids potential threats of delayed copyrighting by interested parties that might have occurred had the software been simply released in the public domain. According to its architects’ logic, naming this hack³ ‘copyleft’ is very simple: “Proprietary software developers use copyright to take away the users’ freedom; we use copyright to guarantee their freedom. That’s why we reverse the name, changing ‘copyright’ into ‘copyleft’” (Sarai Reader 2001:181). In his essay *Hau to do things with words*⁴, Christopher Kelty rightly finds that free software is in its very nature a critique of existing laws, contracts, and business practices, and that it has the potential to explicitly change the “political-economic structure of society” (Kelty 2001:3).

In copylefting what is reversed is the meaning of copyright protection which through copylefting is used not to conceal and congeal a program’s source-code but rather to divulge it and encourage future modifications. In other words, copylefting ensures that future users of the system would have not only

³ A hack is an occasion in which a system is turned against itself.

⁴ Hau is the Maori name of the wind-god. In the context of gift-cultures *hau* can be loosely translated as the ‘spirit of the gift’. The hau demands that the gift be returned to its owner. (Mauss 1990).

conceptual but also legal interpretive flexibility. The users' right to hack and tinker is ensured by law, and this is a legal cure to a perceived social problem and not simply a techno-legal modification. As Stallman puts it:

I'm trying to change the way people approach knowledge and information in general. I think that to try to own knowledge, to try to control whether people are allowed to use it, or to try to stop other people from sharing it, is sabotage. It is an activity that benefits the person that does it at the cost of impoverishing all of society. ... The principle of capitalism is the idea that people manage to make money by producing things and thereby are encouraged to do what is useful, automatically, so to speak. But that doesn't work when it comes to owning knowledge. They are encouraged to do not really what's useful, and what really is useful is not encouraged (Stallman 1986)

In order to correct Capitalism's paradox and achieve its grand goals, the FSF had to use the existing copyright law and create the GPL as a binding license agreement. Stallman describes the motivation and rationale behind the GPL: "if a program has an owner, this very much affects what it is, and what you can do with a copy if you buy one. The difference is not just a matter of money. The system of owners of software encourages software owners to produce something – but not what society really needs." (2001). Stallman knows better; he asks and answers: "What does society need? It needs information that is truly available to its citizens... Society also needs freedom... and, above all, society needs to encourage the spirit of voluntary cooperation in its citizens." (Ibid). Note Stallman's choice of words: it partially explains the connection some people made between his views and socialism. As we will see later, this language constituted part of the problem free software posed to the software establishment and was later challenged, and critically changed. However, when I interviewed him Stallman strictly denied any socialist connection:

The idea that cooperation is an ethical imperative and that society should encourage it is much older than Marx—the world's major religions have been promoting these views for millennia. It makes no sense, therefore, to give Communism credit for them. It is usually enemies of the FSF that call them "communists", perhaps because they find it easier to criticize communism than FSF's actual views. This practice is known as 'red baiting' (Stallman 2002)

Socialism aside, Stallman felt that no other group ever bothered to build a fully functional operating system, and that any early attempts to build such a system amounted to mere distribution of what was out there. His perception, though, was not accurate. The FSF was not operating in a social void; at least one other relevant

social group—academic computer science research centers—besides the FSF had long wanted to distribute a free operating system together with its respective source code, although they wanted to do so for different reasons.

The most notable attempt other than Stallman's at building such a free operating system was concentrated at Berkeley in the late '70s. Forking from early versions of AT&T's commercial versions of Unix, Berkeley's team released several versions of what came to be known as the 'Berkeley Software Distribution' or simply BSD (McKusick 1999:33). For Berkeley, closed operating systems constituted a different problem than for Stallman. Berkeley's BSD people were not interested in changing the way people treat information, rather, they wanted to maintain the level of cooperation that was standard for academic research in general, and to continue to do their cutting edge research on Unix, which AT&T's was trying to commercialize particularly:

With the commercialization of Unix, the researchers at Bell Laboratories [a subsidiary of AT&T] were no longer able to act as a clearing-house for the ongoing Unix research. As the research community continued to modify the Unix system, it found that it needed an organization that could produce research releases. Because of its early involvement in Unix and its history of releasing Unix-based tools, Berkeley quickly stepped into the role previously provided by the labs (Ibid:34).

Besides its desire to maintain a high degree of academic collaboration Berkeley's researchers had another problem with a closed and commercial version of Unix – it was too expensive for wide use. To understand this problem we have to look at the workings of one of BSD's powerful allies – the US Department of Defense (DOD) which had a significant influence on the way BSD was developed. To a large degree BSD attained its fame when Berkeley was contracted by the DOD's Advanced Research Project Agency (DARPA) to build components of the operating system for the nascent ARPANET, of which Berkeley was an important node. The main reason that DARPA chose BSD was because it was proven to be portable⁵ and this was

⁵ Portable in this context means that the software was not machine specific but, rather, could run on different types of computers.

essential to DARPA which wanted full interoperability on its network but didn't want to commit itself to a single hardware manufacturer. This requirement for interoperability was the main impetus to the development of a set of communication protocols that later became the foundation of the Internet and that, more importantly to our story, however, was a trigger to a rewrite of a large portion of the code in BSD. Up until the late '80s BSD was distributed with the source code on the condition that the receiver would first get (and pay for) a full license from AT&T. The perceived cost problem then was triggered by DARPA funded development of networking which opened up possibilities for connecting cheaper machines to the network at nodes for which the cost of the AT&T licenses would be prohibitive. Acting on this alarming signal from the new 'market' Berkeley went ahead and started a process that took a decade and involved litigation but by which end the parts of the BSD code that were inherited from AT&T were re-written, and BSD became a truly free operating system. (Ibid:40).

By that time, however, Stallman's GNU and the derivative works were way ahead. Because BSD was *not* free from its AT&T lineage early enough, in the in the mid-80's the FSF was concerned with building all the pieces of the first truly free operating system puzzle and then putting the puzzle together—from scratch. Stallman looked at both AT&T's commercial Unix version and the entangled AT&T/BSD as a pictorial reference of how the puzzle should look like when it's fully assembled. One important difference between Stallman's GNU and Berkeley's BSD that we should note is that while BSD—the Berkeley Software Distribution—as its name suggests was focused on distribution, GNU shifted the focus from distribution to software development:

In fact, our primary purpose is this software development effort; distribution is just an adjunct which also brings in some money. We think that the users will do most of the distribution on their own, without needing or wanting our help. (Stallman 1987)

But beyond this tactical difference there was a much more profound difference in philosophy that made the FSF's effort even more remarkable. Nikolai Bezroukov offers an excellent analysis of the different social regimes that resulted in different software licensing schemes for BSD and for GNU. He finds that the

BSD license ... can be considered as a codification of a venerable academic tradition. Other works can be used freely by a researcher as long as proper credit was given for the original work. The history of science is a proof that this idea of "proper credit" is a very important and efficient social mechanism. But GPL's advocates a completely different approach ... GPL explicitly promotes the "right of the strongest" or "the law of the jungle". The key feature of GPL is protection of the right to redistribute both original work and create and distribute derived works as long as the license is preserved ... In other words by promoting the equality of all users GPL implicitly denies the original author IP [intellectual property] rights. Notice that there is no attempt in the GPL to differentiate "users" into different classes. ... GPL implicitly denies the original author IP rights because of its explicit message "equality of all users at any cost"

To summarize what we have seen so far: the FSF was a self-defined relevant social group. Its interest was clear: to change the way people use knowledge and information generally and to revolutionize the way software is written and distributed particularly, in short—to set software free. The FSF saw software as a tool in the creation of human knowledge and not as knowledge in its own right for which an author should receive intellectual property rights. This view was in direct opposition to existing software ventures' interpretations and interests (as we will see more clearly in the next sections) which perceived software development and distribution as a way to make money, and wanted to maintain the fragile status quo in which a select few write the software and many paying customers use it. Motivated by a deep-rooted tradition of knowledge sharing, the academic research centers shared Stallman's vision and interpretation of what software could and should be, but at the same time the academia needed to maintain a set of institutional and legal commitments to commercial and government entities and to their own longstanding honor to the concepts of authorship and expertise. This was more than Stallman's social-innovation-at-all-cost philosophy permitted.

After specifying the relevant social groups and understanding their flexible interpretation of the technology at hand a simple SCOT analysis would now proceed to describe and evaluate the next cycle of development. But, as noted above, I want to alert the reader to some methodological problems and propose a potential cure. Although Pinch and Bijker instruct us that "aspects such as power or economic strength enter the description, when relevant" and that "we need to have a detailed description of the relevant social groups in order to better define the function of the artifact with respect to each group" (1984:34) a key problem with portraying all the relevant social groups on the same playing field remains. By drawing a simple sociogram

that depicts the social groups uniformly and the problems an artifact poses for each group discretely, we risk losing the uneven power-relations that exist among the groups in the first place. Consider for instance a headcount comparison relevant to our case: AT&T was perhaps the largest company that operated in the US during the early '80s. In 1985 AT&T employed over 413,000 people (Harmon and Seeker 1999). At the same time Berkeley's was one of the most praiseworthy computer science research centers in the nation; its research groups included not only top-of-the-line professors but also a constant supply of motivated graduate students. Symbolics and LMI were investor-backed corporations, which hired away dozens of the brightest minds from places like MIT. But the FSF (at least in its early years) was a one man show. It is important that we do not lose this perspective when we socially reconstruct Stallman's feat. Basic SCOT does not allow individuals to be counted for as much as they should be. Even without being a relevant social group an influential individual can make a big difference.

In addition, we should note that the uncertainty principle of social constructivism is at play here. On one hand we cannot appreciate Stallman's endeavor without historical knowledge of its later success (without such knowledge how can one social visionary be counted as more than a fly that, at most, irritates the raging software-industry bull?), but on the other hand, the success of the GNU operating system is exactly what needs to be explained (in Pinch and Bijker's words "it should be the explanandum, not the explanans" (1987: 24)). How, then, can we escape this Catch-22? By drawing a conditional sociogram. In 1985 the FSF is hardly a relevant social group for the development process of a revolutionary operating system. It should merely be a faint 'blip' on our social constructivist radar. And there probably should be other such 'blips' on the screen, dots of which only historical hindsight could tell whether they are a significant signal or mere noise. Only by iterating our analysis at short historical intervals can we distinguish between the decoys and the real explanatory targets we need to shoot down.

The rise of Linux

The invention of Linux in 1991 by Linus Torvalds was the next significant event in GNU's development. Stallman was working for seven years already on his prophetic operating system but while he was splitting his time between code-writing and social advocacy, GNU was advancing slowly and no definite target date for its completion seemed in sight. Because he modeled his system after Unix, Stallman could start his project by building essential but auxiliary tools like a compiler and a text editor (the famous EMACS) and rely on existing components in the interim. He left the toughest task, the building of the operating system's core (or kernel) to the end, and was working at it when Torvalds invented Linux. The Linux Kernel was the last component necessary to make Stallman's GNU operating system fully functional, and fitted it like a hand to a glove. (Moody 2001: chapters 2,3).

To understand this series of events while bearing in mind the caveat of 'blips' on the radar, we can find in Torvalds's story a new social group that with time became more and more relevant: computer science teaching centers. The rise of new operating systems like Unix (and later Microsoft's DOS) and the proliferation of computers that by the mid '70s became commonplace equipment in academia and businesses alike called for the establishment of computer science departments in many major research universities. A new and important task of those centers, like we would expect to find in every other established discipline, was the education of a younger generation of computer scientists. In our terms, the problem this group had with regards to operating systems was not a specific research problem but, rather, the complexity and the difficulties involved in teaching basic operating systems' concepts to young untrained students. For this reason during the '70s Unix was the choice in many teaching centers; its openness made it very attractive as an educational platform⁶. In 1979, ten years after Unix was invented, however, AT&T shipped Unix version

⁶ Unix's simplicity was, of course, relative. I remember spending many frustrating hours in the computer labs during my undergraduate computer science training trying to understand how Unix really worked. The only relief was a funny poster on the wall that read: "Unix is a very user friendly system, it is just very selective on who it makes friends with."

7, which contained a warning that the source code couldn't be shared with students anymore. Moody finds that "Version 7 represented the symbolic closing of Unix inside the black box of proprietary software—a sad end to what had long been the ultimate student hacker's system" (Moody 2001:33). Without the source-code teaching an operating system is, of course, much more complicated and slow—and the teaching community had to react.

As we have seen in Stallman's case and in Berkeley's case before, the ultimate hacker solution to a black-boxing crisis is to rewrite the software from scratch and disengage with the set of commitments (legal, ethical, institutional) associated with the former code tree. Such a rewrite allows the new developer to focus on his own problem-solving process which is based on his interpretive flexibility of what the software is and what problem it solves (or creates). Andrew Tanenbaum, a professor from the Free University in Amsterdam stepped up to the challenge. He appreciated teaching Unix, and was very disappointed when AT&T made the code unavailable to his students. He thought of ways to bypass the problem but reached the conclusion that: "[the only solution] was to write an operating system on my own that would be system-call compatible with Unix...but which was my own code, not using any AT&T code at all." (Moody 2001:33) By now this line of hacker reasoning sounds familiar to us.

Tanenbaum went ahead and from 1979 to 1984 worked on building his version of a Unix-compatible system called Minix. When Minix was released and Stallman, who was scouting for a kernel for GNU, heard of it, he wanted to join forces with Tanenbaum and use Minix's kernel for GNU. But Tanenbaum, we should remember, was not motivated by social ideals and was trying to solve a completely different problem. He was trying to create a teaching-tool for computer science schools that would be well documented and easy to use, and he thought that the whole idea of free software is "silly" and that it makes no economic sense (Moody 2001:22). Instead, he was willing to exchange the kernel for Stallman's auxiliary applications and utilities, and build a commercial system out of the GNU/Minix mix, and he generously proposed to Stallman to share

future revenues that would result from sales of a joint system. Stallman, of course, rejected the despicable offer to sell 'free software'. He continued the work on his own GNU kernel.

Seven years later Stallman's effort had made GNU famous but it still lacked a kernel while Minix was what it was designed to be—a stable, simple to use, and well documented operating system that was classic for educational purposes. It is unsurprising, then, that Torvalds, who was at the time a young computer science undergraduate student at the University of Helsinki, installed Minix on his new machine as he was learning to program and was interested in operating systems. Soon enough, though, Minix was too 'small' for Torvalds. The flip-side of designing a straightforward system is that its simplicity often comes from limited functionality, and that apparently was the case for Minix. Tanenbaum consistently opposed complicating his system or adding new features to it. For his needs, the system was stable. It had reached closure on his terms, and there was no need to change it. Torvalds started playing around with improvements to Minix but after a short while understood that using Minix as scaffolding, aided by Tanenbaum's extensive documentation, he could actually build an alternative system. On August 25th, 1991 he posted a message to the Minix newsgroup.

From:torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)

Newsgroup: comp.os.minix

Subject: What would you like to see most in minix?

Summary: small poll for my new operating system

Message-ID: 1991Aug25, 20578.9541@klaava.Helsinki.FI

Date: 25 Aug 91 20:57:08 GMT

Organization: University of Helsinki.

Hello everybody out there using minix- I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix; as my OS resembles it somewhat (same physical layout of the file-system due to practical reasons) among other things. I've currently ported bash (1.08) and gcc (1.40), and things seem to work. This implies that i'll get something practical within a few months, and I'd like to know what features most people want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus Torvalds torvalds@kruuna.helsinki.fi

(Torvalds posting to comp.os.minix. Quoted in Moody 2001:42)

There are a few mutually supporting observations to make on this remarkable message. First, the concept of newsgroups is in itself revealing in the analysis of relevant social groups. A newsgroup is a self-organizing community that is formed around a common technical interest. By definition, people that participate in newsgroups have a vested interest in the subject that is being discussed and which has a set of meanings for them. Newsgroups complicate the concept of a 'core set'. In the traditional sociology of scientific knowledge a 'core set' refers to the group of "scientists most intimately involved in a controversial research topic" (Pinch and Bijker 1987:27). Studying a 'core set' is an empirical shortcut that overcomes practical problems that arise when trying to understand how scientific knowledge is constructed over time and across geographical boundaries. Focusing on such a limited but influential groups of people also allows the sociologist to monitor the closure that is attained when scientific controversies are settled. By extension, SCOT promotes the study of core sets of people which are most involved in an artifact's technological development. But as this newsgroup posting clearly demonstrates, in cyberspace the boundaries between the core developers and their constituency is easily blurred.

So far we have focused our discussion on several individuals, who represented for us the relevant social groups. Torvalds's call to the community ("I'd like to know what features most people want") was surely not the first of its kind, it took for granted a hacker ethic of cooperation and knowledge sharing. It is emblematic of calls for the relevant social group to voice its own interpretive flexibility without mediation. As this example and later developments in Linux clearly show, in software related analysis we should treat core sets with suspicion because the community is so empowered that it is not ever clear who is part of the core set and whether a significant part of the controversy does not in fact take place elsewhere (i.e. in a private email exchange, or on another newsgroup.)

A Second point to note is that Torvalds defines his relationship to the existing operating systems directly. His system is "somewhat like Minix", "won't be big and professional like gnu" and is clearly Unix-compatible (since he reports he 'ported' *gcc* and *bash* which were Unix-compatible GNU tools). The artifact

being negotiated though is Minix, and this is clear by the fact that the posting took place on the Minix newsgroup. Not long after this posting, and after a fight with Tanenbaum on the ‘right’ way to build operating system kernels, Torvalds started a new newsgroup which was dedicated to Linux⁷. The departure from the Minix newsgroup was more than symbolic. It allowed a real social group to align behind Linus and Linux.

A third and connected point to note should be familiar by now. We see yet again a case in which a significant software development effort is replicated because the different groups’ interpretive flexibility cannot be reconciled. Torvalds chose to start a new operating system because he found it easier to re-develop than to negotiate with Tanenbaum the meaning of Minix. Linux was an operating system that initially solved Linus Torvalds’s problems, not anyone else’s.

From these three points we learn one important lesson: the focus on controversial stages of an artifact’s technological development complicates the identification of the relevant social groups. As this example shows, and as we have seen in Stallman’s case, often the direction of future development is decided *before* the relevant social group is constituted. Influential individuals like Stallman, Tanenbaum or Torvalds can become a magnet for a social group which is formed only at a second stage. We cannot hope for a simple SCOT analysis of the interest of the social groups at these controversial points simply because the groups don’t exist yet. In addition, we have the uncertainty principle working again. Linus Torvalds is an undergraduate, while Stallman is a Harvard alum and by that time a very famous hacker, and Tanenbaum a distinguished professor and creator of a highly regarded operating system. How can we treat them equally? We can’t. We can only add Torvalds and the community he created as potentially influential social factors, and look at later historical events to see if this potential was actualized.

The next steps in the development of Linux were crucial to its later proliferation. A group of talented hackers joined Torvalds’s efforts and helped him to add virtual memory, a graphic user interface and

⁷ ‘Linux’ was Torvalds’s username on the university computer system, and the new operating system inherited its name from its creator

networking capabilities to the budding software. Torvalds decided to release the resulting system under the GNU GPL. He was concerned less with social advocacy and more with ensuring that future modifications will always be available for incorporation into the main development branch (a property that the sophisticated GPL enforces) (Moody 2001:78). With these new features, Torvalds's original kernel, and the valuable tools that were ported from GNU, the GNU/Linux bundle became an exceptionally stable, accessible, and free alternative to the alternative operating systems that were available at the time. With the completion of Linux, Stallman's decade-old vision of a fully functional, lively, and free operating system that was not driven by prospects of future economic profit was finally realized.

In SCOT terms, the rise of Linux highlights its constitution as a technological frame. The term 'technological frame' is introduced by Bijker as "a frame with respect to technology, rather than as the technologist's frame" (1987:172) and as an alternative to earlier concepts such as 'technological style', 'technological tradition', and 'technological paradigm' which Bijker finds to be too centered around engineering communities and at a level which is too high-up for his detailed explanatory needs. Bijker argues that:

[The] meaning attributed to an artifact by members of a social group play a crucial role in my description of technological development. The technological frame of that social group structures this attribution of meaning by providing, as it were, a grammar for it. This grammar is used in the interaction of members of that social group, thus resulting in a shared meaning attribution. ... A technological frame is built up when interactions 'around' an artifact start and continue.... On the one hand, a technological frame can be used to explain how the social environment structures an artifact's design. ... On the other hand, a technological frame indicates how existing technology structures the social environment (Bijker 1987:172)

In our case, then, the two earlier technological frames that of Minix and that of GNU clearly show how a technological frame works. Minix structured its own social environment, the group of academics who were interested in operating systems. Minix was structured by that environment too: its simplicity was a direct result of the environment's educational, simplistic needs. Minix was never designed to perform any real world tasks, and there was no need to continue its development. GNU, in parallel, had its social aspirations built into

its design in the form of the GPL and free distribution practices. GNU was shaped by the environment in a negative way: its design was meant for it to *not* be what other systems were (closed, expensive, buggy). Specifically GNU was designed to be, as its name suggests, not Unix. Linus Torvalds was initially a member of both of these technological frames. Bijker finds that:

A technological frame structures the interaction of members of a social group. But it can never do so completely: first, because different actors will have different degrees of inclusion in the frame (actors with a high inclusion interacting more in terms of that technological frame and actors with a low inclusion to lesser extent), and, second, because all actors will in principle, be members of more than one technological frame (Bijker 1987:173)

This observation is useful when we try to explain the four cases we observed already (GNU, BSD, Minix, and Linux) in which the hackers involved showed a tendency to re-write their free operating system from scratch instead of changing an existing system. In SCOT's terms, these are all cases in which an actor who had a low level of inclusion in an existing technological frame chose to constitute a new technological frame in which he will have a higher level of inclusion and a higher degree of power. This power would, in turn, translate to better control on both the technology and the social environment upon which it rests.

First attempt at closure: a rhetoric move from Free Software to Open Source

During the following five years (1992-1997) Stallman and Torvalds worked on getting GNU/Linux off the ground and were aided by the rise of the Internet and the fact that computer networking was becoming readily available for home and small office use. Since GNU/Linux, from its early versions and on, included all the development tools necessary to continue and develop applications that extend the operating system's basic functionality, anyone that had an Internet connection could easily become a part of a growing community of GNU/Linux users and developers.

And millions of people did, including new relevant social groups like the group of hackers piloted by Eric Raymond that decided to create a new framework for open software and information-sharing that would combine the values of free software with the Protestant virtues of capitalistic business. The new group

believed that free software has many more merits to offer than just social ones and that it should be endorsed by businesses as a superior development methodology. For this reason, Raymond did not want to be associated with anything that might be considered socialist. Based on the achievements of the FSF, but with an emphasis on methodologies rather than on social causes, and despite Stallman's vehement rejections, Raymond founded the Open Source Initiative in 1998. The name that was chosen for this organization is important, and we will return to it shortly. Raymond—by all accounts a verbal figure—describes the differences in philosophy between his organization and the FSF:

Licenses are important, but they are not the heart of the matter. The heart of the matter is that a bunch of volunteers, with asynchronous access to openly available source code can build a highly complex piece of software in the absence of any explicit corporate management. The change at issue is a change in the forms of life and work of software programmers, along with a change in the process of technical innovation. (Raymond quoted in Kelty 2001:19).

In other words, according to OSI's interpretive flexibility 'free software' means something completely different than what it means to the FSF. Since the vision of building a fully free operating system was shown by the FSF to be realizable, there was now time and attention to focus on the programmers' life and work habits, and an opportunity to reflect on the process of technological innovation itself, and what it means for businesses. The GNU/Linux experience showed that the open software development model produces better software than the traditional closed model in which only a small group of programmers can see the source and everybody else must blindly use an opaque block of bits. This process is described in the Open Source organization's homepage in evolutionary terms:

The basic idea behind open source is very simple: when programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, and people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing (Open Source Initiative 2002).

Raymond was fascinated by GNU/Linux's success and even before he instigated the OSI, he published a very influential paper called *The Cathedral and the Bazaar* (1997) in which he anatomizes a project called 'fetchmail', "that was run as a deliberate test of some surprising theories about software engineering

suggested by the history of Linux”. In the essay he looks at the ways software is written, debugged and used “in terms of two fundamentally different development styles, the ‘cathedral’ model of most of the commercial world versus the ‘bazaar’ model of the Linux world” and shows that “these models derive from opposing assumptions about the nature of the software-debugging task.”(Ibid) This manifesto was aimed at promoting a radical change in the way people design, build, and use software yet Raymond never mentions any social causes. He never aims to change the way people treat knowledge in general, and he is not so much concerned with licensing schemes that would ensure the future freedom of the software. In his pragmatic approach Linux-style development should be seen as better simply because it allows faster, more reliable development. For these same reasons it should be adopted by businesses worldwide.

In an innocent-looking footnote on a later version of the *Cathedral and Bazaar* paper we find a remnant of the very bitter controversy. At the very bottom of the paper Raymond writes “I changed ‘free software’ to ‘open source’ February 9 1998.” Raymond’s choice of a new metaphor to describe an existing activity was, of course, not accidental. To be successful in achieving his new goals and to differentiate his organization from the original free software project, Raymond needed to make his intentions clear. The FSF prompts people to “think of ‘free’ as in ‘free speech,’ not as in ‘free beer’” (GNU Homepage 2003) but Raymond whose project is to convince the business-world to accept the tenets of free software development understood that businesspeople cannot distinguish between the double meanings of the word ‘free’ which in English entails the meaning of both ‘libre’ and ‘gratis.’ Since the business community became a new relevant social group in Raymond’s eyes, and since technically, the free software technology was ready for stabilization, Raymond decided to eliminate the conceptual problem by a rhetorical move. He changed ‘free’ to ‘open’ and ‘software’ to ‘source’ to emphasize that the new system is compatible with old-world business values.

In our terms, establishing the ‘open source’ metaphor was plainly Raymond’s mindful attempt to reach what SCOT calls ‘rhetoric closure’ which Pinch and Bijker (1987:44) describe as follows:

Closure in technology involves the stabilization of an artifact and the “disappearance” of problems. To close a technological “controversy,” one need not solve the problems in the common sense of that word. The key point is whether the relevant social groups see the problem as being solved. In technology, advertising can play an important role in shaping the meaning that a social group gives to an artifact.

In other words, the OSI tried to reach closure by eliminating the problem in the eyes of the group that they perceived as socially relevant: the business people. The OSI was motivated by a recent announcement by Netscape (the company behind Navigator, the first widely used internet browser) which decided to release the source code of its browser software to the public, after being persuaded by Raymond that it could help them regain the market share they were quickly losing to Microsoft’s new Internet Explorer. Albeit the fact that in *The Cathedral and the Bazaar* Raymond pays lip service to Stallman (whom he calls “a genius of design”) he didn’t actually think of the FSF as a relevant social group anymore. Faced with a real opportunity for changing the way software is written, distributed and used in business practice, Stallman’s theoretical concepts seemed outdated and even detrimental to the new cause.

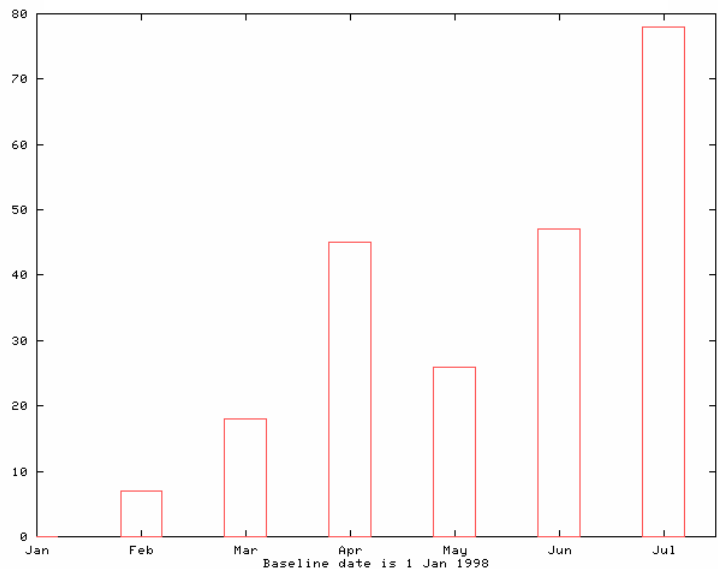
The "open source" label itself came out of a strategy session held on February 3rd 1998 in Palo Alto, California ... We were reacting to Netscape's announcement that it planned to give away the source of its browser. One of us (Raymond) had been invited out by Netscape to help them plan the release and followon [sic] actions. We realized that the Netscape announcement had created a precious window of time within which we might finally be able to get the corporate world to listen to what we have to teach about the superiority of an open development process. We realized it was time to dump the confrontational attitude that has been associated with "free software" in the past and sell the idea strictly on the same pragmatic, business-case grounds that motivated Netscape. We brainstormed about tactics and a new label. "Open source," contributed by Chris Peterson, was the best thing we came up with. (History Of The OSI 2003)

This rhetoric move by OSI was effective and had long lasting ramifications on the future of free and open source software. The name change was soon followed by releasing a new licensing agreement template that was much more permissive than the GPL. It allowed the amalgamation of open-source software with proprietary software in a commercial software distribution without requiring future work to be released under a free software license. This was a deadly mix for complete freedom concepts to which the FSF had repeatedly opposed, but it cleared the way for major businesses to adopt open source methodologies without

having to compromise future software sales and revenues by making their proprietary knowledge available under the GPL. Both the change in the licensing regime and the rhetoric change were aimed at solving the problem of for-profit businesses, the new relevant social group. Soon, free software, in its new outfit as open source, was being imbued with meanings in the corporate world.

Attaining rhetorical closure, at least ostensibly, worked well. Open source projects were finally getting wide attention and creating a real alternative to closed-source systems. Within less than a year from OSI's February 1988 *strategy* session most of the key players in the computer industry and important media organizations have shown some sort of support to the growing open source community. Corporate juggernauts like IBM, Silicon Graphics, Oracle and Hewlett Packard embraced Linux. Apple computers released the source code of its new operating system under an open-source license, and the open-source Apache, which by now is the world's most ubiquitous web-server software, was gaining ground. (History of the OSI 2003). A good measure to the success of a rhetorical move is the reaction in the media. Within a short while both Forbes and the Economist (the hard-core of business journals) ran major stories on the growing phenomena of open source and these were followed by other media organizations exponentially (see the figure below.)

Figure 1: Lexis-Nexis search that tracks the number of references to "open source" in American newspapers and magazines in 1988 (Lexis-Nexis mentions of open-source 2003)



Second attempt at Closure: Sun Microsystems, Microsoft and The Halloween documents

But, of course, not everybody was happy. On one hand there was the FSF and Stallman, who felt that by moving away from the concepts of freedom the OSI had dumped the baby away with the bathwater. In Stallman's eyes the OSI's move was simply a betrayal. As he wrote to me in an email interview: "it is unethical and antisocial to prohibit other people from cooperating when they wish to (e.g. by distributing non-free software)... the Open Source movement was formed to reject that view" (Stallman 2002). On the other hand, there was a new relevant social group: traditional software businesses that felt that—in opposition to OSI's contention that open source makes-sense business-wise—reality shows that the new initiative was putting their existence at risk. Two companies that were members of this group are Sun Microsystems and Microsoft.

Sun was co-founded in 1982 by Bill Joy, who was a graduate student at Berkeley and one of the key architects of Berkeley's BSD, and as such a significant contributor to the free software / open source phenomena himself. By 1998, however, Joy, whom Fortune Magazine, recognizing 25 years of technological innovation, called "the Edison of the Internet," (Schlender 1999) has switched sides. As a chief scientist in Sun, he was a billionaire, and an executive of one of the most well-oiled money making machines in the software industry. Sun was under pressure from open source, and decided to make Solaris (its version of Unix) available under a free license to individual users and to educational/non-profit/research institutions. In a press release that explains what the new license was all about Sun writes:

The Free Solaris program seeks to foster communities of interest and facilitate self-help, learning and discussion among Solaris developers, students, educators and researchers ... Individuals, non-commercial developers and members of the academic community who register for the Sun Developer Connection under the Free Solaris program will have access to all Sun Developer Connection resources. (Sun Microsystems 1998)

In other words Joy's Sun wanted to substantiate its operating system as a technological frame on its own right. Making Solaris, which was widely used already, a free operating system would appeal to the social group we encountered earlier, computer science research centers and educational facilities. However, Sun was not willing to espouse the full implications of releasing its software as open source software and forfeiting the main slice of its \$9 billion (1988) annual revenues pie. On the same press release there is a specific warning:

Use of the Solaris operating environment secured through this offer is limited to non-commercial use only. Participants are free to develop and test applications, but in order to deploy them for commercial use, they must upgrade their software to a commercial license. (Ibid)

In fact Sun was trying to eat the cake and leave it whole: it wanted to satisfy two relevant social groups at once – the academia and developers' community and its share-holders whose sole interest was profitability. This mixed interest produced a confused strategy. Solaris was never released as an open-source system, but Java, a revolutionary programming language developed at Sun was conceived and developed as a full-fledged open source project (on which until this day Sun is hardly making any money). In sum, Sun seems to be torn between two technological frames.

Microsoft, unlike Sun, never understood the value of open source. Bill Gates, Microsoft's founder made his ideas about the meaning of software clear as early as 1976, when he sent an "open letter to hobbyists" to the developer community and early adopters of his BASIC software. Intended, in part, to the MIT AI lab hacker community of which Stallman was a part, the letter read:

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

... As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? ... One thing you do do is prevent good software from being written. ...I would appreciate letters from any one who wants to pay up, or has a suggestion or comment...Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates, General Partner, Micro-Soft (Gates 1976)

In this normative missive, Gates addresses the social group of hackers and hobbyists, and tries to convince them that the biggest problem they have with software is that good software doesn't exist, and that their sharing ethic diminishes the motivation for anyone to spend resources and generate it. However, his description of the hacker community and early computer users as a 'market' (and not a community, or a club) clarifies that what he really perceived as the problem was that he couldn't engage in a profitable exchange with them. It's also a clue to the future of his venture. In 1998 Micro-Soft (which later became Microsoft) was neither small nor soft. Despite what they called 'theft' of their software by hobbyists, Microsoft and the other firms that followed it were able to create a real market for software. For over two decades Gates rode the waves of the computer revolution audaciously making excellent business decisions along the way (most notably, negotiating a deal with IBM according to which Microsoft would license to IBM an early version of the Disk Operating System (DOS) while keeping the intellectual property rights to itself). In a process that deserves a full social constructivist analysis of its own (but is beyond the scope of this paper) Microsoft became the world's largest and most profitable software company, 'Windows' became synonymous with 'operating system', and the world of commercial operating systems has reached closure. That is, until the open source beast raised its head.

In the last week of October 1998, a confidential Microsoft memorandum on the company's strategy against Linux and Open Source software was leaked to Raymond and attests to what was going-on behind the scenes of this apparent stability. Raymond released the document over Halloween weekend to the national press, and it became notoriously known as the Halloween document. Microsoft was forced to acknowledge its authenticity. The document was put together by Microsoft employees who wanted to alert the management on the perils they can expect from the new kid on the block. It reads in part:

OSS [Open Source Software] poses a direct, short-term revenue and platform threat to Microsoft, particularly in server space. Additionally, the intrinsic parallelism and free idea exchange in OSS has

benefits that are not replicable with our current licensing model and therefore present a long term developer mindshare threat.

Recent case studies (the Internet) provide very dramatic evidence ... that commercial quality can be achieved / exceeded by OSS projects....to understand how to compete against OSS, we must target a process rather than a company.

Linux and other OSS advocates are making a progressively more credible argument that OSS software is at least as robust -- if not more -- than commercial alternatives. The Internet provides an ideal, high-visibility showcase for the OSS world. ... The ability of the OSS process to collect and harness the collective IQ of thousands of individuals across the Internet is simply amazing. More importantly, OSS evangelization scales with the size of the Internet much faster than our own evangelization efforts appear to scale. (Valloppillil and Cohen 1988)

What is important to note in this document is the importance the authors grant to the developer community, whose mindshare they are afraid to lose. We saw that for Torvalds too, this group was an important audience. Although Microsoft makes most of its money from selling software to people that don't know anything about computer programming, it is clear to them that in order to ensure that their captive audience remains captive, the developers should be convinced first. Parallelism and free idea exchange is perceived as a threat to Microsoft which is a hierarchical firm and which is used to compete with other companies, and not with processes. The reference to the larger relevant social group (the general user-base) is through the concept of evangelism. Linux is dangerous because it is able to evangelize itself better using the Internet. In our terms: the danger that Microsoft perceives is that it might lose hold on the problem definition for the largest relevant social group. Linux can sway the users to support it by convincing them what their problems are better than Microsoft can. Clearly for Microsoft an operating system has a different meaning than for the open source community. For Microsoft, just like for LMI and Symbolics two decades earlier, software is first and foremost a tool to make money. They are interested in the bottom line. For the open source community software is first and foremost a collaborative process for solving problems. They are focused on the method. Although the OSI's rhetoric moves were aimed at alleviating this tension between methodology and outcomes, Microsoft was not convinced. It continues till this day to perceive software differently, and is so far successful in its attempts to hold firmly to its own technological frame.

A third attempt at closure: redefining the problem Red-Hat and Open-Source business models

One person that understood that rhetoric moves are not enough is Bob Young, the chief executive of Red-Hat software, the leading Linux distributor. Young is an old business-fox that understood early on that if any sort of stability should be attained for free software and open source, the problem that open source software creates should be reformulated. He, like others before him, understood that the way to get to the social group of users passes through the group of developers and software companies. The key problem with open source software that developers had is simple: by giving away the software, they had to give away their revenues. This, as we saw in Microsoft's case was not an option. But Young wanted to think of the problem differently. What software companies want is to make money, not sell software. The question to ask, then, is not "how do you make money selling software?" (which in the free software case reduced to the oxymoron "how do you make money selling free software?") but, rather, "how do you make money in software?" (which in the free software case simply translates to "how do you make money in free software?"). Young writes

No one expects it to be easy to make money in free software. While making money with free software is a challenge, the challenge is not necessarily grater than with proprietary software. In fact you make money in free software exactly the same way you do it in proprietary software: by building a great product, marketing it with skill and imagination, looking after your customers, and thereby building a brand that stands for quality and customer service. (Young 1999:114)

Young's idea rests on the premise that even in free and open source software environments the users are still willing to pay for packaging, distribution, maintenance and professional services. Selling the software is only a part of a bigger transaction, which, according to Young isn't very important. Moreover, once the developers are convinced that they can make money, they will focus on seeing the huge advantages that open source has: better control for the user. The discussion could shift from the problems the developer has to the problems the users have: "Open-source code is a feature. Control is the benefit: Every company wants control over their

software, and the feature of open source is the best way the industry as found so far to achieve that benefit” (Ibid:123).

As a proof of how Young’s ideas work in practice we can look at the short history of Red-Hat software itself. To be sure, Red-hat was not impressed with the OSI’s rhetorical closure attempts. It selected the GPL and not the OSI license scheme for its Linux distribution. The company started by distributing packages of Linux developed by others but soon became a powerhouse for Linux development, testing, and professional services. Young made Red-Hat a very profitable Linux venture, and took it public. The stock options Red-Hat gave to free software mavens like Torvalds for their principle contribution to Linux made them millionaires overnight. Stallman, rejected the stock options he was offered.

But Young’s moves can be perceived on a more profound level. In his essay on the social meaning of the personal computer revolution Bryan Pfaffenberger argues that the personal computer revolution wasn’t a revolution at all because, in the last instance, it succumbed to the dominant ideology that saw the computer first and foremost as a tool for control. Through the workings of three strategies of social behavior that he calls ‘regularization’, ‘situational adjustment’ and ‘reconstitution’ Pfaffenberger explains how the revolutionary aspirations of the first hardware-hackers was diffused, and how through organizational computer networks personal computers became part of the institutional infrastructure for control. This lesson of the docile fate of ‘revolutionary’ computer hardware reverberates in Red-Hat’s story. Ironically, people like Young, as the quote above shows, talk about control as the single most important feature in open source, but while they feel committed to the free software vision they are not ideological about it, and they easily submit to the control of bigger ideologies (Capitalism in our case) without even noticing it. “The very essence of a dominant ideological system, as Habermas and others have argued, is that it forces everyone to think in the terms that it lays down, even if you reject it, you have to reject it using its terms, and so—unwittingly and unintentionally—would-be rebels actually reproduce the ideological structure intact” (Pfaffenberger 1988:45). After two decades of development, when Stallman’s vision was finally realized, and GNU/Linux became a

real, widely used, free operating system, the redefinition of the problems as performed by companies like Red-Hat loops right back into the same social problems of Capitalistic control that prompted the development of free software to begin with. It took 20 years, but eventually, Free and open source software are at risk of becoming a tool for Capitalistic exploitation.

Do we learn from this that open-source can be closed? Can it reach, in principle, stabilization and closure? Its recent history suggests that it might. For the software industry the meanings of free and open source software are becoming clearer, if only because the business world was able to appropriate them into its capitalistic ideology. But at a deeper level the openness of the system suggests that in its very nature free and open source software retains a high-level of interpretive flexibility. Institutionally, closure is at sight, but technically the availability of the source code will always keep its meanings flexible, and there is always a chance that the technological frame could once again shape its social environment openly.

Conclusion: Lessons learned for SCOT

I want to conclude by summarizing three critical points. First, consider Bijker's account of the different phases of social constructivism that are supposed to help us bring order to the relativistic social constructivist chaos. Bijker identifies three potential situations that characterize the relationship among the relevant social groups during an artifact's technological development.

First there is the situation in which no one social group with its accompanying technological frame is dominant...The second situation is characterized by the dominance of one social group and the accompanying technological frame. Probably, this is the most common situation "normal technology", to paraphrase Kuhn... In the third situation, two or even more social groups with clearly developed technological frames are striving for dominance in the field. The difference from the first situation is that in that case the many relevant social groups do not yet have distinctive technological frames with respect to the artifact in question, whereas in the latter situation they have. (Bijker 1987:182)

Although, as Bijker stresses, these situations should not be considered as chronological phases, arguably it seems as if in the case of free and open source software this is exactly the sequence in which the technology

was developed. The first phase, was the phase in which Stallman's interpretation was competing with Symbolic's and LMI's and even more directly with AT&T's—what free software was had yet been fully negotiated. The second situation occurred when what software meant became clear and companies like Microsoft and Sun dominated the commercial software arena. The third stage is happening now: the commercial entities are struggling with open source enterprises, when both are trying to enforce their interpretation of what software means—to them and the groups they find relevant. However, stopping our explanation here would miss a key element of these situations. I believe that we should think of software (or any other technology for that matter) as a Russian doll for which the levels of meaning are encapsulated by one another, rather than being lateral to one another, and thus can be explored in different levels of depth. In a recursive manner our inference of which phase we are faced with changes drastically if we change our technological reference: when we choose Linux 1991 as our reference point, for example, we can identify a Phase I situation, when no interpretation is dominant, without noticing, that for the bigger technological reference of, say, 'operating systems' in general Phase II—stabilization and closure on Microsoft's Windows—is more appropriate. In other words, selecting our point of reference in a technological system introduces methodological uncertainties into the observation, in the same way that Heisenberg's principles predict for atomic particle systems. The technological 'path' is revealed only when we locate ourselves and look at it.

This observation leads me to my second point. Russel (1986) offers a critique of SCOT in which he finds that:

[A] major weakness of the SCOT approach, as the paper describes it, is an inadequate conception of social structure...Far beyond just 'identification' and 'description', groups need *locating* in a structured and historical context. ... The diagram in Pinch and Bijker's paper intended to show 'relationships' between an artifact and social groups is at best trivial; but it may also visually reinforce an impression of social groups isolated from each other and effectively equal in power. (Russel 1986:334-335)

I am sympathetic to Russel's critique, but for different reasons. As the case of free and open source software clearly shows the concept of 'relevant social groups' is misleading not because we cannot fully locate them in

a socio-historical-political-economical concept, but rather, because at key points in the development course the social groups do not yet exist. Instead, at key points in the story we find outstanding individuals and determined corporations. We cannot hope to have an unbiased and symmetrical account of every relevant social group's attitude towards a technology and capture the uneven social interaction between the groups at one and the same time simply because the groups do not play on the same field as the simplified SCOT socio-grams might suggest. Some groups are mediated by individuals who have their own agendas and interested interpretation of what the groups' problems are or ought to be, while other groups incorporate as firms which succumb to larger economic pressures. A simplistic group-interaction approach will not do.

Third, and directly connected to the second point, bearing the uncertainty principle in mind, we need, at the very least, to add a temporal dimension to our analyses, in a way that will allow us to overcome the problems in grasping both position and momentum accurately. This would come at the cost of forfeiting the ostensible stability and the putative ease of designating the relevant social groups and their respective technological frames. In other words, we need to give up the attempts to produce lucid socio-technological snap-shots, and instead commit to methodological iteration—only by iterating the analysis while changing the time-reference can we hope not to lose the evolutionary dimensions of technological change. I believe that maintaining SCOT's commitment to symmetry with regards to 'truth', 'performance', 'quality' or any other normative measurement that supposedly renders one technological alternative as superior to another, we can iterate the analytical process and yet avert the slippery slope of technological determinism.

I hope that my treatment of the historical case of free and open source software can teach us social constructivists a more general lesson. SCOT, with proper modifications as outlined above, needs to and can describe how technology is socially constructed over time and in relationship to the political-economic environment that calls individuals into action, how technology allows relevant social groups to achieve their goals directly, through individuals, or through collaboration and incorporation, and how these groups exert uneven power over each other when trying to reach closure on their terms. Graphically put, the famous SCOT

socio-grams should be animated to allow for more analytical depth which is missing today. We should think of them, as I suggested, as screen-shots from our social-constructivist radar that need to be refreshed in short intervals in order to remain meaningful and distinguish signal from noise.

References

- Bardini T. and A. Horvath, "The Social Construction of the Personal Computer User" *Journal of Communication* Volume 45 n3. 1995.
- Bezroukov, N. "BSD vs. GPL: A Framework for the Social Analysis", Unpublished. 2002 Available online at < http://www.softpanorama.org/Copyright/License_classification/social_dynamics_of_BSD_and_GPL.shtml >
- Bijker, W. *The social construction of Bekelite: Towards a theory of invention*. In W. Bijker, T. Hughes, & T. Pinch (Eds.), *The social construction of technological systems: New directions in the sociology and history of technology* (pp. 154-187). Cambridge, MA: MIT Press. 1987
- Ceruzzi, Paul E. *A History of Modern Computing*. Cambridge, MA: The MIT Press, 1998
- Gates W. "Open Letter To Hobbyists." Letter to developer community. February 3, 1976. Available online at < <http://www.blinkenlights.com/classiccmp/gateswhine.html> >
- GNU's Bulletin, vol. 1 no. 3 - GNU Project - Free Software Foundation (FSF) available online at < <http://www.gnu.org/bulletins/bull3.html> >
- GNU Project Homepage. Free Software Defintion. Available online < <http://www.fsf.org/philosophy/free-sw.html> >. Visited on December 9th, 2003
- Kelty, Christopher M. "Hau to do things with words". *Knowledge and Society*, vol. 13, JAI Press, 2001.
- Kim J. and Watanabe T. "The Social Construction of the Internet and Emerging Problems of Internet Governance", Presentation at "Bugs: Globalism and Pluralism, Montreal April 24th 2002. Available on line at < <http://www.er.uqam.ca/nobel/gricis/actes/bogues/KimWatan.pdf> >. Visited December 18th 2003.
- Harmon J. and J. Seeker "AT&T Resource Link " *Case Research Journal*, vol. 18, no. 2, 56-81. 1999.
- Heisenberg, W. 'Ueber den anschaulichen Inhalt der quantentheoretischen Kinematik and Mechanik' *Zeitschrift für Physik* 43 172-198. 1927. English translation in Wheeler, J.A. and Zurek, W.H. (eds) *Quantum Theory and Measurement*, Princeton, NJ: Princeton University Press. 1983. Discussion available at < <http://www.seop.leeds.ac.uk/archives/win2002/entries/qt-uncertainty/> >. Visited December 18th, 2003.
- "History of the OSI" available on the Open Source Organization. Homepage. Available online at < <http://www.opensource.org/docs/history.php> >. Visited on December 9th, 2003.

Levy, Steven. *Hackers: Heroes of the computer revolution*. New York: Penguin Books, 1984.

Mauss, Marcel. *The gift: the form and reason for exchange in archaic societies*. New York and London: W.W. Norton. 1990.

McKusick M. K. "Twenty years of Berkeley Unix: From AT&T-Owned to Freel Redistributable." in DiBona, Chris, Sam Ockman & Mark Stone.(Eds.) *Open Sources: Voices from the Computer Revolution*. Beijing: O'Reilly Associates: 1999.

"Mentions of Open-Source on Lexis-Nexis in 1988". Graph. Available on the Open Source Organization. Homepage. Available online at < <http://www.opensource.org/graphics/mentions.png> >. Visited on December 9th, 2003From: >

Moody, Glyn. *Rebel Code: Inside Linux and the Open-Source Revolution*. New York: Perseus Publishing, 2001.

Open Source Organization. Homepage. Available online at < <http://www.opensource.org/index.html> >. Visited on July 13, 2003.

Pfaffenberger B., "The Social Meaning of the Personal Computer: or, Why the Personal Computer Revolution was No Revolution," *Anthropological Quarterly* (1988), 39-47

Pinch, T. & Bijker, W. *The social construction of facts and artifacts: Or how the sociology of science and the sociology of technology might benefit each other*. In W. Bijker, T. Hughes, & T. Pinch (Eds.), *The social construction of technological systems: New directions in the sociology and history of technology* (pp. 17-50). Cambridge, MA: MIT Press. 1987

Pinch, T. and R. Kline "Users as Agents of Technological Change: The Social Construction of the Automobile in the Rural United States," *Technology and Culture*, 37, 1996, 763-795.

Raymond, Eric. *The Cathedral and the Bazaar Musings on Linux and Open Source by an Accidental Revolutionary*. Beijing: O'Reilly & Associates. 1999. Also available online < http://www.firstmonday.dk/issues/issue3_3/raymond/ >. Visited on July 13, 2003.

Russell S. "The Social Construction of Artefacts: A Response to Pinch and Bijker" *Social Studies of Science*, Vol. 16, No. 2 May, 1986, 331-346.

Schlender B. "The Edison Of The Internet " *Fortune Magazine*, February 15th, 1999. Available online at < <http://www.fortune.com/fortune/articles/0,15114,376666,00.html> >. Visited Dec 09 2003.

Stallman, Richard. "Why Software Should Not Have Owners". In *Sarai Reader 2001: The Public Domain, Free as in Freedom': Software as Culture*. 2001. pp. 176-180. Available online <<http://www.sarai.net/journal/pdf/175-227%20Free.pdf>>. Visited on July 13, 2003.

-----, Interview with the author. April 2002.

-----, Interview with BYTE Magazine editors David Betz and Jon Edwards, *Byte Magazine*, July 1986. Available online at < <http://www.gnu.org/gnu/byte-interview.html> >

Sun Microsystems. "SUN aggressively expands developer base for Solaris operating environment." Press Release. August 10th, 1988. Available online at < <http://www.sun.com/smi/Press/sunflash/1998-08/sunflash.980810.2.html> >. Visited Dec 9th, 2003.

Valloppillil V. and J. Cohen "Open Source Software: A (New?) Development Methodology", Internal Microsoft Memorandum. October 1988. Available online at < <http://www.opensource.org/halloween/halloween1.php> >

"What is Copy Left." Sarai Reader 2001: The Public Domain, Free as in Freedom : Software as Culture". Available online < <http://www.sarai.net/journal/pdf/175-227%20Free.pdf> >

Young, R. "Giving it all away: How Red Hat Software Stumbled across a New Economic Model and Helped Improve an Industry" in DiBona, Chris, Sam Ockman & Mark Stone. (Eds.) *Open Sources: Voices from the Computer Revolution*. Beijing: O'Reilly Associates: 1999. pp 113-125.