Dissertation title:

The Circulation of Influence - Free and Open Source Software as an Arena for the Realisation of Value

Examination candidate number:

52447

Programme:

MSc in Social Anthropology

Year:

2015

Words:

9833

Table of contents

Abstract

Free and Open Source Software (FOSS) is computer software produced by groups of volunteers loosely organised around certain development practices. FOSS projects lack formal institutional arrangements or instruments of coercive power, and their leaders cannot impose decisions by force. Their success and survival relies on the ability of a small group of core developers to recruit sufficient numbers of volunteers and keep them "constantly stimulated and rewarded" (Raymond 2000).

Given these loose structures and relationships, one would expect FOSS projects to unravel and disappear continuously under the pressure of internal disagreements and strife, personal differences and egos, and the ease with which computer code can be duplicated, changed, and turned into new, competing versions, a process known as "forking" (Lerner and Tirole 2005: 53). Counterintuitively, this is not the case. Forking, although it happens occasionally, it's relatively rare (Glass 2005), and successful FOSS projects like the Linux kernel or the Apache web server have proven to be remarkably resilient.

This dissertation analyses the mechanisms that contribute to the stability and permanence over time of FOSS projects. It defines FOSS as one of the several, competing arenas for the pursuit of certain forms of value that co-exist within Western, liberal democracies; and applies Marx's theory of value, as interpreted by David Graeber (2001, 2005, 2013) and Terence Turner (2006), to the study of FOSS development practices.

By identifying the main values pursued by FOSS contributors and how they are produced, circulated and accrued through symbolic, material tokens, the dissertation uncovers that the ability of core developers to accumulate surplus value contributes to the stability of FOSS projects and the prevention of forking.

1. Introduction

Free and Open Source Software (FOSS) makes its source code, the human-readable version of the computer code, publicly available, so that it can be used, read, modified and shared by anybody. It takes the opposite stand of commercial or proprietary software, which invests great effort into keeping its source code secret.

FOSS is also different from commercial software because it is not developed by tightly managed corporate teams, but by groups of volunteers loosely organised around certain software development practices. FOSS projects lack formal institutional arrangements or instruments of coercive power, their leaders cannot impose decisions by force, they have "no specific lines of (...) responsibility running between the constitutive elements, and no one to blame when things go wrong" (Ratto 2005: 828). Their success and survival relies on the ability of a small group of core developers to recruit sufficient numbers of volunteers and keep them "constantly stimulated and rewarded" (Raymond 2000).

Given these loose structures and relationships, one would expect FOSS projects to unravel and disappear continuously under the pressure of internal disagreements and strife, personal differences and egos, and the ease with which computer code can be duplicated, changed, and turned into new, competing versions, a process known as "forking" (Lerner and Tirole 2005: 53). Counterintuitively, this is not the case. Forking, although it happens occasionally, it's relatively rare (Glass 2005), and successful FOSS projects like the Linux kernel or the Apache web server have proven to be remarkably resilient.

The literature on FOSS suggests that stability and continuity in FOSS projects are connected to a certain leadership style that relies on the transformation of computer code contributions into trust, charisma and reputation. I argue that such transformation takes places through the routine FOSS development practices of patch

writing, patch submission, patch review and code merging, and through the importance and value that FOSS volunteers attribute to them.

To build my argument, I will follow David Graeber (2001, 2013) and present FOSS as one of the several, competing arenas for the realisation of value that co-exist within Western, liberal democracies. The values pursued within this FOSS arena can be classified into motivational, contextual and structural values.

Motivational values are the ones that compel FOSS volunteers to act: passion, a deep interest one feels, develops and nurtures regarding one ore more technical subjects; and self-realisation, a desire to express oneself through creative activity.

Contextual values set the environment, circumstances and rules under which action takes place. They are freedom, understood as self-determination, or the possibility to self-select for certain projects, technical subjects or contribution types; and openness, understood as the right to freely access and build upon the product of others's creative efforts, which in practice means the source code written by fellow contributors.

Structural values guarantee the survival of FOSS over time. Merit is the main structural value: the social recognition by peers of the worth of one's creative output, and of one's ability to influence and shape the future of any piece of software to which one happens to contribute.

Using Marx's theory of value as interpreted by Terence Turner (2006) and David Graeber (2001, 2005, 2013), I will analyse how structural values are produced, circulated and realised through the FOSS development process.

Turner and Graeber postulate that Marx's theory of value is not exclusive to capitalist societies with markets and a state, but that capitalism is only one example of a general tendency, present in many societies, 1) to represent the human creativity invested into production as symbolic material tokens that turn such incommensurable creativity into

comparable units, 2) to think of such comparable units as part of a totality of production and 3) to employ the material tokens as media of exchange, circulation, appropriation and accumulation of value (Turner 2006).

It should be possible to apply this framework to the study of non-capitalist societies (Turner 2006) and, by extension, to the analysis of co-existing arenas for the realisation of certain values (such as FOSS), in order to improve our understanding of how alternative forms of value manage to exist and persist within societies dominated by neoliberal ideals of wealth accumulation and market efficiency.

The dissertation is structured in four sections. The first one introduces the concepts of source code and binary code, provides some background on FOSS and its historical origins, and describes its software development practices.

The second section reviews some of the existing literature about the ethics of FOSS, its constituent values and its relationship with liberalism.

The third section outlines the main concepts of Marx's theory of value as interpreted by Terence Turner and David Graeber, in particular its definition of society as the combined outcome of individual creative action, its broad concept of production, the representation of productive action through symbolic material tokens, and the social character of value.

The final section applies Marx's theory of value to the FOSS development process, and reveals how the ability of core developers to accumulate surplus value contributes to the stability of FOSS projects, their continuity over time, and the prevention of forking.

2. Software, Free Software and Open Source Software

This section introduces the concepts of software, source code and binary code; explains what Free Software and Open Source Software (FOSS) are and how they differ from proprietary software; and it describes aspects of the FOSS development process that are relevant to this dissertation.

2.1 Source code and binary code

Software is the name we give to the programmes and applications that execute in our computing devices, and that make them useful to us. It is the intangible component of computing, as opposed to the physical one or hardware: the computers themselves.

Software is made of computer code. This code takes different forms during its writing phase and its execution phase. The code that software developers write, generally known as source code, can be read and understood by humans, but not by computers. The human readable source code needs to be translated into binary code, sequences of ones and zeros that can be read and executed by machines. The translation is done by an utility, itself a piece of software, called "compiler" or "interpreter" depending on how it carries out the translation process. In general, binary code is not human readable, at least not easily.

We have become accustomed to the idea of purchasing binary code. When you buy a license for a piece of software, what you are normally buying is the right to execute its binary code in a certain computing device, and therefore to use the software to do whatever it is that it does: send emails, write documents, or help you carry out complex calculations. We never see the source, the human readable state of the computer code. This is because source code is considered to be the magic sauce of software: if it were to fall in the hands of your competitors, it would be easy for them to replicate it, change it just enough to avoid accusations of theft and copyright violation, then distribute a

similar application to yours, undermining your income in the process. Source code "secrecy", therefore, is "a means to control the market" (Kelty 2008: 101).

What is perhaps less known is that things were not always like this. There was a time when binary code was not bought and sold, and source code was freely shared.

2.2 Free and Open Source Software (FOSS)

In the late 1950s and early 1960s, when computing was still young and mostly an academic endeavour, computers were big and quite unique. In fact, they were so different from each other that software that ran on a certain type of computer would not run in another without considerable adaptation. As a result, commercial companies sold mainly the hardware, and bundled the software into the deal. Software, by itself, was rarely bought and sold.

At that time, sharing source code was common practice. It allowed people to learn from each other, port applications to different computers, reuse work to avoid duplication of effort, and improve or adapt software to perform new tasks and serve new purposes.

This practice of sharing source code, which precedes both personal computers and the Internet, constitutes the origins of Free and Open Source Software (FOSS). Strictly speaking, the beginning of Free Software can be traced to the start of the GNU project in January 1984 and the subsequent formation of the Free Software Foundation in October 1985 by Richard Stallman (Moody 2001). The beginning of Open Source software can be tracked to the Freeware Summit organised in 1997 by Tim O'Reilly, and the creation of the Open Source Initiative by Bruce Perens and Eric S. Raymond in 1998 (Kelty 2008).

Although different in name, Free Software and Open Source Software refer "to identical practices, licenses, tools, and organizations" (Kelty 2008: 116). The short descriptions provided by the entry pages of fsf.org and opensource.org, the websites of

the Free Software Foundation and the Open Source Initiative respectively, are indeed remarkably similar.

fsf.org (accessed in July 2015) states that "Free software developers guarantee everyone equal rights to their programmes; any user can study the source code, modify it, and share the program".

opensource.org (accessed in July 2015) says that "Open source software is software that can be freely used, changed, and shared (in modified or unmodified form) by anyone".

As per the definitions above, both Free and Open Source software make their source code publicly available, so that it can be used, read, modified and shared. From this point of view, FOSS is the antithesis of commercial or proprietary software, which zealously guards its source code to keep it secret.

2.3 The FOSS development process

FOSS is written and maintained not by tightly managed corporate teams, as proprietary software is, but "through collaborative, informal networks of professional or amateur programmers" (Ghosh 2005: 23) who constitute a community of contributors. Contributors may be paid by companies to collaborate in the development effort, or might do so in their spare time, but in both cases the contributions are volunteered: they are offered to the community as a means to fix, improve or grow the software being developed.

This community model constitutes an economic puzzle. "To an economist, the behaviour of individual programmers and commercial companies in open source processes is startling. (...) Why should thousands of top-notch programmers contribute freely to the provision of a public good?" (Lerner and Tirole 2005: 47-48).

But the community model is not only an oddity from an economic perspective, it also has two other major implications. The first is that FOSS contributors are "self-selected" (Raymond 2000); the second, that all FOSS is peer-reviewed.

2.4 Self-selection

When Raymond (2000) says that contributors are "self-selected", he means that developers are free to choose the FOSS projects to which they contribute, selecting those that raise their interest, and participating in them mainly for their enjoyment and pleasure.

Contributions can take many forms, and do not necessarily involve computer code. Community members might report faults (or "bugs") in the software, write documentation, answer questions from inexperienced users, translate the software to other languages, design the graphical user interface, and of course write code, either to fix problems or to incorporate new features.

Although all contributions are appreciated, in practice some are more appreciated than others: "those who write the code truly control the project" (Zawinski 1999). Between the contributors who write code, "core developers" (Raymond 2000) are those "who control the code base" (Mockus et al. 2005: 186), which means they have the power to decide which code contributions are incorporated into the software (or "merged upstream") and which are not.

Core developers also write the bulk of the code (Lerner and Tirole 2005) and most of the new functionality (Mockus et al. 2005) in spite of being a small group: "a single core developer is common, and one to three is typical" (Raymond 2000). Core developers are often the authors of the initial version of the software, as in the case of Linus Torvalds and the Linux kernel (Raymond 2000). They can also acquire core status through a continued stream of contributions to the code base (Mockus et al. 2005).

They might be appointed by a previous core contributor (Raymond 2000), or sanctioned by the community to take the role through voting (Mockus et al. 2005).

In spite of the importance of core developers, the success of a FOSS project depends on attracting sufficient contributors to guarantee bug reporting and fixing (Raymond 2000, Mockus et al. 2005). Therefore, much in a project's future hangs on the ability of the core developers to recruit volunteers. Doing so successfully, according to Raymond, requires a certain "leadership style and set of cooperative customs" that cannot be based on what he calls "leadership by coercion" or the "principle of command", which he considers "effectively impossible to apply among volunteers" (Raymond 2000). Raymond seems to be referring to Eric Wolf's second mode of power, "the ability of an *ego* to impose its will on an *alter*, in social action, in interpersonal relations" (Wolf 1990: 586). Since FOSS contributors are self-selected and keep the freedom to stop contributing at any time, imposing in them a specific course of action seems indeed impossible.

Raymond suggests instead that the ability to recruit and conserve volunteers relies on keeping them "constantly stimulated and rewarded" (2000). Stimulation is achieved through the satisfaction of having one's code incorporated to the overall code base; reward derives from witnessing and experiencing the software's improvements.

2.5 Peer review

The second implication of the community model is that all FOSS is peer-reviewed, and it is so in two ways.

The software itself, as a single body of computer code and a functional unit, is peer-reviewed through usage. The more a piece of software is used, the higher the likelihood that its faults will be found and reported, which in turn increases the likelihood that they will be addressed and ultimately fixed. This generates a cycle of continuous improvement, increasing the software stability and quality (Raymond 2000).

Individual code contributions, known as "patches", are also peer-reviewed before being incorporated to the project's code base. Patches are submitted to public forums where all subscribed can read them, comment on them and recommend changes and improvements to the code. Although in theory anybody can issue comments, in practice patch review is mainly done by core developers, experienced contributors, or experts in certain functional areas.

2.6 Neither organisations nor movements

FOSS also lacks formal institutional arrangements. Although some FOSS projects might constitute themselves into entities like foundations for administrative and tax purposes, and others "have grown into large semiformal institutions with complex governance structures" (Coleman 2009), when it comes to their development practices they are not corporations, organisations, or consortia; they are not collectives since they don't require membership; and they are not a social movement, since they are articulated around practices and not ideologies (Kelty 2008).

Given the loose structures and relationships described in this section, one might expect FOSS projects to unravel and disappear continuously under the pressure of internal disagreements and strife, personal differences and egos, and the ease with which computer code can be duplicated, changed, and turned into new, competing versions, a process known as "forking" (Lerner and Tirole 2005: 53). Counterintuitively, this is not the case.

Although forking does happen, as shown by the examples provided by Lerner and Tirole (2005), it is "extremely uncommon" (Glass 2005: 88). Successful FOSS projects have also proven to be remarkably resilient: the GNU C Compiler (GCC) has been going since 1987, as recorded in https://gcc.gnu.org/wiki/History; Linux, since 1991 (Moody 2001); Debian, since 1993 (Coleman 2013); and Apache, since 1995 (Mockus et al. 2005). Very recently, one of the core developers of OpenEmbedded expressed to me his "shock" at the fact that the project has now been running for ten years.

The literature on FOSS seems to agree that stability and continuity in projects are connected, as the successful recruitment of contributors is, to a certain leadership style:

> While the leader has no "formal authority" (is unable to instruct anyone to do anything), he or she has substantial "real authority" in successful open source projects. (...) The presence of a charismatic (trusted) leader is likely to substantially reduce the probability of forking (Lerner and Tirole 2005: 64-65).

> The greater the perceived talent of OSS project leaders, the less likely that their authority will be questioned when they arbitrate on disputes, choose between competing contributions, set the direction for the project, and generally prevent forking. (...) OSS is fundamentally a reputation-based economy (Fitzgerald 2005: 105).

Becoming a leader in the context of a FOSS project is a matter of "exceptional contributions" (Szczepanska et al. 2005: 438). But how those contributions are transformed into authority, trust, charisma and reputation remains, however, unclear.

I argue that such transformation is intimately connected to the routine practices of patch review, patch submission, source code merging and software use; and to the importance and value that FOSS contributors attribute to them.

3. What hackers value

In this section, I review what has been written about the ethics of hacking, the moral principles or values that underpin the actions of those involved in FOSS.

The term "hacker" is a broad one. A hacker is "a technologist with a penchant for computing and a hack is a clever technical solution arrived at through non-obvious means" (Coleman 2014: 1). Although not all hackers contribute to FOSS, and not all FOSS contributors would call themselves hackers (Coleman & Golub 2008), the FOSS phenomenon is firmly at the core of what has been written about the hacker ethic, and seems to have been the main inspiration for such writings (Levy 2010, Himanen 2001, Kelty 2008). Most of the ethnographic material in Kelty's work (2008) comes from FOSS projects, and Linus Torvalds, creator of the Linux kernel, is presented by Himanen (2001) as the prototypical hacker. It could be argued that what most authors have called the "Hacker Ethic" is in fact the "FOSS Ethic", or at least that both ethics share a core set of values. In what follows, I focus on those core values, the ones that hackers and FOSS contributors have in common.

When I speak of "values", I do so as defined by David Graeber: as "conceptions of what is ultimately good, proper or desirable in human life" (2001: 1). These values don't simply refer to what we want, but most importantly to what we should want: they are the "criteria by which people judge which desires they consider legitimate and worthwhile and which they do not." (Graeber 2001: 3).

The other fundamental aspect of values as defined by Graeber, which seems to be inspired by the work of Clyde Kluckhohn, is that values are not just "abstract philosophies" of life, but ideas that have "direct effects on people's actual behaviour" (Graeber 2001: 3). Our values bear some weight in the way we conduct ourselves in our daily affairs. When we must decide which course of action to take between a wealth of possibilities, our values guide the decision. According to Pekka

Himanen, this is the meaning of values in the "traditional philosophical sense: the overriding goals guiding action" (2001: 124).

In what follows, I look at two attempts to describe the FOSS values, or "Hacker Ethic", as both call it: one by Stephen Levy (2010), the other by Pekka Himanen (2001). I also highlight the aspects of the hacker ethic that have received most attention from anthropological studies, in particular its relationship to liberalism.

Finally, I propose a different framework for the hacker ethic, one that classifies its values based on the role they play in guiding FOSS contributors's actions, and in doing so make possible the FOSS development process as explained in section two.

3.1 The Hacker Ethic - Stephen Levy

The first attempt to describe the hacker ethic was probably Stephen Levy's. His book *Hackers: Heroes of the Computer Revolution*, initially published in 1984, dedicates chapter two to explain what drives the actions of those involved in the early days of hacking in American academic institutions, and the software industry that was born from them. Levy refers to these drives as a "philosophy" and an "ethic" (Levy 2010: ch. 2). Its main tenets are information freedom, decentralisation, computer programming as artistic expression and meritocracy, all in the context of technology as a means to understand the world and act upon it.

Information freedom is rooted in a concept of creativity that emphasises its evolutionary character. Creativity does not happen in isolation, and new things do not appear spontaneously: they are always the result of building upon existing creations. Thus access to the work of others is what enables creativity. In the hacker world, this means that the source code of computer programs should be freely available to everybody, to be reused and improved upon. This is a requirement in order to realise one's creative potential.

Bureaucracy, the organisational approach of centralised systems, is perceived as a barrier to information freedom. Bureaucracies create boundaries, lay obstacles that stop information from flowing into the hands of hackers. They are also based in arbitrary rules, which hackers despise as the opposite to the logical algorithms of computing. Instead of bureaucracy, hackers want to "promote decentralization" (Levy 2010: ch. 2), an organisational approach that relies on individual's impulse to fix things that are not working, and where people can follow their interests (Levy 2010). With this, Levy hints to a connection between hackers's preference for decentralised systems and their concept of work, an idea that Himanen later developed in *The Hacker Ethic* (2001).

For hackers, computer code possesses an inner beauty derived from its elegance when addressing a specific problem, and from its efficiency in the utilisation of computing resources (Levy 2010). The ability to create beauty through code is the measure by which hackers are judged, instead of superficial, external or circumstantial criteria "such as degrees, age, race, or position" (Levy 2010: ch. 2). This constitutes the hacker community as a meritocracy.

3.2 The Hacker Ethic - Pekka Himanen

In his book *The Hacker Ethic and The Spirit of the Information Age* (2001), Pekka Himanen presents the hacker ethic as as alternative to the Weberian protestant ethic found in capitalism, in particular with regards to their conceptions of work and money.

According to Himanen, there are seven values in the hacker ethic: passion, freedom, social worth, openness, activity, caring and creativity (2001).

The hacker work ethic is a combination of the first two: the freedom to pursue one's passions. While work in capitalism becomes a duty, an end in itself (Himanen 2001), and a painful obligation, for hackers it is the vehicle for realising their passion in life, that which they care about, that which they find interesting, exciting and enjoyable

(Himanen 2001). Where capitalism privileged pain and toil (Sahlins 1996), hackers privilege pleasure and joy.

Hackers's opposition to bureaucratic, centralised systems is a consequence of their concept of work as a way to pursue one's pleasure. For hackers, work is the main source of pleasure, not scarce material goods (Sahlins 1996). Since work is plentiful, self-pleasure must not necessarily come at the expense of the pleasure of others, and there is no competition involved in satisfying individual desires. While capitalism's world of pain is a world of scarcity, hackers's world of joy is a world of plenty.

The idea of centralised, bureaucratic control as a disciplinary force necessary to avoid Hobbesian destruction (Sahlins 1996) loses all meaning when confronted with the hackers notion of pleasure through work. This sets hackers apart from the mainstream Western cosmology as described by Marshall Sahlins (1996). For hackers, self-regulation is possible because a hacker's search for joy does not undermine the joy of others. And self-regulation is not only possible but desirable, because only self-regulation can guarantee the pursuit through work of whatever it is one finds interesting and enjoyable. This lays at the heart of the hacker's trust on decentralised systems and their search for freedom.

Hackers's understanding of work positions their ethics in direct opposition to the ethics of capitalism, and so does their attitude towards money. Hackers do not see money as a value or an end in itself (Himanen 2001). According to Himanen, what motivates their activity are social worth and openness (2001). Himanen's concept of openness is the same as Levy's information freedom: it refers to the requirement of sharing the outcome of one's work as a condition for enabling creativity. Himanen's concept of social worth is equivalent to Levy's "meritocratic trait" (Levy 2010: ch.2). For hackers, "recognition within a community that shares their passion is more important and more deeply satisfying than money" (Himanen 2001: 51).

Activity and caring are the values that define what Himanen calls the "nethic", or the "hackers' attitude toward networks" (Himanen 2001: 140). Himanen uses activity as an umbrella term that encompasses hackers's ideals of freedom of expression, privacy and self-determination. Caring "means concern for others as an end in itself" (Himanen 2001: 141). All these are put into practice in the hackers's active defence of the Internet as an open, decentralised, neutral, deregulated and universally accessible network. This defence of a certain conception of the Internet is what, according to Christopher Kelty, establishes them as a "recursive public" (2005). For Kelty, the Internet and its related communication channels enable what he calls "geeks", a broad group of technology knowledgeable individuals that also includes hackers, as a public. The Internet is "the condition of their association" (Kelty 2005: 185), the very thing that makes them a public. Conscious of its critical role, geeks mobilise to protect and influence "the development of the technical and legal structures of the Internet" (Kelty 2005: 193). This is the defining characteristic of a recursive public: "a shared, profound concern for the technical and legal conditions of possibility for their own association" (Kelty 2005: 185).

The last value of the hacker ethic according to Himanen is creativity, which is the "imaginative use of one's own abilities, the surprising continuous surpassing of oneself" (Himanen 2001: 141). Himanen's creativity seems to be connected to Levy's artistic expression, both an instrument for and an articulation of hackers's self-expression.

3.3 The Hacker Ethic and liberalism

Hackers's concept of work as the vehicle to realise one's passions and their disregard for money set them apart from the ethics underlaying capitalism and the Western cosmology described by Marshall Sahlins (1996). However, their ideas about freedom, openness, privacy, meritocracy and individual self-expression anchor them firmly to the Western liberal tradition. Coleman & Golub consider FOSS one of the settings in which the principles of liberalism are enacted, defined and reinvented through practice,

sometimes reinforced in their mainstream meanings and articulations, other times questioned and contested (2008).

These principles of liberalism are:

> protecting property and civil liberties, promoting individual autonomy and tolerance, securing a free press, ruling through limited government and universal law, and preserving a commitment to equal opportunity and meritocracy (Coleman & Golub 2008: 257).

Liberalism is also connected to Charles Taylor's concept of expressive self, a certain idea of subjectivity that is built upon three key concepts:

> First, that humans are capable of exteriorizing their inner selves through creative action; second, that this action is a deeply moral act; and third, that it is not enough simply for the subject to act, but that its acts must be recognized by others for them to be truly expressive of itself (Coleman & Golub 2008: 267).

FOSS as characterised above by the descriptions of the Hacker Ethic is clearly liberal in its pursuit of self-expression through the creative, artistic act of writing computer code; in its defence of individual autonomy versus centralised bureaucratic power; in its adoption of individual merit as the main criteria by which people are judged and valued; and in its quest after peer recognition. But FOSS relationship to liberal ideas of freedom of speech and private property is more contentious. First of all, FOSS contributors expand the reach of the usual definition of "speech" to include computer code. Second, the Free Software movement has been at the forefront of the resistance against copyright laws and software patents.

For FOSS contributors, "code is speech" (Coleman 2009). The Free Software Movement has been inviting people to think about freedom in software as "free speech, not free beer" since 1990 (Stallman & Lessig 2010: 247). A software program is indeed a

particular instantiation of an abstract idea, of an algorithm, a certain approach to solving a problem. As such, it can be expressed in many ways: both in natural language and in a multiplicity of programming languages. It is hard to argue that something written in any human language is not speech.

But not only writing code is speech: for hackers, running code also constitutes a type of argument. In fact, it is the definitive form of argument, "the final statement" (Kelty 2005: 201). This is because code has an objective component that speech lacks: code either runs or doesn't run, either works or it doesn't, and when working, it can work at different degrees of optimality, it can be faster or slower, better or worse. In Kelty's words: code "is the ultimate rhetorical power - particularly because it possesses the virtues of exactness and functionality that they [hackers] often claim political speech lacks" (Kelty 2005: 201). For FOSS contributors, working code settles any discussion. In fact, producing code can become a condition for speech: one's right to express an opinion becomes tied to the degree to which one contributes code, as illustrated by this exchange in the Tizen project mailing list:

> My quick research (please accept my apologies in advance if it is not accurate) shows that so far on the behalf of your account there has been submitted 0 bug reports in JIRA, 0 changes in Gerrit and I guess the same amount of code reviews. These statistics lead me to the thought that you are NOT interested in Tizen platform development, therefore, the good news for you, open source governance seems actually out of the scope of your work anyway. (Tizen General mailing list archive 2015)

The contributor writing this message claims his interlocutor has no right to criticise the project's governance model because he has not contributed to the code base in any way, neither with bug reports nor with code patches. In this particular exchange, to which the interlocutor did not reply, code became a requirement for speech.

If computer code is speech, then it falls under the same constitutional laws that protect freedom of speech in most liberal democracies, and as such, its distribution and circulation cannot be stopped (Coleman 2009): that would be tantamount to censorship. It also means that computer code is subject to copyright law, not patent law (Salin 1991). Hackers, therefore, challenge the existence of software patents; and although they are not opposed to copyright per se, they do contest its "broad interpretation" (Salin 1991) and its overprotection by laws like the Digital Millennium Copyright Act (DMCA) in the USA (Coleman 2009).

The main expression of hackers opposition to copyright laws is the development of FOSS licenses, which utilise the autonomy that copyright invests in authors not to restrict the right of others to reuse a computer program, but to encourage it, to protect it, and to guarantee its extension to any derivative works, effectively turning copyright on itself. For this neat inversion of copyright, these licenses are often referred to as "copyleft" (Coleman 2009).

3.4 An action-based framework for the Hacker Ethic

Freedom, openness, information flow, creativity, artistic expression, decentralisation, passion, self-expression, self-determination, caring, social worth, meritocracy. All these words have been thrashed out in this section in a effort to uncover the values that underpin the actions of FOSS contributors, and their relation to the broader cultural contexts of capitalism and liberalism.

Although there have been several attempts to explain each value individually, none of the above descriptions of the Hacker Ethic gives us an organising framework that can provide structure and aid understanding of how all these values fit together and interact with each other.

Hackers are women and men of action: writing computer code and building technical systems is for them the ultimate, fundamental form of intellectual argument. It seems

fitting then to construct a framework for hackers's values that uses action as the organising principle. Such a framework would split the hacker values into three categories: motivational, contextual and structural.

Motivational values are the ones that compel hackers to act: passion, a deep interest one feels, develops and nurtures regarding one or more somehow technical subjects; and self-realisation, a certain idea of subjectivity that includes a desire to express oneself through creative activity, and which is perceived as a moral act.

Contextual values set the environment, circumstances and rules under which action takes place. They are freedom, understood as self-determination, or the possibility to self-select for certain projects, technical subjects or contribution types; and openness, understood as the right to freely access and build upon the product of others's creative efforts, which in practice means the source code written by fellow contributors.

Structural values guarantee the survival of FOSS over time, both as a special kind of movement articulated mainly through practice (Kelty 2008), and as a set of specific software projects. Merit is the main structural value: the social recognition by peers of the worth of one's creative output, and of one's ability to influence and shape the future of any piece of software to which one happens to contribute.

In what follows, I attempt to explain the mechanisms by which merit keeps FOSS projects running, in spite of their lack of coercive enforcement instruments and their loose organisational structures. I will build my argument on the theoretical framework that David Graeber has called the "Chicago value theory", or the "ethnographic theory of value" (2013: 223).

4. The ethnographic theory of value

In this section, I summarise the main concepts of what David Graeber has called the "ethnographic theory of value" (2013: 223), a "volatile synthesis" (Graeber 2013: 221) of Marx, Hegel and Dumontian structuralism that starts from Marx's insight that value measures the importance of actions and not of objects (Graeber 2005).

The most important representative of this approach is Terence Turner, who contributes an alternative reading of Marx that distances itself from any "economistic reductionism" (Turner 2006: 3) and from any functionalist understanding of human society as a "self-constituting or self-reproducing system" (Turner 2006: 5).

4.1 Value theory as a counterbalance to "systemic" approaches

David Graeber speaks about value theory as representing one of the two sides in a fundamental dilemma within social theory: "how to square systemic approaches with individualistic ones" (Graeber 2005: 445). Systemic approaches are those that start by imagining society as a total system or structure, and then try to explain "how it is maintained and reproduced over time" (Graeber 2005: 445). Individualistic approaches start from individuals and their actions, and understand society as the result of the combined outcome of those actions.

In this binary conflict, value theory represents the individualistic side, which emerged from the German philosophical tradition and its idea of "culture"; while functionalism and structuralism represent the systemic side, which emerged from the Anglo-French philosophical tradition and its idea of "civilisation" (Graeber 2001).

Graeber observes that anthropology, which has managed to devise very powerful models that excel at analysing social systems as pre-existing totalities, has a tendency to prefer systemic approaches. But that such approaches end up revealing fundamental

"theoretical contradictions" that generate "a sense of crisis" (Graeber 2005: 445). It is at these times of crisis that value theory is brought back in to provide some individualistic balance to the systemic excesses.

It is hard to adopt a systemic approach in the analysis of something like FOSS, a recent phenomenon that can be traced to certain attitudes towards knowledge and creativity by individuals involved in a very specific academic field. What kind of pre-existing totality is there to imagine, when FOSS technology products and processes are so visibly generated by the combined effect of personal contributions? In spite of the fact that this dissertation explores precisely how FOSS "is maintained and reproduced over time" (Graeber 2005: 445), a systemic endeavour according to Graeber, it must do so from an individualistic perspective that does not conceive FOSS as any kind of external, pre-existing reality that is imposed over software developers, and respects the fact that FOSS is first and foremost an approach to the production of technical artifacts that emphasises the role of active and self-determining individuals. Hence the attraction of value premises as a theoretical framework for its study.

4.2 Turnerian Marxism and "Graeberian" politics

In his postscript to HAU Journal's special issue on value as theory, Graeber (2013) explains five core ideas that underpin the ethnographic theory of value as the importance of actions. The first three are derived from Terence Turner's interpretation of Marx's theory of value, and deal with the concept of production, symbolic representations of value, and society as a set of competing "imaginary totalities" for the "realization of value" (Graeber 2013: 226). The last two explain how such "imaginary totalities" interact with each other and their relationship to politics.

Turner considers conventional Marxist anthropology, with its emphasis on economistic arguments, its focus on modes of production built around the appropriation of material surplus (Graeber 2001), and its ideas of structural causation and determination, as having little relation to the most important element of Marx's thought: his theory of

value, which is a dialectical and symbolic analysis of the nature of society (Turner 2006).

For Marx, society is the result of individual productive action. Production is not only the making of commodities or material goods to fulfil perceived needs: it is also the production of people and of relationships between them. This process is reflexive and open-ended, which means it creates new perceived needs, changes the actors themselves and the way they relate to each other. This concept of production means that "social structures (...) are really just patterns of action" (Graeber 2001: 59), and that it is us, through our creative drive to transform the world around us in pursuit of certain goals, who create society.

These "patterns of action" are extraordinarily complex: when engaging in productive activity, we change ourselves, change our relationships to others, and generate new reasons to engage in additional acts of production. Individuals involved in this process tend to be aware of only one aspect of it: the goal they are trying to achieve with their productive activity. It is simply impossible for us to comprehend the sheer scale and complexity of the patterns of action that create our social structures. According to Graeber, this is the reason why, as Durkheim observed, society appears to us as an external, constraining entity (2001), something alien to us. It is this same limitation of human imagination, our inability to conceive the full extent of our combined actions, that generates what Marx called "commodity fetishism" (Graeber 2001: 65), our tendency to perceive the value of an object not as derived from the human action that created it, but as a characteristic of the object itself. Commodity fetishism, therefore, is the phenomenon by which we attribute "subjective qualities to objects" (Graeber 2001: 65). We see commodities "as having human powers and properties" (Graeber 2001: 65) when, in reality, the object has no intrinsic value: its value is the value of the human action that brings the object into being.

The second fundamental aspect of Marx's theory of value, after his concept of production, is his analysis of how the importance of human productive action is

represented by material tokens, symbols that become themselves "objects of desire" (Graeber 2013: 225) and therefore contribute to generate the same forms of productive action they represent. Such symbols turn incommensurable human productive action into comparable units that, together, make an imaginary totality (Turner 2006).

This can be better understood through Marx's analysis of money. Money was considered a measure and a medium of value: a measure because we "can use it to compare the value of different things" (Graeber 2001: 66) and a medium because we can use it as an intermediary to exchange different things. As a measure and a medium, money is a tool. Marx, however, uncovered a third aspect of money: the fact that it is not simply a tool, but a value in itself. We sell our productive action, our capacity to work, to our employers in order to obtain money; but money also represents the value of our productive action. Money not only symbolises the meaning and importance of human creative activity: it is also the object of that activity, the reason why workers perform it in the first place. So money has a kind of reflexive character: it "plays a necessary role in bringing into being the very thing it represents" (Graeber 2001: 67).

The third component of Marx's theory of value relates to its social character: value "can only be realized in other people's eyes" (Graeber 2013: 226). These "other people" is what constitutes "society" from the actor's perspective, which means that "society" is mainly an imagined construct, conceived as a coherent totality, "a kind of universe" (Graeber 2013: 226).

The social character of value means that we can identify what Marx called different "spheres" of value (Graeber 2001): there is a sphere of production, where value is created; there is a sphere of circulation, enabled by the representation of value through material tokens; and there is a sphere of realisation, which is always social in nature, and it involves someone other than the producer. The separation between the sphere of production, and the spheres of circulation and realisation, combined with the

symbolic representation of value through material tokens, make possible the accumulation of surplus value, often by people who are not the producer (Turner 2006).

According to Graeber, societies will usually have several of these imaginary totalities that enable the realisation of different values (Graeber 2013). When a society possesses several of these imaginary arenas for the realisation of value, two types of competition develop in it: an internal one, between the members of a single arena, to accrue whichever values they pursue; and a second one at the social scale, between the arenas, to impose their values as the most important within society as a whole. This, according to Graeber, is what politics is really about: not just the accumulation of value, but the definition of what value actually is, and how different values relate to each other (Graeber 2013).

The existence of several arenas for the realisation of value within a single society also means that any one individual finds herself continuously moving between these different universes. This poses the question of how we can do so without experiencing contradiction and discomfort. Graeber believes this is possible because of the imaginary nature of these arenas. Although they "propose a total view of the world" (Graeber 2013: 229), that doesn't necessarily mean the individuals participating in them believe that view of the world is actually real. They focus on their goal, in the pursuit of certain forms of value, in the same way players focus on the stakes of a game, or an audience suspends their disbelief to mainly care about whether the characters of a fictional narrative achieve or not their goals (Graeber 2013). However, particularly when there are many competing arenas, some of them might attempt to establish themselves above others by claiming they are somehow more real, or that they have "some special purchase on the nature of reality, as in the case of science or revealed religion" (Graeber 2013: 232). This claim to the truth can be understood as a political strategy, one whose goal is to turn a certain value into a "metavalue" (Graeber 2013), a value that encompasses all others and becomes a criteria by which all other values are judged.

4.3 The scope of Marx's theory of value

Turner believes that Marx's theory of value, as outlined above, is not exclusive to capitalist societies, to societies with markets or with a state: capitalism is only one example of a general tendency, present in many societies, 1) to represent the human creativity invested into production as symbolic material tokens that turn such incommensurable creativity into comparable units, 2) to think of such comparable units as part of a totality of production and 3) to employ the material tokens as media of exchange, circulation, appropriation and accumulation of value (Turner 2006).

This effectively means that it should be possible to apply Marx's theory of value to non-capitalist societies, as Turner himself does with the Kayapó of Central Brazil (2003, 2006), and by extension to any of the several imaginary arenas for the realisation of value that, according to Graeber (2005), often coexist within a single society, FOSS being one of them in the case of Western, liberal democracies.

5. A Turnerian analysis of FOSS as an arena for the realisation of value

In this section, I apply the main principles of Marx's theory of value to the analysis of the FOSS development process, following the same general structure used by Turner in his study of Kayapó villages (2006).

Like Turner, I will start with a brief outline of the main features of the FOSS social structure, which were explained in detail as part of section two. I will then proceed to identify and describe the main values pursued within the FOSS arena, and the symbolic tokens through which they are circulated and appropriated by social actors. Finally, I will explain how surplus value is generated in the context of FOSS, and how the appropriation of this surplus value by the core developers contributes to the stability of FOSS projects.

5.1 The social structure

FOSS is computer software developed by a community of contributors.

Contributors are self-selected, which means they freely choose the projects and areas in which they wish to participate, based on their personal interests. Although some FOSS developers are paid to contribute to certain projects, all contributions are volunteered. FOSS projects lack institutions and instruments to enforce contributions: it is simply impossible to impose an obligation to contribute on an individual or a corporation.

The community of contributors is loosely structured around certain software development practices. Although some projects constitute themselves as non-for-profit organisations for administrative purposes, create governing bodies or develop relatively sophisticated procedures to join the community of contributors (Coleman 2013), in practice the development process "has no central coordinating authority, no

specific lines of power and responsibility running between the constitutive elements, and no one to blame when things go wrong" (Ratto 2005: 828).

Contributions can take many forms, but the bulk of them, and the ones considered most important, involve computer code. There is usually a progression in the nature of contributions. Most volunteers start by reporting problems in the software (bugs) or improving the technical documentation. They then move onto contributing code: first for bug fixes, and progressively for improvement of existing features and the addition of new functionality.

At the top of the contribution chain are the core developers, who control access to the body of computer code (the source tree) that constitutes the canonical or "upstream" version of the software. Core developers are the ones who actually incorporate the code contributions from the community to the upstream source tree, a process referred to as "merging". Core developers are normally a very small group: having a single person merging to the source tree is not uncommon.

Code contributions take the form of "patches", small snippets of computer code presented in a standard format that highlights the changes introduced in the existing source tree. All patches are peer-reviewed: they are submitted to a public forum where contributors can look at them, appraise them and provide suggestions for improvement. Although in theory any contributor can do patch review, in practice it is carried out by experienced developers, experts on certain areas and core developers.

Becoming a core developer in a FOSS project has been described as a matter of "exceptional contributions" (Szczepanska et al. 2005: 438). Exceptional or not, such contributions must be sufficient to generate a tacit agreement within the community regarding an individual's capacity to make sound technical and non-technical decisions for the benefit of the project as a whole. This agreement on the core developers's abilities is critical to the integrity of a project. The moment it disappears, volunteers

simply stop contributing, or can decide to duplicate the source tree and continue development in their own terms, a process known as "forking".

In spite of the fact that core developers lack the ability to impose their will by force, and the ease with which software can be duplicated and turned into new versions, forking is rare (Glass 2005), and the group of core developers within projects relatively stable.

5.2 The values

The main value pursued when contributing to FOSS I will call "influence".

FOSS developers seek self-realisation through the creative expression of their view of the world in the form of computer code. Code by itself, however, is not enough: one's view of the world must be enacted, which means computer code must be part of a functional unit put to use. A developer's view of the world is realised only when the computer code that expresses it can be installed and run to perform the tasks it was designed to perform, and solve the problems is was designed to solve.

FOSS contributions become a way to achieve enactment of one's view of the world by embedding computer code into a functional unit. This ability to get code merged into an upstream source tree, code that reflects one's perspectives on how a certain aspect of the world should work, how a certain problem should be approached, and how computer code in general should be written and structured, is what I call "influence". By contributing code that is incorporated into a FOSS application, one hopes to influence the way that application is designed, to impact the way that application works, to determine the way that application evolves, so that it matches and helps realise one's vision of a problem space.

What I have called "merit" in section 3.4 is the social recognition of influence: the acknowledgement, by the members of a community of FOSS contributors, of an individual's ability to impact the shape of the software developed in common. It is

merit, as the social recognition of influence, that propels a contributor to the role of core developer, cements and maintains her position as one. Influence and merit will usually be accumulated hand in hand, but not always: influence can be exercised without an equivalent degree of social recognition, and an individual's merit can be (and is) manipulated by means that do not involve the source tree. In general, however, influence is a condition for merit: there is no core developer in any FOSS project who has not authored patches merged upstream.

5.3 The symbolic tokens

Influence is acquired when one's software patches are merged to the upstream source tree, and also through the patch review process. Patches, the small, carefully formatted, peer-reviewed snippets of computer code shared through a FOSS project's public communication channels are the symbolic material tokens that represent the contributors's creativity invested into the production of a FOSS application. The totality of production is not the overall number of patches merged to the upstream source tree, but the lines of code that make the tree. Patches manipulate and continuously modify that total body of code, adding to it, removing from it, or changing it. The patch format clearly specifies which lines are added, which ones are removed, and which ones are changed, providing some measure of the impact, or influence, a patch will exercise over the upstream code base.

Influence is not simply about adding lines of code however. In fact, patches that remove unnecessary "cruft" and address code "bloating" are highly regarded. There are also certain areas of the code that are likely to be considered more important than others, and patches contributing to those more valuable.

Attribution is also key to track influence, and consequently patch information will also include several personal signatures: those of the original author(s) and the core developer merging it upstream. Depending on the project, patches might also be signed by the submitter (the person who actually sends the patch for review to the public

communication channels), and the reviewer (the person who declares the patch fit for merging).

All the above information is produced and kept with version control tools like Git, its recording and preservation deemed necessary in order to be able to "bisect" the code and pinpoint which lines in it introduce problems. This careful tracking of the source code history is part of a marked archival tendency in FOSS (Coleman 2013, Kelty 2008). All emails sent to a project's public mailing lists, all daily conversations happening through a project's Internet Relay Chat (IRC) channels, are preserved and made publicly available, together with the detailed history provided by source control tooling. Whatever the origin of this diligent conservation of all project activity, it is not unreasonable to assume some connection to the influence and merit values.

5.4 Spheres of circulation and realisation

The patch review process constitutes the sphere of value circulation in FOSS. In this sphere, value circulates, but not through exchange as in capitalist markets. Influence is accumulated by individual contributors through merging, but generally not at the expense of others and, once accrued, it is hard to lose. This is because there is no limited pool of influence to share: influence is not scarce, there are no restrictions to the amount of contributors that can submit patches, or to the amount of patches that can be merged to a FOSS upstream tree. FOSS, as observed before, is a universe of plenty.

The value, however, is not realised when a patch is merged: a developer's influence is only realised when someone downloads, installs and uses the software to which she contributed her patches. Influence is consummated when software is put to use outside of the community of contributors, in the "real world".

So in FOSS, as in capitalist production (Graeber 2001), we find that the spheres of circulation and realisation of value are clearly distinct, in contrast to what happens with

Kayapó villages, where there is no separation between the spheres of circulation and realisation (Turner 2006). But there is a difference between the separation of spheres in FOSS and in capitalism. In capitalism, the sphere of circulation (the market) creates a vast gap between the sphere of production (the factory) and the sphere of realisation (consumption) through "the anonymity of economic transactions (...) From the perspective of those (...) using commodities, the history of how these commodities were produced is effectively invisible" (Graeber 2001: 79). This invisibility is what causes alienation, the tendency to invest commodities with human qualities, to perceive their value not as derived from the human action that created them, but as a characteristic of the commodities themselves.

Although some of the users of FOSS applications might be blissfully oblivious to how the software is produced, the sphere of circulation in FOSS does not anonymise contributions as market transactions do. In fact, it personalises them, by making sure every single patch merged upstream is always attributed to their authors, submitters, reviewers and mergers. In doing so, it reduces the degree of alienation, by explicitly recording how any piece of software is the result of the combined actions of a group of individuals, and making those records publicly available for any interested parties to see.

5.5 Surplus value

The influence accrued by the patch author is not the only influence generated when a patch is merged upstream. Some additional influence is created by the act of merging itself, and also by the patch review process. The first is appropriated by the core developer performing the merge; the second, by the contributor(s) reviewing the patch. This additional influence can be considered surplus value, since it is in excess of the influence required by the patch author to maintain her condition as a contributor (Turner 2006).

A patch can be understood as the expression of a certain point of view regarding how a specific software problem should be solved. It is, first and foremost, a reflection of the author's perspective on that software problem. When a reviewer recommends changes to the patch content, changes that are very often applied by the original author, the reviewer adds her own perspective to the code, effectively expressing her own creativity through someone else's patch. When that patch is merged, it generates influence for all the parties whose point of view the patch incorporates, both authors and reviewers.

The act of merging a patch is in itself an statement of agreement with the perspectives reflected in that patch, so every single patch a core developer merges contributes to the realisation of her vision for the software being produced.

From this emerges that core developers and patch reviewers are able to appropriate influence generated by the productive work of others, although this form of appropriation can hardly be called exploitative: once again, accruing influence does not usually happens at the expense of others, since there is not a limited pool of influence that must be shared.

However, the ability to appropriate this surplus influence and its accompanying merit, does help core developers keep their positions of leadership, contributes an explanation to the lack of forking, and sheds some light on how stability is achieved in FOSS projects.

6. Conclusion

The brief model of FOSS presented here, based on Turner's interpretation of Marx's theory of value, can only be an abstraction, a clumsy approximation to how the combined effects of the individual actions of contributors create and re-create FOSS on a continuous basis, changing not only the software being developed, but the contributors themselves and the relationships between them patch by patch, review by review, merge by merge.

The model simplifies the messy reality of FOSS projects, where the proportional relationship between influence and merit is distorted by externalities; patches submitted by different contributors enter in direct competition and grant influence to one of them at the expense of the others; and the decisions and the leadership position of core developers are continuously debated and questioned.

With all its limitations, I hope this model can somehow contribute to a better understanding of how arenas for the realisation of alternative forms of value survive within societies dominated by neoliberal ideals of wealth accumulation and market efficiency.

Bibliography

Coleman, E.G. 2009. *CODE IS SPEECH: Legal Tinkering, Expertise, and Protest among Free and Open Source Software Developers*. Cultural Anthropology, Volume 24, Issue 3, 420-454.

Coleman, E.G. 2013. *Coding Freedom*. Princeton: Princeton University Press.

Coleman, E.G. 2014. Hackers. In *The Johns Hopkins Guide to Digital Media*. Baltimore, Maryland: Johns Hopkins University Press (available on-line: http://gabriellacoleman.org/wp-content/uploads/2013/04/Coleman-Hacker-John-Hopkins-2013-Final.pdf, accessed 25 July 2015)

Coleman, E.G. and Golub, A. 2008. *Hacker practice: Moral genres and the cultural articulation of liberalism*. Anthropological Theory, Volume 8, Issue 3, 255-277.

Fitzgerald, B. 2005. Has Open Source Software a Future? In *Perspectives on Free and Open Source Software*. Cambridge, Mass.: MIT Press.

Ghosh, R.A. 2005. Understanding Free Software Developers: Findings from the FLOSS Study. In *Perspectives on Free and Open Source Software*. Cambridge, Mass.: MIT Press.

Glass, R.L. 2005. Standing in Front of the Open Source Steamroller. In *Perspectives on Free and Open Source Software*. Cambridge, Mass.: MIT Press.

Graeber, D. 2001. *Toward an anthropological theory of value*. New York: Palgrave.

Graeber, D. 2005. Value: anthropological theories of value. In *Handbook of Economic Anthropology*. J Carrier (ed). Aldershot, UK: Edward Elgar

Graeber, D. 2013. *It is value that brings universes into being*. HAU: Journal of Ethnographic Theory, Volume 3, Issue 2, 219-243.

Himanen, P. 2001. *The hacker ethic*. London: Vintage.

Kelty, C. 2005. *Geeks, Social Imaginaries, and Recursive Publics*. Cultural Anthropology, Volume 20, Issue 2, 185-214.

Kelty, C. 2008. *Two Bits*. Durham: Duke University Press.

Lerner, J. and Tirole, J. 2005. Economic Perspectives on Open Source. In *Perspectives on Free and Open Source Software*. Cambridge, Mass.: MIT Press.

Levy, S. 2010. *Hackers*. Sebastopol, CA: O'Reilly Media (citations from electronic edition in Safari Books).

Mockus, A., Fielding. R.T. and Herbsleb, J.D. 2005. Two Case Studies of Open Source Software Development: Apache and Mozilla. In *Perspectives on Free and Open Source Software*. Cambridge, Mass.: MIT Press.

Moody, G. 2001. *Rebel Code*. Cambridge, Mass.: Perseus Pub.

Ratto, M. 2005. *"Don't Fear the Penguins": Negotiating the Trans–local Space of Linux Development*. Current Anthropology, Volume 46, Issue 5, 827-834.

Raymond, E.S. 2000. *The Cathedral and the Bazaar* (available on-line: http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/index.html, accessed 12 July 2015).

Sahlins, M. 1996. *The Sadness of Sweetness: The Native Anthropology of Western Cosmology*. Current Anthropology, Volume 37, Number 3, 395-428.

Salin, P. 2015. *Freedom of Speech in Software* (available on-line: http://www.philsalin.com/patents.html, accessed 2 August 2015).

Stallman, R. and Lessig, L. 2010. *Free software, free society*. Boston, MA: Free Software Foundation.

Szczepanska, A.M., Bergquist, M., and Ljungberg, J. 2005. High Noon at OS Corral: Duels and Shoot-Outs in Open Source Discourse. In *Perspectives on Free and Open Source Software*. Cambridge, Mass.: MIT Press.

Tizen General mailing list archive, 2015 (available on-line: https://lists.tizen.org/pipermail/general/2015-July/003589.html, accessed 1st August 2015)

Turner, T. 2003. *The Beautiful and the Common: Inequalities of Value and Revolving Hierarchy among the Kayapó*. Tipití: Journal of the Society for the Anthropology of Lowland South America, Volume 1, Issue 1, 11-26.

Turner, T. 2006. *Production, Value and Exploitation in Marx and in Non-Capitalist Systems of Social Production*. Unpublished manuscript.

Wolf, E. 1990. *Distinguished Lecture: Facing Power - Old Insights, New Questions*. American Anthropologist, Volume 92, Issue 3, 586-596.

Zawinski, J. 1999. *Resignation and Postmortem* (available on-line: http://www.jwz.org/gruntle/nomo.html, accessed 12 July 2015).