Chapter 3

Free Software as Property

J. Martin Pedersen

3 Free Software as Property

> "All property relations in the past have continually been subject to historical change consequent upon the change in historical conditions" (The Communist Manifesto).
>
> "The Tao abides in non-action, Yet nothing is left undone" (Lao Tzu).

3.1 Introduction.

This final chapter is about Free Software. It will provide a detailed analysis of what Free Software is and how it came to be a global social movement. There is a clear and deeply embedded normative element to the Free Software movement in that it posits *community* as a desirable alternative to *private control* over software and information and knowledge in general. In that sense, the Free Software movement reverses the process *from* commoning *to* privatisation: commoning substitutes for privatisation on normative grounds. This normative element is articulated in the GNU General Public License, which is a legal document, more precisely a software license, that defines the Free Software community. The reversal of this process – *from* privatisation and *to* commons – is in part a struggle over software values and the dignity of hackers, suggesting that we may understand the Free Software struggle as part of the beginning of history (cf. De Angelis 2007)[58].

58 As opposed to the neoliberal idea of the "end of history".

As argued in Chapter 1, the work of the Free Software and Culture movements' leading voices can all too easily be seen as liberal apologias and as re-enforcing capitalism by providing "free labour" (Terranova 2000) and "offer liberal and neoliberal economics a refinement of its logic that does not significantly break with its overall political rationality" (Terranova 2009). The difference between resulting in a strengthening of capital or resulting in the emergence of a new mode of production, so I have argued in this essay, turns on a (mis-)conception of property. In response I will now present Free Software as a model for property.

Free Software, we may say, is an instance of *neo-commoning* that shares tendencies with the traditional commons and the neo-commons movement of the pirates, who hacked the transatlantic network of commerce, causing "a crisis in the lucrative Atlantic trade" (Rediker 2004: 9) during the Golden Age of Piracy, as noted in the Introduction, through a self-organised defence against - and alternative to - privatisation. Although Free Software most certainly is a phenomenon of "our" culture, it nevertheless constitutes novel forms of co-productive relations that challenge existing conceptions of property. Indeed, the very configuration of social relations with regard to software, or more specifically computer code, that inheres in Free Software has deliberately been shaped in an "other" way.

> "We could not establish a community of freedom in the land of proprietary software where each program had its lord. We had to build a new land in cyberspace--the free software GNU operating system, which we started writing in 1984" (Stallman 2001a).

What sets Free Software apart from the culture within which it is unfolding, in addition to the property relations novelty that I am presenting here, is that it is built in “the new land” of cyberspace. In this frontier land of opportunities – in the “liquid architecture” of cyberspace - it was, and to some extent still is, possible to do things in ways that differ significantly from conventional societal forms. Cyberspace is a permissive space, although it is increasingly enclosed, corporatised and regulated (as we saw in Chapter 1). It has permitted the Free Software movement to maintain its novel nature and grow into a successful global project, which sustains its particular social relations, based on the values of sharing and cooperating, with regard to the creation of and care for software.

While the leading voices of the movement itself do not want to understand Free Software as an instance of property, or a configuration of property relations, it is most certainly, according to the definition of property presented in this essay, an instance – a very novel and interesting instance at that – of a particular configuration of social relations with regard to things. The “things” are software and as software pervades almost every aspect of the world in which we live, embedded in all kinds of devices - car engines and brakes, flight control systems, ambulances, voting machines and of course your personal computer and the Internet, which connects people world wide - it is a very important set of social relations.

As already stated, understanding Free Software as an instance of property might not – from a strategical or tactical point of view – be useful for the Free Software Foundation and the movement it facilitates. It might well be that the future development of the Free Software movement's cause is better served by not addressing issues of property at all, so as not to contradict the basic interests of those corporations who profit substantially from

existing property relations in the tangible realm, particularly sections of the IT industry, whose collaboration the Free Software movement is in part dependent upon. In that sense I am admittedly exploiting the phenomenon of Free Software as a case study and a springboard to present not only a critique of private property, but also present an alternative approach to property configurations. However, I do think that the Free Software Foundation and their political and intellectual fellow travellers in the "Free Culture" movement are aiming too low. After all, their movements have emerged in resistance to privatisation – and they often make reference to enclosure of land and use anti-privatisation rhetoric and arguments as well. As such I think they are at best misguided and at worst misguiding their followers in the struggle against privatisation, both generally and particularly with regard to the struggle – and its viability - for Free Software and "Free Culture" in the long term. Indeed, as argued in Chapter 1, without a substantive critique of ownership in the tangible realm, the position of the Free Software and Free Culture movements remains a liberal apologia, thus harbouring an internal contradiction where privatisation is opposed yet supported in its most basic form, namely with regard to the tangible means of production. Instead of rejecting property in the intangible realm and thus implicitly supporting private property in the tangible realm, the rhetorical power of property can be made to work *against* privatisation. In other words, the power of the "framing effect" (see Section 1.3.2) can be subverted. That is one of the aims of this essay.

It is with these concerns in mind that in this chapter I will further develop the case for Free Software as a novel and potentially revolutionary instance of property.

Given that there potentially are "as many theories of property as there have been systems of property rights..." and "...that the

institution of property has had its history and that that history has not yet come to an end" (Schlatter 1951: 10), because "the meaning of property is not constant..." and the "...actual institution, and the way people see it, and hence the meaning they give to the word, all change over time" (Macpherson 1978: 1), Free Software, as a case study in property relations, is interesting. It is interesting because it forces us to see property relations in a new perspective, from the perspective of the particular social relations that characterise Free Software, and because it shows that social relations and care for and development of goods and resources can be successfully organised collectively and autonomously. In turn, the insights derived from such a conceptualisation can be used to strengthen critiques of property relations in the context of the tangible means of production and land, especially because the process of understanding Free Software as property recursively becomes a process of understanding property in a new way. It is not because Free Software *needs* property as such, rather property needs Free Software. However, a weakened private property regime is a weakened threat of enclosure: that is the central point that the Free Software movement is missing when they reject property as a useful means of social organisation.

Finally, it is also very relevant to note that "property concepts do not change without an incipient or fundamental change in the nature of the society itself" (Schurmann 1956: 507). If, then, we consider the widely accepted idea that things are changing fundamentally, that we are living on a trajectory toward a globalised village, or in a networked information society and a knowledge based economy; and if we keep in mind that profound societal changes in the past went hand in hand with the advent of new configurations of property relations, such as in the transition into capitalism, then Free Software understood as property has implications far beyond software.

However, before turning to these matters it will be necessary to introduce the "nature of code", because it is crucial to understand just how software works in order to fully grasp the significance of the Free Software principles and why the movement has emerged and grown to be so successful. That will be the task in the following section.

Upon explaining the nature of code in Section 3.2, I will present, in Section 3.3, the history and background of the Free Software movement and make a few notes on its growing economic and cultural significance[59]. In Section 3.4 I briefly present the concept of a "recursive public", before turning, in Section 3.5, to the software license at the centre of it all, namely the GNU General Public License. The GPL, as it is commonly known, will be explained in detail and with reference to copyright law and its inherent and central element of *reciprocity in perpetuity*. I will also offer an insight based on architectural metaphors in a political context (Pullan 2004), where the GPL is understood not merely as a software license, but also as a constitution of the Free Software community, which is a growing voluntary association of hackers, software developers, policy makers, politicians, activists, lobbyists that act within global civil society in the interest and for the promotion of Free Software and Free Culture in general.

Section 3.6 addresses the ways in which the Free Software movement as a recursive public has organised its own defences against violations of their self-legislative boundaries.

---

59 Unfortunately the political economy of Free Software is largely beyond the scope of this essay. In a previous draft of the PhD thesis, the political economy constituted half the work, but I developed a focus on property relations instead, because it was absent from the literature.

Let us now take a look at the nature of code.

3.2 The nature of code.

In order to understand the social, ethical, political and cultural significance of *Free Software* it is necessary to understand the technical foundations of *software in general*. That is what I call the nature of code and it involves also understanding how hardware - without which software is meaningless, useless and indeed impossible – works.

Computer hardware only understands binary code. Binary code consist of zeros and ones, referring to whether a switch is OFF (zero) or whether it is ON (one), because at the most basic level a computer is "only" a collection of switches that still largely operate on the principles defined by John von Neuman (1945) in "First Draft of a Report on the EDVAC". Essentially, the Central Processing Unit (CPU) found in computers – and many other gadgets nowadays – is simply zillions of switches squeezed into an incredibly small space.

Binary notation is not very easy for human beings to handle and that is why programming languages are crucial for the development of software, just as human (natural) languages are crucial for a conversation. Were we to communicate by way of, say, Morse coding with our eye lids our communicative capacity would be greatly limited, although, of course, should we lose the power of speech such communication would be very useful. In the same way, some very special software is sometimes written directly in binary code for specific purposes on a "low level" by specialised experts. This is the exception that proves the rule.

At the lowest level, then, computer code is binary, which is also called "object code", but for most programming purposes, in practice, it is not possible to write in binary form. A *programming language* partly solves this problem by allowing for a semantic abstraction away from this lowest, binary or object code level. Computer programming languages include algorithms and types, variables, and values ordered in so-called libraries (or collections), mainly derived from mathematics and rather far from the level of object code. The following table illustrates in simple terms the principle difference between these levels:

Binary	***Hexadecimal***	***Assembly language***	***Instruction description***
01111011	**7B**	**MOV, A, E**	**Move contents of register A to register E**

Illustration 7: Code and abstraction.

It is much easier for a human mind to write "MOV", "A" and "E" when wanting to move the contents of register A to register E, and it is much easier to remember that function in those terms than it is to remember that the binary string "01111011" instructs the computer to do so[60].

60 It should for good measure be noted that this illustration and its explanation do not actually include a high-level programming language example, but merely illustrates the principle of abstraction and the relations and usefulness for the human mind of using such abstraction. Assembly language, as a matter of fact, corresponds "one-to-one" (or directly) to the binary notation level, whereas in higher level languages a few words can compile to many more binary (object code) instructions. As such this illustrates the concept of abstraction towards natural language, but not the

Using a programming language only partly solves the problem of the difference between source and object code for the obvious reason that the source code still needs to be translated into object code.

Conceptually speaking there are generally two main forms of translating code into its binary destination. The semantically higher level of a given piece of code can be translated either by means of *interpretation*, which means that another programme sits as a translator between the programme and the hardware while the programme is running, that is when a user is *executing* it. The interpretation approach makes for a slow running programme, but might be a preferred option for testing and experimenting with code during development.

A faster option is *compilation*, which is done by a *compiler*. It is faster in terms of running the programme, once compiled, but it takes considerable time to translate or compile a programme. A compiler is itself a programme or set of programmes, which translates a given *source code* into *object code*, according to the specified environment. Once source code has been *compiled* into binary object code it cannot be translated back into its source code origins. Generally, software (whether Free or non-free) is distributed in binary form, because it is only in that form that it can be run (executed, as it were) on a computer. Thus, most commonly, when you download a computer programme, such as the Firefox web browser, it is in a binary form.

What distinguishes Free Software from non-free software is that the source code of Free Software programmes, although

---

complexity that programming languages actually entail. I provide an example of high-level programming language below.

distributed in binary, compiled form, is always made available for the public. Exactly how this works will become clear throughout the rest of the chapter.

Let us take a look at an example of a source code segment.

The excerpt (on the following page) is an example of code from the Linux kernel, which is a famous Free Software project. The text between the demarcations /* and */ are comments. The demarcations, tell the compiler to ignore whatever comments are written between them during its process of translating source code into object code (or binary form). The comments are needed for humans to better understand what the code does; what the intention of the programmer was; when and why s/he wrote it; and what ever else s/he might want to share. In this case it also includes contact information:

```
/* Tell the user who may be running in X and not see the console that we
have panic'ed. This is to distinguish panics from "real" lockups. Could in
theory send the panic message as morse, but that is left as an exercise for the
reader. And now it's done! LED and speaker morse code by Andrew
Rodland <arodland@noln.com>, with improvements based on suggestions
from linux@horizon.com and a host of others.*/

void panic_blink(char *buf)
{
static unsigned long next_jiffie = 0;
static char * bufpos = 0;
static unsigned char morse = 0;
static char state = 1;

if (!blink_setting)
return;

if (!buf)
buf="Panic lost?";

if (bufpos && time_after (next_jiffie, jiffies)) { return; /* Waiting for
something. */
```

Illustration 8: C source code

In this example, written in the high-level programming language C, we learn that someone has contributed to the kernel code by equipping it with a morsing mechanism so that the kernel can send messages to the user during extreme "panics" through LED's and the system speaker. If the kernel panics the user is likely to experience what is generally called a crash: your computer freezes, the input devices, such as mouse and keyboard, no longer function and you might have to reboot via the reset button, potentially causing data loss or perhaps even hardware damage.

Comments are important because code can sometimes be difficult to understand even for proficient programmers. In other words, 'ideas' in software are contained both in the actual code *and* in the complementary comments in which the code is wrapped; together they form what we refer to as source code. Commenting is an elementary aspect of creating software; and comments are absolutely essential for the modification of code in a complex system, which might need to be adapted to local purposes or expanded to work with novel or with more devices than initially imagined (or available).

The source code hence refers to both the composition of algorithms and types, variables, and values *and* to the commentary that the people creating and maintaining the source code write as the code base of a programme evolves. During compilation the comments are ignored and are thus not part of the object code. They are lost. Although it is theoretically possible to reverse engineer and replicate the *functions* of a programme, by snooping on the data flows going in and out of the programme, it is not possible to establish exactly how these functions were implemented, by means of exactly what algorithms and so on. Certainly the comments are lost entirely and it is also possible to write and compile code in such a manner that it is even more difficult to reverse engineer.

Access to the code, then, is necessary to understand any given software programme fully, to customise it for local, specific needs and to repair it.

Therefore, the functionality of complex systems (from a single desktop computer to networked systems controlling nuclear power stations, airports, trains and ambulances) can only be analysed in depth if there is access to the source code of the software that makes it run.

If this obvious need of access to source code in order to analyse it is disregarded for whatever reason then we can speak of a process of knowingly designing insecurity and creating a *black box technology*. Software without access to the source code is a product where a public peer-review is impossible and the resulting software is *non-free* software[61]. It is, in part, for these very reasons that the Free Software license, the GPL, stipulates that all source code must be available to the public for scrutiny.

As software increasingly pervades all aspects of technology and social life the question concerning access to the source code – or not – is of increasing and alarming importance. The ubiquitous presence and ever increasing importance of computers for all kinds of social relations call for such public scrutiny options and the accountability that Free Software makes possible and advocates. Given the intimate relation between a computer and human users further stresses the extreme importance of access to the source code in order to facilitate public scrutiny and, in the widest sense, to facilitate a democratisation of technology. If the future of the networked information society is shaped by technologies of which only a few corporate programmers, subject to non-disclosure agreements, know the actual internal functioning, the future of technology is a future of unnecessary uncertainties, whereas if the networked information society's underlying technology is based on Free Software and Free Software derived principles of openness and freedom, then uncertainties are kept at a minimum. That is why a social movement for software freedom and reform of those intellectual property laws that regulate software and other production of

61 Some quotes will be used in which non-free software is mistakenly labelled *non-proprietary* software.

cultural artefacts has emerged and continues to grow and act in the lobbies of public policy making institutions.

Let us take a look at this movement.

3.3 A brief history of Free Software and its imaginary, scientific and cultural origins.

I want to first take note of the way in which computer science – and software as such – is embedded in the scientific commons. Software is not possible without the common scientific knowledge upon which it rests. I will also suggest that the idea of creating programmable devices has been part of the collective imagination across eras and civilisations.

Moreover, as science and technology, as well as social science, increasingly utilises software for modelling and calculating matters, software becomes a crucial element in the advance of science, technology and social science. In the same way as public roads are needed for market relations, so is software needed for many activities associated with public goods. In Section 3.3.2 the specific history of the Free Software movement is briefly presented.

3.3.1 Embedded in the scientific commons.

Computer science has a peculiar history, because it cannot be separated from the (other) scientific traditions upon which it rests. Computer science is at once connected to ancient history, yet stands as a symbol of an advanced, high technology society. In order to programme a computer – that is to write computer

code in a programming language, as already suggested above – it is necessary to draw upon various of the principle branches of mathematics for the purpose of logical reasoning and quantitative calculation, as well as generating graphical representations of what is being calculated. For instance, drawing a circle, or part of one, on a computer screen involves knowledge and principles that, as far as is known today, began to be established by Sumerian mathematicians (3000 - 2300 BCE) and were perfected by Pythagoras and his followers approximately 500 BCE. The equation with which to calculate the circumference of a circle ($C=2\pi r$) and its derivations are thus central to generating the graphical representations that make your computer usable for such things as browsing the Internet or, indeed, writing a thesis.

Computer science brings together a lot of established scientific knowledge from different eras, cultures and traditions and, recursively, as a tool for the advancement of most sciences, whether natural or social, it feeds back into those scientific systems (of thought). Most social scientific quantitative research involves the use of computers and the design of human-computer interfaces draws upon the social sciences and humanities. Notable in this context is the pioneering work of Lucy Suchman at Xerox's Palo Alto Research Center (1979-2000), collected in “Plans and Situated Actions: The Problem of Human-machine Communication” (1987) and Vernon Pratt's “Thinking Machines: Evolution of Artificial Intelligence” (1987).

The history of programmable machines is surprisingly old. During the Islamic Golden Age, al-Jazari (1136 - 1206)[62], a polymath, published a “Book of Knowledge of Ingenious

62 Full name: Abū al-'Iz Ibn Ismā'īl ibn al-Razāz al-Jazarī

Mechanical Devices", with which modern application of science to mechanics began to take form:

> "We see for the first time in al-Jazari's work several concepts important for both design and construction: the lamination of timber to minimize warping, the static balancing of wheels, the use of wooden templates (a kind of pattern), the use of paper models to establish designs, the calibration of orifices, the grinding of the seats and plugs of valves together with emery powder to obtain a watertight fit, and the casting of metals in closed mold boxes with sand" (Hill 1991: 64).

It was not only basic mechanical applications, however, that al-Jazari championed. Noel Sharkey at University of Sheffield has replicated one of al-Jazari's remarkable devices, speculating that this might have been a programmable automaton, pre-dating Leonardo's automaton, hitherto considered the first programmable machine. One of the many amazing automata that al-Jazari devised was "a boat with four automatic musicians that floated on a lake to entertain guests at royal drinking parties. It had two drummers, a harpist and a flautist". The heart of Sharkey's replica "is a rotating cylindrical beam with pegs (cams) protruding from it. These just bump into little levers that operate the percussion. The point of the model is to demonstrate that the drummer can be made to play different rhythms and different drum patterns if the pegs are moved around. In other words it is a programmable drum machine" (University of Sheffield n.d.).

Particularly noteworthy, apart from the fact that the programming of machines is nothing very new, is that the idea and the imagination of programmable machines and automata go even further back in history, stretching into ancient myths. The Greek

god Hephaestus, the “divine blacksmith, the artisan-god, the demi-urge who has created admirable works and taught men the mechanical arts”, from whom Prometheus stole the (technology of) fire – and who created Pandora as humankind's punishment for that theft - also devised programmable automata to assist in his workshop. The box of evils and hope had been opened. Most famously Hephaestus constructed and programmed Talos, the giant “man” of bronze, a robot that is, “whose duty it was to guard the Cretan tree and prevent its being approached” (Aldington and Ames 1972: 126). It is curious to note that Hephaestus was born as a cripple and thus did not possess the full level of mobility that the other gods and the humans did. Was that why he “naturally” became the god of creating things for overcoming “human” limitations and replicating human capacity, bringing at once evils and hope? At any rate, al-Jazari, we may say, stood on the shoulders of Talos the giant when he created his programmable automata and in turn figures like James Watt (1736 – 1819) and Charles Babbage (1791 - 1871), the conceptualiser of what can definitively be considered a programmable computer, and Ada Lovelace (1815 - 1852), the first “programmer” (of Babbage's non-existent machine), stood on the shoulders of al-Jazari.

It is equally instructive to consider the work of Frances Yates. In The Art of Memory (1966) and Theatre of the World (1969) Yates traced the conceptual history of techniques and arts of memory in the workings of the architectural, poetic, rhetorical, theatrical and occult imaginations across cultures and time. She thus provided an analytical narrative of (dis)continuities ranging from the associative memory structuration of the Greek poet Simonides, through the neo-platonic memory theatre of hermetic philosopher Giulio Camillo and medieval cathedrals, to the occult magic of Giordano Bruno, heralding the modern concept of “scientific method”:

> "It is a curious and significant fact that the art of memory is known and discussed in the seventeenth century not only by ... [those] ... still following the Renaissance tradition, but also by the thinkers who are turning in the new directions, by Francis Bacon, by Descartes, by Leibniz. For in this century the art of memory underwent yet another of its transformations, turning from a method of memorizing the encyclopaedia of knowledge, of reflecting the world in memory, to an aid for investigating the encyclopaedia and the world with the object of discovering new knowledge. It is fascinating to watch how, in the trends of the new century, the art of memory survives as a factor in the growth of scientific method" (Yates 1966: 355).

The history of the art of memory is a history of the concepts without which it would not be possible to imagine the kind of digital computers that we know today. Yates, in an astute aside, notes that this history of storage, retrieval and manipulation of information for the purpose of organising forms of and access to knowledge might provide useful insights for the development of the digital computer. Indeed. This is not only the case for the internal workings of the digital computer, where data is stored with reference to its storage location – similar to the associative memory of Simonides – but also the conceptual order of the graphical user interface, which for most practical purposes is the way that most people know, recognise and use a digital computer. The graphical user interface, like the art of memory, uses icons in specific loci to refer to specific information and knowledge stored elsewhere (beyond the visible field of the computer user). This point was picked up on by Nicholas Negroponte (1995) and

later Peter Matussek (1999) in the context of the "invention" of the graphical user interface of contemporary computers:

> "This new interface put to new use an old insight of the Roman rhetoric manuals – namely, that the highest degree of mnemonic efficiency is exhibited by techniques involving topographical arrangements of mental images (loci et imagines). That the use of image-based technology might have involved an actual historical reprise in the computer age was explicitly reflected already by the Architecture Machine Group who developed the Spatial Data Management System during the seventies." (ibid.)

Software makes computers work. It controls the CPU and makes communication possible between the various hardware entities that make up a computer, but it also structures the graphical user interface. As the term suggests an inter-face is a two way system: accessing the underlying, lower level command structures and machine instructions through pointing and clicking (and writing) in the two-dimensional graphical interface *and* very importantly, receiving the return of the requested computations shaped in that very fashion. The interface thus structures both access to and computed returns from the digital magic realm that only specialist low level programmers could otherwise understand. How we create this interface, the principles, known and unknown, that are at play to quite some extent define the boundaries of the novel epistemological terrain of cyberspace. By extension, without access to the source code, the minds of people in a "networked information society" are shaped by black-box technologies: if there is no access to the source code, we cannot know exactly how we are interfacing with our

computers, with cyberspace and with other people through digital media[63].

> "Any time you engage with information, the reality that you extract from that information is shaped by the tools that deliver it. Microsoft's information presentation is such a monoculture that it edits out a lot of other realities. So you have a new kind of monopoly that affects the way people think in ways that are invisible to them. It's a very dangerous form of monopoly, especially now that they are talking about the "trusted computing" model, where it will be very difficult for you to save and then pass on documents on systems without identifying yourself … That system is supposed to be designed to help control digital rights management. By its nature it will be great for political rights management, because it's an enormously penetrative surveillance tool, and it makes it hard to do anything anonymously involving a computer. Here is a monopoly in essence, the Wintel monopoly -- Windows/Intel -- which has enormous global power and which no government is willing to stand up to, at least effectively, so far" (Barlow in Doherty 2004).

---

63 Beyond the scope of this essay, these aspects of software and software freedom might be related to Article 19 of the Universal Declaration of Human Rights (and related declared rights of the freedom of thought and communication): "Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers."

The problem of software as a black-box is not limited to the graphical interface, of course, but even more so pertains to the core of any given programme. The file sharing programme called Kazaa, whose developers were later to create Skype, was a Trojan Horse that once installed on your computer tracks your computer use and Internet surfing habits for the purpose of targeted advertising and collection of such data in general. The code segments included in a programme for such purposes are called Spyware or Malware. When uninstalled, Kazaa leaves the Malware behind and a third-party programme called "KazaaBegone" (Merijn n.d.) is required to purge your computer of unwanted, snooping code. Skype also has functions that turn your computer into a "super node" on the Skype network without your knowledge, unless you have informed yourself and found out how that can be avoided. Bev Harris, founder of Blackboxvoting.org and author of "Black Box Voting: Ballot Tampering in the 21st Century", has done a lot of work to expose the problems of software that cannot be scrutinised in public. In particular, she has drawn attention to Diebold Election Systems, a company with strong ties to powerful political factions. Journalistic investigations have revealed what becomes possible if democracy is processed through black box technology:

> "Following the 2003 California election, an audit of the company revealed that Diebold Election Systems voting machines installed uncertified software in all 17 counties using its equipment" (Fitrakis 2004).

The inscrutability of the software system of these machines made voters in the U.S dependent on "third-party" monitoring bodies:

> "Like Ohio, the State of Maryland was disturbed by the potential for massive electronic voter fraud. The voters of that state were reassured when the state hired SAIC to monitor Diebold's system. SAIC's former CEO is Admiral Bill Owens. Owens served as a military aide to both Vice President Dick Cheney and former Defense Secretary Frank Carlucci, who now works with George H.W. Bush at the controversial Carlyle Group. Robert Gates, former CIA Director and close friend of the Bush family, also served on the SAIC Board" (ibid.).

This vicious cycle of technological fraud and control would be severely minimised, or even eliminated, if the voting machines – should they be necessary at all – were run on Free Software that could be assessed by the public. In more general terms:

> "Exclusive access to the how of storytelling lets a storyteller monopolise the what ... [A] television program or commercial holds us in its spell as much through the magic of broadcasting technology as its script. Whoever has power to get inside that magic box has the power to write the story we end up believing" (Rushkoff 2004: 21).

Computer science through its application as information technology today is central to the workings of many scientific disciplines, social organisation and leisurely pleasures. On that basis there is a good ethical and social case to be made for Free Software based implementations, rather than black-box technologies. Keeping the knowledge and science behind one of contemporary times most central technologies as business secrets seems to me to be a dangerous route for knowledge and

development. Especially taking into consideration that there are good claims and arguments that Free Software develops faster and is more versatile than its counter-intuitive counterpart, non-free software. Moreover, with importance far beyond software, Free Software is a paradigmatic case of getting "inside that magic box" and thus begin revealing the technological foundations of the global village.

3.3.2 A brief history of the Free Software movement's resistance to privatisation.

The history of digital computing in recent decades has been well documented (Ceruzzi 2003 is a good starting point) and the history of Free Software and hackers has been the topic of historical investigation from the early days (e.g. Levy 1984).

The software commons, as we may call the hackers' community, enjoyed a glorious, but relatively brief initial period of success.

> "When I started working at the MIT Artificial Intelligence Lab in 1971, I became part of a software-sharing community that had existed for many years. Sharing of software was not limited to our particular community; it is as old as computers, just as sharing of recipes is as old as cooking. But we did it more than most … We did not call our software "free software", because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source

> code, so that you could read it, change it, or cannibalize parts of it to make a new program" (Stallman 1998).

However, in 1976 William Henry Gates the Third publicly began his project to enclose software and being a corporate lawyer's son with a keen sense of business and the capacity to speak in public policy lobbies, Bill Gates as he is commonly known, was to become very successful at that. His "Open Letter to Hobbyists", dated February 3 that year, addressed the community of computer hobbyists who were copying and sharing software in order to be able to use their computers for fun and for a wide variety of projects. By calling this practice of sharing theft and those practising it thieves, combined with lobbying for extensions of so-called Intellectual Property Rights law to include software, Gates divided and conquered the emerging culture of computer use so successfully that on that basis he was to become the richest man in the world (Mames and Andrews 1994). In the early 1980s one hacker felt that privatisation was so severely threatening the hacker community of sharing and cooperating that action had to be taken. This man was Richard M. Stallman, who later became the founder of the Free Software Foundation (and thereby of the Free Software movement). Stallman is the author of the GNU Manifesto (1985) and the GNU General Public License (1989) and he here describes a moment of significance in the process of enclosure of hacker culture:

> "The situation changed drastically in the early 1980s … The AI lab hacker community ... collapsed … In 1981, the spin-off company Symbolics had hired away nearly all of the hackers from the AI lab, and the depopulated community was unable to maintain itself … The modern computers of the era, such as

> the VAX or the 68020, had their own operating systems, but none of them were free software: you had to sign a nondisclosure agreement even to get an executable copy. This meant that the first step in using a computer was to promise not to help your neighbor. A cooperating community was forbidden. The rule made by the owners of proprietary software was, "If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them."" (Stallman 1998).

The social values of sharing and cooperating had – without articulation – governed the software commons of the hackers. Stallman was very perceptive of exactly that; and the experience of the loss of the community that was sharing those values was what drove him to recreate a community where these values could thrive. This time they were to be (legally) articulated in order to clearly define that community and its boundaries. Like the Magna Carta and the Charter of the Forests articulated already existing and, by the commoners, practised values once they came under threat, so did the GNU General Public License (GPL) articulate the already existing and practised values of the hacker community. It is in precisely this sense that I suggestively call the Free Software movement a community of neo-commoners, because it opposes the powers of privatisation and enclosure.

> "The basic idea of the free software movement is that nobody should have such power over anyone else. Users deserve freedom, so software should be free. Thus, proprietary software is something worse than an inconvenience. Proprietary software is a social problem, and our aim is to put an end to it. Free software is sometimes more powerful and

> reliable, but what concerns us most is that it is a more ethical way to distribute software" (Stallman in Biancuzzi 2009).

Stallman's project was to create a free operating system, written from scratch, and protected in such as way that it could never be enclosed. This "new land in cyberspace--the free software GNU operating system" (Stallman 2001a) was the beginning of a remarkable history. The idea was to create a UNIX-like system. With the same architecture and based on the same principles, but without code that was exclusively owned. In 1987 Stallman released the first version of the GNU C compiler. C is a programming language and the GNU C compiler, obviously, can compile C source code into binary code. A compiler is necessary to create all the other programmes that make up a complete operating system and as such the GNU C Compiler was a milestone in the process towards an entirely free operating system. The GNU C Compiler has since been greatly extended and is now known as the GNU Compiler Collection, thus maintaining the same acronym: gcc.

Until the 1990s, however, the GNU operating system still lacked a so-called kernel, which is the core of an operating system, which acts as a mediator between programmes (or applications) and the hardware level of the computer. The Free Software Foundation's attempt to write a kernel for GNU, called GNU Hurd, has been wrought with difficulties and has never materialised in a form that has been widely used. Things changed for GNU in spring 1991 when Linus Torvalds, a keen Finnish student interested in computers, began writing a kernel that he called Linux. Soon thousands of people joined him in developing the Linux kernel – and as his code was released under the GPL, a whole community rapidly grew around it. Torvalds here describes the initial conditions:

> "I had taken a course in UNIX and C, the fall semester before. The first time I actually touched UNIX was fall 1990, when I had a UNIX course at Helsinki University. Actually, it was the first UNIX course they ever had at Helsinki University, because it used to be a VAX and VMS place. They had just gotten a UNIX machine for trying out that newfangled thing, and it turned out to be a huge success. Within a few years, they had switched over everything to UNIX. But that first machine was used for this small course in UNIX and C, and I immediately felt that this was what I wanted to have. It made sense. Then when I bought a PC, I wanted UNIX on it, and the rest is kind of history" (Torvalds in Richardson 1999).

The history has been tremendously successful. The combination of the incomplete GNU operating system, especially the GNU C Compiler, and the Linux kernel, compiled by the GNU C Compiler, became the GNU/Linux operating system, which is now widely used in a wide variety of so-called distributions and by millions of people and many large companies around the world.

A distribution is an operating system: a collection of thousands of libraries and applications put together by companies for profit or by voluntary associations for the greater good. There are hundreds of GNU/Linux distributions available for free download on the Internet[64]. In 1995 the Apache ("a patchy") web server, named after the many patches contributed by a

64 The best overview is provided by http://distrowatch.com/

geographically widely dispersed community, was released under a GPL inspired and compatible license, called the Apache License. The Apache web server has been the most popular web server since 1996 and is currently, November 2009, run on 55.32% of the world's web servers. It is followed by Microsoft server products, which maintain 18.98% of the market share of active web servers (Netcraft 2009a).

Then came a wide variety of freely available web oriented scripting and programming languages that extended functionality of existing web building tools and made it possible to build very complex sites. Fused in the way that a distribution is, entire Content Management Systems (CMS) began to emerge, for instance Drupal, released under the GPL in 2001. In November 2009, the White House moved its website to a Drupal CMS as part of its promotion and support of Free Software (Netcraft 2009b). With these Free Software tools it has been possible for years now to build an entire web server and complex web sites based entirely on Free Software. Likewise, it is possible to surf the web, write texts, create and modify images, and a thousand other things on a computer run entirely on Free Software. Commerce built on Free Software is by now a multi-billion dollar industry, led by IBM. Many companies are developing Free Software around which they have created a portfolio of services, such as support, as a business model.

It is the principled stance of the Free Software Foundation that has made this possible, because the Free Software “ecology” has grown due to the protection measures articulated in the GNU General Public License. The GPL defines a defence against enclosure, as we shall see below.

However, as is common in social movements, it came to political differences over these principles of defence. Some key players in

the Free Software movement did not want to be *neo-commoners* with social and political aims, but merely wanted to derive an engineering methodology from the principles of Free Software. On these grounds the Free Software movement in the late 1990s split in two.

Within the movement a faction had emerged that did not consider the social and political aims of Free Software as important, indeed they considered the principled stance of the Free Software Foundation as a hindrance to marketing Free Software to the IT industry. What they wanted to promote was merely the concept of open access to source code, thus limiting their focus to the engineering methodology of Free Software. It gave rise to the establishment of the Open Source Initiative (OSI), which "respect the four freedoms [that define Free Software, as we shall see below] but they don't defend the four freedoms" (Stallman 2007). While the Free Software movement is based on a socio-political principle articulated in the GPL, the OSI only promotes a method of development. In great part the OSI "business people" based their initiative on a rejection of the term "free", which they considered harmful for the acceptance in the business world of Free Software. This limitation is also recognised by the Free Software Foundation, but they insist on the term, because of the way in which it invokes the notion of right and refers to rights discourses. A "free man" or "free woman" lives in a "free society", and a "free society" has "free software".

> "The term "free software" is prone to misinterpretation: an unintended meaning, "software you can get for zero price," fits the term just as well as the intended meaning, "software which gives the user certain freedoms." We address this problem by publishing the definition of free software, and by saying "Think of 'free speech,' not 'free beer.'" This

> is not a perfect solution; it cannot completely eliminate the problem. An unambiguous and correct term would be better, if it didn't present other problems … Every proposed replacement for “free software” has some kind of semantic problem—and this includes “open source software” (Stallman 2007).

The visions of freedom were always integral to the Free Software movement:

> “I designed the GNU GPL to uphold and defend the freedoms that define free software--to use the words of 1776, it establishes them as inalienable rights for programs released under the GPL. It ensures that you have the freedom to study, change, and redistribute the program, by saying that nobody is authorized to take these freedoms away from you by redistributing the program under a restrictive license” (Stallman 2001a).

Stallman had explicitly been using rights language and libertarian philosophy in the (U.S.) American way as a means to protect the fragments of the hacker community, a voluntary association of individuals exercising their freedoms of speech and assembly, which by the early 1980s began to feel the effect of primitive accumulation or market expansion. The customs of the hacker community were under threat by privatisation and in this way the Free Software movement is a social movement that share history with other social movements to secure civil liberties to protect existing customary, communal practices. The GPL “enshrine[s] a sort of customary law or act as a declaration of customs within hackerdom” as socio-legal scholar Maureen O'Sullivan puts it,

and the "...preamble of the GNU GPL, in particular, employs a style of language richly reminiscent of the often countered "We the People..." sections from the constitutions of many nations" (2005).

Bruce Perens is one of the co-founders of the Open Source Initiative, together with Eric Raymond. Perens is a key Free Software programmer and is the author of the Debian Social Contract and Debian Free Software Guidelines65 upon which the Open Source Definition is based, and which he co-wrote. Not long after articulating it, Perens realised that the enhanced marketability and commercial palatability gained by discarding the term Free - and thus the reference to and socio-political struggle for principled (software) freedom – came at the cost of the protection of the values upon which the Free Software Foundation stood strong. In 1999, "around a year" after the split created by the Open Source Initiative, Perens posted an often quoted email with the title "It's Time to Talk About Free Software Again", in which he stated that:

> "Open Source has de-emphasized the importance of the freedoms involved in Free Software. It's time for us to fix that. We must make it clear to the world that those freedoms are still important, and that software such as Linux would not be around without them ... Sadly, as I've tended toward promotion of Free Software rather than Open Source, Eric Raymond seems to be losing his free software focus. The Open Source certification mark has already been abused in ways I find unconscionable and that

65 Two important Free Software manifestos, which helped define the movement by declaring certain principles, terms and aims.

> I will not abide. I fear that the Open Source Initiative is drifting away from the Free Software values with which we originally created it" (1999).

However, the phenomenon of Free Software is now best known to people by the name of Open Source, which hides the social and political aspects of freedom from view. We can of course never know how far the Free Software movement would have reached into the public imagination without the marketing trick of the Open Source business people.

Linus Torvalds whose project has benefited very well from the principles of Free Software - "[m]aking Linux GPL'd was definitely the best thing I ever did" (Torvalds n.d.) - as a paradox stands as the opposing voice to Stallman's ideological voice. Torvalds is "absolutely uninterested in politics" (Torvalds in Richardson 1999). OSI co-founder Eric Raymond is even more explicit:

> "[I]n the battle we are fighting now, ideology is just a handicap. We need to be making arguments based on economics and development processes and expected return" (Raymond 1998).

Promoting an engineering standard on the basis of economic *short-term* incentives stands in strong contrast to the *long-term* social goals of Free Software. Time and again Stallman, in essays, interviews and talks, raises awareness of this crucial distinction. In his essay "The GNU GPL and the American Way" he states:

> "The Open Source Movement, which was launched in 1998, aims to develop powerful, reliable software and improved technology, by inviting the public to collaborate in software development. Many developers in that movement use the GNU GPL, and they are welcome to use it. But the ideas and logic of the GPL cannot be found in the Open Source Movement. They stem from the deeper goals and values of the Free Software Movement. The Free Software Movement was founded in 1984, but its inspiration comes from the ideals of 1776: freedom, community, and voluntary cooperation. This is what leads to free enterprise, to free speech, and to free software" (Stallman 2001a).

Reflecting expressly the view on Free Software that I am outlining here, Stallman, in "Why Open Source misses the point of Free Software", writes:

> "Nearly all open source software is free software. The two terms describe almost the same category of software, but they stand for views based on fundamentally different values. Open source is a development methodology; free software is a social movement. For the free software movement, free software is an ethical imperative, because only free software respects the users' freedom. By contrast, the philosophy of open source considers issues in terms of how to make software "better"—in a practical sense only. It says that non-free software is an inferior solution to the practical problem at hand. For the free software movement, however, non-free

> software is a social problem, and the solution is to stop using it and move to free software" (Stallman 2007)

Students of social movements will be familiar with this kind of split. In "conventional" social movements this split is often, colloquially, explained in superficial terms as the difference between "revolution" and ·reform". In a "Strategy for Labour", Andre Gorz (1964), made a distinction between (a) *reformist reforms* that strengthen the underlying logic, institutions and legitimacy of prevailing power relations, and (b) *non-reformist reforms* that undermine the logic, institutions and legitimacy of power, thus opening possibilities of deeper change. Gorz's distinction helps explain the difference between Free Software and Open Source: the former is a social and political movement that seeks to "undermine the logic, institutions and legitimacy of power" by advocating fundamental reform of mainly copyright and patent law. The latter is a trademark for a network of programmers, who prefer and consider superior software which provides access to the source code without addressing the "underlying logic, institutions and legitimacy of prevailing power relations". If the Free Software commons is disembodied, as I argue, then Open Source is no commons at all.

Despite these differences in policy – one faction being somewhat stripped of social and political values – the two sides continue to work with a shared aim: the advance of software with access to the source code. Open Source is a concept that has been adopted by large sectors of the IT industry and beyond the world of software, while Free Software principles and politics continue to influence a wide variety of activities, equally not limited to software.

However, it is not a synergistic relationship only. Deviation from the original principles has given rise to a proliferation of licenses that are making it difficult for developers and businesses to decide on a particular license. When licenses are not entirely compatible with one another it does not strengthen the original Free Software based software commons, but establishes *several* software commons. Perens has long since realised that he made mistakes, not only when promoting Open Source over Free Software, but particularly in the context of the proliferation of licenses contingent upon splits in the movement:

> "[T]he fact that there are 73 licenses is a problem. Many of those licenses are incompatible with each other. To understand the legal implications of mixing software under two of those licenses together in the same program, you'd have to learn 5256 different combinations! … And the worst thing about this is, it's my fault! Well, partially. When I wrote the rules for Open Source licensing in 1997, as a policy document of the Debian project, not many people took what we then called "Free Software" seriously, and it was unthinkable that 73 different licenses that complied with my Open Source Definition would ever be written" (Perens 2009)

The complexity that Perens here points to and the subtle – or not – differences between the respective licenses are beyond the scope of this essay; indeed, undertaking such as task as to map out these differences would require an essay of its own. We must maintain a focus on Free Software in broader philosophical terms, rather than a specialised, detailed analysis of licenses. However, it is necessary to be aware of these differences in

general terms. This figure shows in a simple way the complexity arising from different categories of software (FSF 1996):

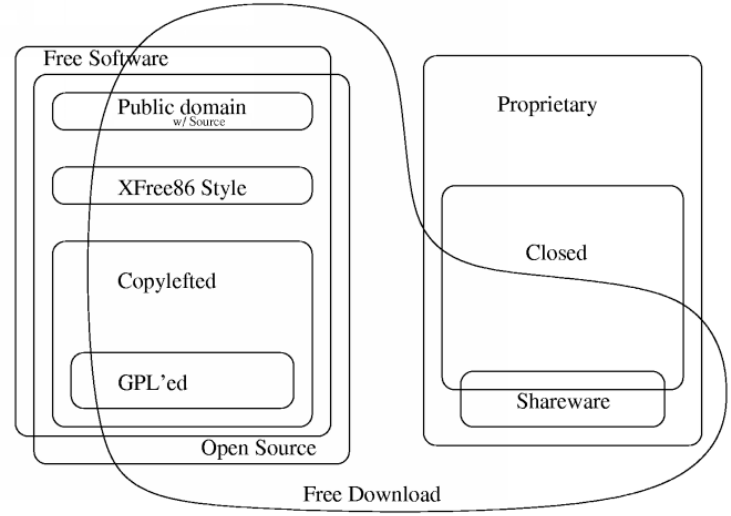

Illustration 9: Software categories

Apart from providing an overview of the complexity of categories of software, this figure also shows us, as Stallman noted above, that software released under the GPL falls within the category "Open Source". Because the most widely used license is the GPL and because one of the most famous Open Source projects, namely the Linux kernel, is released under the GPL, the political division between Free Software and Open Source is even more complex than suggested above by Stallman and the relation between these factions reveals a peculiar aspect. While the Open Source movement tends to depoliticise Free Software, the most commonly used Open Source license is the GPL, which remains unaltered and thus, essentially, in a stealthy manner, we may say, still advances the cause of the Free

Software movement. In a sense, the de-politicisation – insofar as an Open Source project, which most do, choose the GPL as a license - remains superficial. That is because the principles are enshrined in the license and the associated code will always be accessible for Free Software commoners. An Open Source project released under the GPL remains a vehicle of Free Software principles and the code that it contains enters the structured, Free Software commons, but these underlying principles of freedom are hidden from view when the majority of users - and the public in general - only recognises the given piece of software as “open” and not “free”.

This historical outline with a view to certain underlying social and political principles does little justice to the rich history of a fast growing movement, but it should provide the reader with sufficient knowledge required to understand the specificities of the Free Software phenomenon and the software license that articulate the movement's social values, particularly in the context of copyright and property relations, to which I turn in Section 3.5. Before a presentation and analysis of the GPL, however, I want to further contextualise Free Software in socio-political and cultural terms. I do so by way of the concept of a “recursive public”.

3.4 The Free Software movement as a recursive public.

Christopher Kelty (2008) has conceptualised the phenomenon of Free Software and the cultural significance of Free Software in terms of his concept of a “recursive public”. The choice of the term recursive is obvious in the context of software, because the concept of recursion is a basic and very central aspect of computer programming. In other sciences it is also, relatedly,

known as a “feedback loop”, which in simple terms means that the output of a process becomes an input into that same (on-going) process. Recursion separates simple programmable devices from what we know as digital computers. It is a term and concept that is central to the imaginary of hackers and formed the basis for the naming of the GNU operating system: GNU is a recursive acronym that means “GNU's Not Unix”. In this case it is a humorous wordplay typical of the hacker community, but Kelty takes it to a serious social scientific level in his conception of the Free Software community as a “recursive public”. Likewise, I used the concept of recursion, in the Introduction and Chapter 1, to argue that the tangible/intangible divide as a fundamentally distinguishing factor in the configuration of property relations is misleading, because there is a recursive relation between goods and resources in these respective realms.

One of the first steps that Kelty makes in his narrative is to clarify the relation between Free Software and the Internet, which is also recursive, as already suggested above:

> “The Internet is a unique platform - an environment or an infrastructure - for Free Software. But the Internet looks the way it does because of Free Software. Free Software and the Internet are related like figure and ground or like system and environment; neither are stable or unchanging in and of themselves, and there are a number of practical, technical, and historical places where the two are essentially indistinguishable” (Kelty 2008: 4).

The visions of freedom of information, speech and circulation of knowledge that are intrinsic to the Free Software movement were also clearly in the mind of Tim Berners-Lee when he developed

the Hyper Text Transfer Protocol, which is the technical aspect of the World Wide Web and which, together with email, have come to define the Internet. Berners-Lee imagined that the World Wide Web could connect all computers in the world and so provide access to all information in digital existence. Providing the tools for access and providing free and open access to the scrutiny of such tools was central to his vision (Berners-Lee 1999) – and reflected the utopian visions of technology that have been central to social movements in cyberspace for decades (Turner 2006).

Although the Free Software Foundation existed before the World Wide Web, it only grew slowly, in part because the programmes that they distributed were on recorded magnetic tapes that were sent with conventional mail companies; and it is with the circulation of the programmes, and the sharing of code, that the values of the community are perpetuated and thus that the community grows. However, these programmes and the vision of sharing and cooperating behind them, helped give shape to the Internet, which in turn provided a framework for distributing Free Software more smoothly and infinitely faster. Offering code on a website for download allows anyone, anywhere – who has Internet access and the hard- and software required to do so - the possibility to download the programme in question.

One dimension of the recursive nature of the Free Software public can thus already be found in this relation. One of the outputs of the Free Software movement in the early days were the programmes that came to define what we know as the Internet, and the Internet, in turn, became an important input in the development of the Free Software movement. This is an important relation that shows how openness and freedom perpetuate themselves. They come around if they go around, so to speak. It also shows how crucial Free Software - in practice (through provision of software tools) and in theory (through

defining and perpetuating visions of freedom and openness) – was and continues to be for the Internet and the World Wide Web. In that respect the Free Software movement has been actively creating the environment in which the movement thrives and because it thrives it continually recreates and strengthens the environment in which it exists. The output of the Free Software movement becomes an input to the system – cyberspace - upon which it is dependent. This is the *technical* aspect of Free Software's recursive relation to its environment.

Free Software also has a recursive relation that is *socio-political*. As a public the movement is recursive because it creates the foundations for its own success, similar to how it continues to create the technical foundations in which it thrives. Kelty writes:

> "A recursive public is a public that is vitally concerned with the material and practical maintenance and modification of the technical, legal, practical, and conceptual means of its own existence as a public; it is a collective independent of other forms of constituted power and is capable of speaking to existing forms of power through the production of actually existing alternatives" (2008: 3).

> "Recursive publics are publics concerned with the ability to build, control, modify, and maintain the infrastructure that allows them to come into being in the first place and which, in turn, constitutes their everyday practical commitments and the identities of the participants as creative and autonomous individuals" (ibid: 7).

Two of the most important legal challenges for the Free Software movement, with respect to the "institutional ecology" within which that movement exists and has to survive, are copyright and patent law (Benkler 2006; Frischmann 2007). I will, for brevity's sake, leave aside the question of patent law, although the Free Software movement in a variety of ways also contest existing patent laws[66], and only consider copyright law.

It is copyright law that has permitted the articulation of the GPL, to which we turn below, and copyright law reform is one of the main foci of the political lobby work of the Free Software movement. They work to "modify" copyright law, but they also seek to "maintain" it, because copyright law constitutes the legal foundation upon which they rest as a movement. That Free Software is based upon copyright law, yet seeks to reform copyright law in accordance with the subversive way in which the Free Software license, the GPL, is anchored in copyright law, is an important aspect that is often misunderstood.

A recent political initiative can help illustrate how the Free Software movement approaches the issue of copyright reform. The Swedish Pirate Party, which is a political platform "to legalise [non-commercial] internet file-sharing" and other cyberspace customs and which gained 7.1% of the Swedish votes and thus "won one of Sweden's 18 seats in the European parliament" (Schofield 2009), has proposed a copyright reform that would harm the cause of Free Software. It is somewhat ironic that a party, which it would not be possible to imagine the

66 The Free Software Foundation works to exclude the realm of computer software from patent law entirely. See for instance "Patent Reform Is Not Enough" available online at http://www.gnu.org/philosophy/patent-reform-is-not-enough.html and the "End Software Patents" campaign at http://endsoftpatents.org/.

emergence of without the prior existence of the Free Software movement, should propose reforms that would severely harm – quite possibly entirely undermine – the work of the Free Software movement. The harm consists of a radically shorter copyright term, namely five years, after which a copyright covered work would enter the public domain. As we shall see, the GPL rests upon copyright law to protect against enclosure and therefore that protection would be rendered useless after five years, because source code in the public domain can be enclosed in future software that is not Free Software. Because non-free software does not reveal its source code – only the binary programme is copyrighted – the source code of non-free software would never enter the public domain anyway. What could legally be shared non-commercially within copyright law reformed according to the Pirate Party's proposal would only be the binary programmes. However, this completely overlooks the nature of non-free software, which is not only protected by copyright, but also by EULAs (End User License Agreements). The use of EULAs would most likely exempt non-free software altogether from any reforms to copyright law in any case. Furthermore, as Stallman writes, non-free software could include a time bomb that simple renders it unusable after five years, meaning that nothing useful would enter the public domain:

> “Thus, the Pirate Party's proposal would give proprietary software developers the use of GPL-covered source code after 5 years, but it would not give free software developers the use of proprietary source code, not after 5 years or even 50 years. The Free World would get the bad, but not the good. The difference between source code and object code and the practice of using EULAs would give proprietary

> software an effective exception from the general rule of 5-year copyright — one that free software does not share" (Stallman 2009).

It is for these reasons that the Free Software movement is vitally concerned with the "practical maintenance and modification" of copyright. Without copyright there can be no Free Software as we know it. It is beyond the scope of this essay to investigate further the details of copyright reform from the perspective of the Free Software movement. However, this example shows that the nature of Free Software is such that conventional approaches to copyright law reform, such as reducing the term (before a protected work enters the public domain, from which it can be enclosed through inclusion into non-free future works), simply no longer makes sense in the context of Free Software.

The Free Software movement's work to reform copyright and the creation of Free Software as such are better understood as a contribution to the democratisation of technology to which a reform of copyright law is integral and necessary, but by no means sufficient. Within the philosophy of technology Andrew Feenberg has written on the democratisation of technology. He states the need for this in a manner very relevant for the case of Free Software:

> "Technology is power in modern societies, a greater power in many domains then the political system itself. The masters of technical systems, corporate and military leaders, physicians and engineers, have far more control over the patterns of urban growth, the design of dwellings and transportation systems, the selection of innovations, our experience as

> employees, patients, and consumers, than all the electoral institutions of our society put together" (1999: 131).

With such a powerful position in the everyday lives of people and the way in which software is integral to most technology, either in development, application or general use, we may understand the work of the Free Software movement, conforming a recursive public, as a contribution to the democratisation of technology. Feenberg takes note of how technology is both a tool for domination and a tool for liberation, and that its value is determined both by the prevailing mindset in which it is implemented, what Feenberg calls its "code", and the ways in which technologies are put to use.

> "[T]he computer is neither good nor evil, but both. By this I mean not merely that computers can be used for either domination or democratization but that they can evolve into very different technologies under the influence of different strategies of development" (Feenberg 2002: 91).

The ambiguity or ambivalence of technology Feenberg presents like this:

> "1. Conservation of hierarchy: social hierarchy can generally be preserved and reproduced as new technology is introduced. This principle explains the extraordinary continuity of power in advanced capitalist societies over the last several generations. This continuity was made possible by technocratic strategies of modernization, despite enormous technical changes.

> 2. Subversive rationalization: new technology can also be used to undermine the existing social hierarchy or to force it to meet needs it has ignored. This principle explains the technical initiatives that sometimes accompany the strategies of structural reform pursued by union, environmental, and other social movements" (Feenberg 1998).

The work of the Free Software movement, we may therefore say, is an example of "subversive rationalization" both with regard to the technical dimension and with regard to socio-political dimensions. The Free Software movement exhibits a recursive relation with regard to not only the technical foundations – cyberspace and software - but also with regard to the institutional ecology. In the context of the legal aspect of the institutional ecology, Free Software, as we shall see in Section 3.5 below, is dependent on copyright law, while at once working actively in political lobbies to reform that very copyright law (as well as lobbying to exempt software from patent law). The recursive Free Software public, then, instantiates a process of "subversive rationalization" of software technology and thus contributes to a democratisation of technology led by civil society.

Kelty has conceptualised the recursive phenomena of Free Software and cultural derivatives in the wider Free Culture movement in such a way that other social movements can learn from the example. If some hackers with long beards can subvert copyright law and transform the powerful software industry and thereby set a precedent for a significant transformation of

societal relations, perhaps other movements can do so, too[67]. Certainly for social scientists the concept of recursive publics can be applied to other domains. Imagine, say, a definition of organic food articulated by the permaculture movement[68], a driving test articulated by the Bicycology movement[69], or, indeed, property relations articulated by anti-capitalist movements. The example of the Free Software movement – for the rest of global civil society - stands as empirical evidence that it is possible to organise your own social relations and articulate your own property relations, that is to autonomously establish a community through voluntary associations through a subversion of the decision making authority that defines copyright as an instance of private property.

---

67 There are many movements that are successfully contesting the value measures of capital and changing their social relations with regard to things in their struggles against market mechanisms, see for example “We Are Everywhere” by the Notes from Nowhere Collective (2003). However, the Free Software movement remains the only movement that has articulated its values and social relations into legal language in such a manner that it has been accepted in courts of law and thus is directly subversive of the existing letter of the law .

68 The Permaculture Association writes: “The word 'permaculture' comes from 'permanent agriculture' and 'permanent culture' - it is about living lightly on the planet, and making sure that we can sustain human activities for many generations to come, in harmony with nature. Permanence is not about everything staying the same. Its about stability, about deepening soils and cleaner water, thriving communities in self-reliant regions, biodiverse agriculture and social justice, peace and abundance”. Available at http://www.permaculture.org.uk/knowledge-base/basics

69 “Bicycology is a cyclists' collective that offers a range of activities to promote cycling and make links with wider issues of environmental and social responsibility. We use our passion for cycling to pursue our vision of a just and sustainable world through a combination of education, entertainment and creative direct action”. Available at http://www.bicycology.org.uk/

These organisational lessons provided by the example of Free Software have been the subject of a paper by cyberspace visionary Douglas Rushkoff, originally written for the London think tank Demos:

> "The emergence of the internet as a self-organising community, its subsequent co-option by business interests, the resulting collapse of the dot.com pyramid and the more recent self-conscious revival of interactive media's most participatory forums, serve as a case study in the politics of renaissance. The battle for control over new and little understood communication technologies has rendered transparent many of the agendas implicit in our political and cultural narratives. Meanwhile, the technologies themselves empower individuals to take part in the creation of new narratives. Thus, in an era when crass perversions of populism, and exaggerated calls for national security, threaten the very premises of representational democracy and free discourse, interactive technologies offer us a ray of hope for a renewed spirit of genuine civic engagement" (2004: 16).

These are great promises. However, as we covered in Chapter 1, the philosophical problems inherent in "information exceptionalism" and their consequences for Free Software and Free Culture politics result in a very important recursive relation being absent, namely with the tangible realm. The Free Software movement is "vitally concerned" with copyright reform and abolition of software patents, but *they are not* vitally concerned with substantial reforms of property relations in the tangible realm, on the contrary. The material foundations of cyberspace – and thus the realm in which software development takes place –

is certainly part of the infrastructure that allows Free Software to come into being in the first place. Without a critical approach to ownership in the tangible realm the Free Software movement will remain vulnerable to enclosure led by those capital interests.

The most important commons is the commons of the land and the tangible means of production and distribution. That is the shared material reality of humanity from which all other possibilities arise, whether tangible or intangible. The information commons is a luxury, the icing on the cake. It is costly and it is precious and has excelled in perpetuating the seemingly ubiquitous propensity of human beings to engage in sharing and cooperation when constraints are lifted. The liquid architecture of cyberspace has facilitated these emergent processes very well. But the proliferation of sharing and cooperating, which attracts so much attention - from rent seekers and anti-capitalists alike – is not confined to cyberspace, *nor* to the intangible realm.

The difference between tangible and intangible is not what determines whether people share and cooperate. As we have seen there is a long, rich history of commoning. Commoning is a shared skill of humanity and not a skill that suddenly, morphogenetically appeared on a global scale when the doors to cyberspace were opened. Rather, cyberspace provided people with a space that was not yet enclosed. There were few fences in cyberspace, so sharing and cooperating was possible. It was possible because the constraints of private property – present in almost all other dimensions of life – were absent. Now they are invading cyberspace, seeking rent and expansion of capital interest. It is laudable to form a movement to strike back and protect cyberspace, but a more reflexive approach would not stop at the gates of the tangible realm. The threats of capital will not go away as long as capital exists in its particular form. It will return, it will continue to seek new ways of enclosure, which

suggests that it is necessary to address this problem of capital at the most fundamental level, namely with regards to ownership.

Addressing merely the symptoms of avarice and capital expansion in the intangible realm condemns Free Culture to an eternal and defensive battle and separates Free Software and Free Culture from the global movement of movements struggling to take back the land and the means of production. Without acknowledging and acting upon its recursive relationship to the tangible realm, Free Software remains a virtual commons that is detached from the struggles for real commons. Having witnessed the phenomenal emergence of commoning in cyberspace – when the constraints of private property were lifted – we can only imagine what transformations the tangible realm would undergo if constraints were lifted there. As I said above, the opposition here is not *tangible versus intangible*, but private property versus forms of property that facilitate collective creativity and self-organisation.

Nevertheless, the achievements of the Free Software movement are remarkable. It is in the GPL that these achievements are manifest and in the following section this software license and copyright reforming declaration of hacker values will be explained in detail.

3.5 The GNU General Public License: copyright subversion and constitution.

Contemporary literature addressing copyright law in the context of software is replete with gaps, misunderstandings and misleading statements with regard to Free Software and the GPL.

It will be instructive to briefly present a few of those misunderstandings here.

3.5.1 Misunderstanding the GPL.

A frequent misunderstanding of Free Software is that it is placed in the public domain. We can find this replicated in the third edition of an Oxford University Press textbook on Intellectual Property Law:

> "[The Free Software movement] is dedicated to the idea that code should be made publicly available rather than protected by copyright law. For example the Free Software Movement develops code and places it in the public domain. It can be used by anyone, with the proviso that they agree to the terms of the General Public License, which dictates that any improvement made to the software will be similarly placed in the public domain" (Davis 2008: 75-76).

As we shall see in more detail later in this chapter, this is not only misleading but false. The only correct statement in the quote is that "[i]t can be used by anyone, *with the proviso that they agree to the terms* of the General Public License". Firstly, Free Software is protected by copyright law, that is its very foundation. Hence, secondly, Free Software is not at all placed in the public domain. This is the genius of Free Software. Instead it is protected from enclosure through a subversion of copyright and that subversion is articulated in the GNU General Public License (the GPL). The GPL is best understood as a set of sub-clauses to copyright, hence it rests upon copyright law.

Turning to Pearson Longman's "Intellectual Property", Seventh Edition, we find a long, densely case referenced chapter on copyright (Bainbridge 2009: 239-296), yet not one mention of Free Software. The chapter begins:

> "Copyright law has a history of development that can partly be explained by reference to technological change … The Copyright, Designs and Patents Act 1988 was an attempt to keep abreast of developments in technology coupled with an intention to enact legislation that would take future change in stride. Of particular concern was the protection of computer programs and of other works stored or transmitted in digital form" (ibid: 239).

If we look to another set of leading voices in the field, Bently & Sherman's Intellectual Property Law textbook, we find no mention of the phenomenon of the GPL in the second edition (2004) at all, but in the current edition (2008) space has been made for a mentioning. On page 266 a section is devoted to the work of the Free Software Foundation, adding little to the debate. It has to be noted that one of the greatest technological changes in this context in contemporary times, namely the advent of the Internet, which is built in great part with Free Software and recursively has made the further success of the Free Software movement possible, is hardly taken into account by the legal, academic establishment.

In the following section, I present the GPL and its legal, and above all property implications in more detail.

3.5.2 The GPL: just a software license?

The GNU General Public License ("the GPL") is a software license, which, as is also the case of non-free software licenses, determines the conditions of distribution of a piece of software. The GPL was first published in 1989. The GPLv2 was published in 1991 and the process towards GPLv3 began officially with a global gathering at MIT in January 2006, which has been recorded, documented and discussed extensively, as has the gatherings that followed: the Second International Conference on GPLv3, which was combined with the 7º Fórum Internacional Software Livre, took place April 19-22 in Porto Alegre, RS, Brazil; the third happened in Barcelona, June 22-23; the fourth took place in Bangalore, India, August 23-2; and the fifth took place in Akihabara Tokyo, Japan, November 21-22, 2006. Each of the conferences were organised by the local Free Software groups and coordinated with the civil society of developers and users. The process was coordinated by four committees, each composed of "18 to 22 members who were chosen from vendor, developer, hacker and open source communities" with a privilege of the original author, Richard Stallman, who "would make the final decisions on hot-button issues like digital rights management (DRM). However, even with Stallman as the ultimate decider in what stays and goes from the license he created in 1989, committee members were optimistic that the right issues are being addressed" (Loftus 2006).

The GPLv3 was finally published in June 2007, with a preamble and 18 sections of legalese in more than 5000 words; it is deliberately written for and within global civil society, rather than for any specific national jurisdiction (an aspect to which I return briefly below) and the GPLv3 is now the recommended software license by the Free Software Foundation. But how - exactly - does it work?

Software, like a book, a painting or a poem, is by default copyrighted and the exclusive right to define distribution terms belongs to the creator (unless s/he, like many academics, have signed away their so-called "intellectual property" as part of signing their employment contract). A software license is an expression of the creator's specific conditions with respect to distribution of the copyrighted software.

Copyright specifies the control powers and use privileges, conferring on the author - and the author only - an exclusive set of rights to: (i) reproduce or copy the copyrighted work; (ii) prepare derivative works (modify the work); (iii) distribute copies of the copyrighted work to the public by sale or other transfer of ownership, rental, lease or lending; (iv) perform or display the copyrighted work publicly. It is this articulation of copyright that the Free Software movement aims to radically reform and alter. As we shall see they have managed to do so with quite some success.

The Free Software movement's creations, that is the software they write and release, rest upon the provisions of copyright law, because the GPL specifies what the copyright holder permits others to do with a Free Software programme. The GPL is legally speaking a set of sub-clauses to copyright. These sub-clauses are articulated in such a way that they – at once – build on copyright *and also* subvert the function of copyright. The Free Software Foundation calls these sub-clauses "distribution terms" and they specify certain freedoms that are provided to users, but also specify certain conditions that the users are required to observe and follow in order to enjoy the privileges of freedom. In writing the GPL the Free Software community has constituted itself as the relating-subject (A+C), classified (free) software as its

related-to object (B) and specified their *relational modalities* and thus established a (software) commons.

3.5.3 Copyleft freedoms: reciprocity in perpetuity.

The general concept that is at play in the GPL's articulation of sub-clauses to copyright, or distribution terms in extension of copyright, has been labelled Copyleft. The articulation of the GPL has spawned a variety of other Copyleft licenses, notably those of the Creative Commons70, and as such the GPL is a particular instance of Copyleft, which defines and articulates the "four freedoms" of Free Software:

> "To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code or any program derived from it but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable" (FSF 2001).

The four freedoms of Free Software are thus:

- The freedom to run the program, for any purpose (freedom 0)
- The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.

---

70 The Creative Commons was explained briefly in Chapter 1.

- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this (FSF 2009)

The code and the freedoms become inseparable through the ingenious element of *reciprocity in perpetuity* that is inherent in the GPL. Its opponents call this relational modality a "viral clause" in order to provoke associations with computer vira and illness in general[71]. For the software privatisers, GPL'ed code is a contamination, because it brings with it – as the code and the freedoms are inseparable – the freedom to share and cooperate *and* protects this freedom against enclosure.

The *relational modality* that instantiates *reciprocity in perpetuity* is a clever articulation of sub-clauses to copyright that on the one hand binds the code and the freedoms, while on the other, as a consequence of this binding, ensures reciprocity between developers and users within the community. In logical terms it is stipulated in the GPL that if a GPL'ed code segment X is included in programme Y, then Y, if it is released to the public, must also be released under the GPL. In that way you are obliged to extend and forward to others the four freedoms awarded to

71 Not unlike the subversion of the "framing effect" with regard to property that I have presented in this essay as a response to Stallman's warning that "most people" are unable to understand property beyond an absolute, natural rights-based conception, David Bollier has given a positive meaning to the term "viral" in his "Viral Spiral: How the Commoners Built a Digital Republic of Their Own" (2008). This attempt reflects my own view: rather more information, than less, rather investigate, than obscure.

you by the copyright holder through the distribution terms defined in the GPL, in case you elaborate on a given segment of Free Software and redistribute it. If you just modify and keep your modified software to yourself you are not obliged to do anything and can simply enjoy the four freedoms in private. In the GPL Version 3 the relational modality that ensures reciprocity in perpetuity is articulated as follows[72]:

> "The GPL - Section 5: Conveying Modified Source Versions.
>
> You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:
>
> - a) The work must carry prominent notices stating that you modified it, and giving a relevant date. [*In order that fellow commoners know that code has been changed and when.*]
>
> - b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". [*The conditions or additional terms referred to here are irrelevant for our analysis.*]

72 The entire text of the GPL is available online @ http://www.gnu.org/licenses/gpl.html.

- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. [*This is the reciprocal specification: "the entire work" is the original code, plus your contribution, which then enters the Free Software commons. A can never be separated from C and the relational modality (reciprocity in perpetuity) attaches to, or follows B as it circulates. i.e. the commons grows.*]

- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so. [*This is irrelevant for our analysis.*]

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the

> compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate". [*This clarifies that a compiled – i.e. binary - Free Software programme (or application) can be used with other programmes without subjecting these other programmes to the conditions of the GPL, thus defining the limit of the reciprocal element. The exact details are not strictly relevant for this analysis, but concerns the freedom to combine Free Software in binary form with programmes that are not Free Software. GNU/Linux distributions, such as Ubuntu, do just that.]*

Reciprocity in perpetuity should be clearly distinguished from the reciprocal give and take that characterises a market economy, in which individuals enter into contractual relations that are characterised by *direct reciprocity*. Reciprocity in perpetuity is likely to be a feature of most commons: the commons is always there, for you to access and use and take from; however, it demands care and attention in turn. A commons can be destroyed by enclosure, but also by neglect or over-use. In the moment that a commoner does not perform the duty of care that has been distributed to her, the reciprocal link is broken: it might exclude her from the commons or contribute to its collapse. This is most obvious if we think of commons of the land and the ecological balance that sustains them. The GPL ensures that everyone is able to access the Free Software commons, and also that everyone will act in ways that ensure its continuity (and in fact, growth) into the future. Reciprocity in perpetuity refers to an attitude of responsibility and responsiveness that is necessary in

order for the commons to remain perpetually *there* (see also Section 2.1.3 on the distribution of care).

3.5.4 Copyleft loves copyright.

The GPL, anchored firmly in copyright law[73], yet subverting copyright, ensures *me* that if *you* use a bit of my code and add to it, then the bit that you added will be available to me on the same conditions. In that way our common creations are bound to and by the same freedoms in perpetuity. Free Software hackers are (neo-)commoners:

> "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 … It doesn't mean abandoning the copyright; in fact, doing so would make copyleft impossible. The word "left" in "copyleft" is not a reference to the verb "to leave" — only to the direction which is the inverse of "right"" (FSF 2009).

---

73 Not only is copyleft dependent on copyright protection, but the GPL, that is *its specific wording*, is protected by copyright. The GPL itself is therefore not copylefted, but remains under conventional copyright. In this way the GPL *also* interfaces with and makes use of existing copyright law. Stallman explains why: "We don't want people to circulate modified texts that purport misleadingly to be the GNU General Public License. Copyright does not restrict the writing of license text. Thus, if you want to write a license with wording similar to the GNU GPL but not exactly the same, you can do so. But you can't copy our preamble without our permission, so you can't make it appear to have come from us" (Stallman in Biancuzzi 2009).

Because the GPL is “merely” a set of sub-clauses in extension of existing copyright law, which is awarded automatically upon a creation's release to the public, in the moment that you do not adhere to the terms and conditions under which the GPL puts you, the GPL is rendered invalid. It follows that you can no longer claim the four freedoms of Free Software, since they are only yours to enjoy as long as you reciprocate them. Therefore, when breaching the GPL the software in question is no longer covered by the GPL's additional distributions terms, but reverts to being covered under conventional copyright law. That, of course, means that you are not allowed *at all* to copy or redistribute the code in question. Breaching the GPL by enclosing code is thus a *de facto* breach of copyright. I look at court cases setting legal precedents for such breaching in Section 3.6.

In other words, the GPL is a “hack of genius” (Meretz 2004: 31) that utilises existing law from within the system otherwise threatening Free Software development, namely copyright law, and subverts it through a reconfiguration that ensures reciprocity in a community instead of exclusion on behalf of an individual (see also Oksanen and Välimäki 2006). Copyleft, then, is not only a word play, but a whole new way of imagining copyright. It is on this basis that the Free Software movement is working to reform copyright law. They do not by any means want to eliminate copyright law, since without copyright the GPL loses its trespassory protection and hence means of defence. This has already been tested in a court of law (see Section 3.6 below).

That copyleft is dependent on copyright is often misunderstood, not only in influential textbooks on copyright law as we saw above, but also among anti-capitalists. The attentive reader will by now be aware that this reliance of a commons on the

institution of private property is by no means contradictory. On the contrary, in capitalist democracy, it is in fact inevitable.

The communitarian form of property that Harris describes, and which we adopted as a model of an autonomous commons within capitalism, represents the Free Software commons well. Its trespassory protection, given by copyright yet expressed as copyleft, circumscribes a realm of *collective-freedom-to* share and cooperate. This relational modality is articulated in the form of the GPL (a property protocol), which provides use privileges, and indeed a certain amount of control power to anyone whose actions do not undermine the conditions of reciprocity stipulated within it. The control power of the copyright holder is used to surrender the exclusivity of that control power, making it available to everyone who agrees to surrender theirs in turn under the same conditions. Use privileges are opened up to anyone in that way. The capitalist characteristic of property, the exclusive right to wealth effects is, as a side-effect of the surrender of control power, made non-exclusive: everyone can potentially sell products and services based on GPL'ed software code, as long as the code continues to circulate freely.

Understood in this way, the configuration of property relations in the Free Software commons can be illustrated in this manner (see next page):

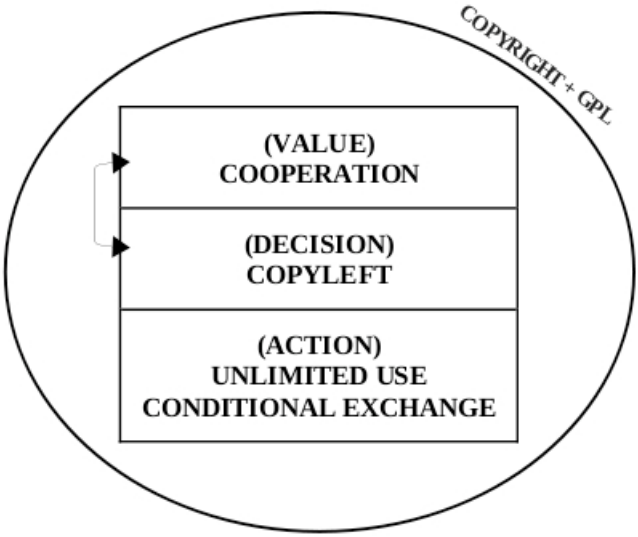

Illustration 10: The GPL as property configuration.

Both the original decision to (conditionally) surrender control power through copylefting one's creation, as well as any other decision made with regard to software code released under the GPL are legitimised by reference to common values of the hacker community, such as the fostering of sharing and cooperation. The GPL is an articulated protocol of such common values, and affords the author and everyone else use and exchange privileges.

Copyleft uses copyright as its enforcement mechanism in a world dominated by private property relations and authorised self-seekingness on behalf of corporations – that is, authorised profiteering in the interest of shareholders. In a world of

continuous enclosure, that is increasing individual and quasi-individual control powers over land (and everything else), subversion of enclosure might be the only way to stop its progress short of reverting to increased state regulation. To subvert enclosure is to subvert individual and quasi-individual control powers, by using the authority so invested to surrender some control power (conditionally) and open up use privileges to others. This is what copyleft does. It is also, in essence, what social centres and hacklabs do: some social centres are squatted, others are rented, and again others privately owned. In all three versions some degree of control is conferred respectively upon (i) the quasi-individual collective of squatters, (ii) tenants or (iii) landlords. In the squat, control power is *de facto* rather than *de iure* based on the physical possession and occupation of the building or plot of land. The rented social centre means that the use-privileges and some control power has been contracted out from the owners to the tenants. In the case of a social centre being privately owned by the social centre collective (often in form of a cooperative), control power lies even more straightforwardly with the centre. In all cases, however, this control is used to open up use-privileges to the wider community, as well as surrendering some decision-making power over how the space is used and by whom (though usually not the power to alienate the title on the market, i.e. the power to sell the centre).

Within capitalist democracy, most commons will have to rely on some sort of enforcement mechanism that can protect the commons from enclosure. Private property rights come with such state sanctioned powers of enforcements attached and, in principle, instances other than copyright can be “hacked” in a similar way.

The relation between the GPL and copyright law is one of dependence. But this dependence has less to do with the

fundamental need for private property in social organisation, or with the logical priority of private property. Rather, it has to do with the relentless nature of capitalist privatisation which creates the need for strong trespassory protection of a commons in the first place.

If hackers bought a piece of land and fostered a forest garden, they could constitute themselves by articulating their decided upon relational modalities with regard to their forest garden commons. As discussed in Chapter 2, coming together to buy a piece of land in legal terms is simply an instance of group private property – like a corporation – but what constitutes a commons is not only a matter of its precise legal foundations. A commons is an idea and it is an experimental process of commoning: working together, sharing and cooperating. As an act of creation the commons is on a trajectory away from the state and its modalities – by which door it exits is not necessarily a crucial matter. It is a collective expression and fulfilment of needs and desires. A commons self-articulates in and through commoning and its emergent property relations and protocols. One way it can defend itself is through the co-option of capitalist trespassory protection for its own ends.

Structurally speaking – with regard to social organisation – the "only" difference between private property and the configuration of property inherent in the GPL is the shifted focus from individual exclusion and self-seekingness to a sharing and cooperating community. Both are relations between people with regard to things, structured by normative protocols.

If we recall the process described in the Introduction *from* the Magna Carta and the Charter of Forests *to* the American Declaration of Independence, which was a process from *rights articulated for collective and communal benefit* to *rights*

articulated for individual privilege, we see here the exact reverse: copyright is articulated for the privilege of individuals to exclude others, whereas the GPL subverts that individual privilege and transforms it into an articulation that ensures collective benefits in a community of reciprocity. Private property - in the sense of it conferring decision rights, sanctioned by the state - can therefore be really useful for *commonism*. The Free Software commons is a function of private property. Standing on that foundation, it is a rather safe commons. However, it is not necessarily on the legal basis of private property that the Free Software commons is *constituted*. It is constituted as a commons by the voluntary association of hackers. They act according to their common constitutional liberties, as it were.

3.5.5 Constituting a commons.

In addition to being a clever legal document, moreover, the GPL is also a *constitution* of the Free Software movement (or community). It defines the boundaries of the software commons and binds together the commoners in the practices of commoning. It communicates a global vision for the community of software freedom, and articulates its relational modality. Furthermore, the GPL is an expression of the idea that freedom as *collective-freedom-to* needs to be written into the normative protocols that guide behaviour in capitalist democracy, and indeed, that it *can be* written into protocols. Inscribing *collective-freedom-to* in that manner requires certain conditions to be observed by all, in order for this freedom to remain collective into the future. But as such, these conditions are voluntary and reciprocal: you only have to abide by the rules if you want to use the resources of the commons, and you can

expect reciprocity in doing so. The commons is protected both through the practices of commoning and reciprocity in perpetuity, but of course also by the trespassory rules that copyright enacts. However, with Free Software, trespassory protection does not *exclude* people. Rather, it asks them to act in a particular kind of way. The Free Software commons is "open" to people not according to their *identities* (in the birth certificate kind of sense) but according to their *actions*.

Wendy Pullan (2004) in her architectural studies of the Israeli wall built to contain the Palestinian people makes an analytical distinction between thick and thin walls. Thick walls "structure differences and transitions, thereby embodying and fostering a certain richness of meaning". Thick walls are constitutional of identity, yet permeable. Pullan uses the example of the Roman *poemerium*, the symbolic furrow later echoed in the city walls, "which deviated as necessary and were added to and changed over time to represent the practical structures of daily life" (ibid.) to communicate what a thick wall is. A thick wall is a facilitator, a mediator and point of reference, whereas thin walls, such as the Israeli one, are "constructed expressly to separate and divide".

Pullan's perspective is helpful to understand the GPL in metaphorical terms. We can understand the GPL as a thick wall around the Free Software community, protecting it, but not excluding the rest of the world unconditionally: the wall that the GPL instantiates is best understood as an invitation to join an intentional and autonomous community, whose goal is "to give people liberty, and to encourage cooperation, to permit people to cooperate" in the understanding that one should "never force anyone to cooperate with any other person, but make sure that everybody's allowed to cooperate, everyone has the freedom to do so, if he or she wishes" (Stallman 2001b).

The GPL is based on distribution rather than exclusion (Weber 2004) in that it de-emphasises the regulation of an individual owner/creator who can exclude others - and for how long - from access to and use of software code. Rather the GPL instead emphasises how, and under which conditions software code can be shared and distributed in a common fashion. In doing so, the GPL unites people: it builds communities. The Free Software movement – "vitally concerned with what allows them to come into being in the first place" – has in many senses set new standards for autonomous constitution. This again underpins the notion of the Free Software community as a recursive public: it thrives in global civil society and strengthens global civil society by showing by example how global voluntary associations can organise and protect themselves.

Because it is a global network of communities composed of members residing in respective jurisdictions, each subject to different specificities of local copyright law, the GPL is also an experiment in global(ised) law making beyond the nation state through voluntary associations[74]. A property law made within global civil society by a social movement. The global dimension is reflected in the recently completed process to update the GPL

74 In an aside it should be noted that lex mercatoria exhibits similar traits. Legal sociologist Guenther Teubner argues that "Lex mercatoria, the transnational law of economic transactions, is the most successful example of global law without a state ... [but] it is not only the economy, but various sectors of world society that are developing a global law of their own. And they do so ... in relative insulation from the state, official international politics and international public law ... Technical standardization and professional self-regulation have tended towards worldwide coordination with minimal intervention of official international politics. The discourse on Human Rights has become globalized and is pressing for its own law, not only from a source other than the states but against the states themselves. Especially in the case of human rights it would be "unbearable if the law were left to the arbitrariness of regional politics" (Teubner 1997: 3-4).

to Version 3, which includes efforts of “denationalization”, in order to position the GPL within global civil society, in an “attempt to cut the language of the license loose from any particular system's copyright law” (Moglen 2006), so as not to confine it to any specific nation state's legal system and its terminology.

Free Software is created for both individual use and the common good. It contributes to society by creating commonalty: the Free Software community is a voluntary association of individuals whose creative agency make up a software commons. The GPL facilitates a codification of unwritten rules, norms, and customs derived from, on the one hand, the social and political concern that free access to source code be crucial for society, and on the other, the practical realisation that good software is produced by sharing and experimenting with each other’s code freely and openly as a community. Realising that the most central element of software is the need to share, circulate and distribute it, for the sake of software evolution itself and for the sake of the common good of the people, the GPL articulates freedoms that focus on sharing and cooperating and secures the continued possibility to do so.

For many years the GPL remained untested in court and as such the legal validity of the self-organised and autonomously declared software freedoms remained unknown. The Free Software movement never wished to test it, but kept to a private policing and enforcement of the GPL when breaches became known (see below). When the time came for the GPL to enter a court of law the movement was a global community with well-established and widely recognised customs, and many awaited the first decisions with great anticipation.

3.6 Defending the GPL: a recursive public self-organises.

The way in which the Free Software movement has responded to violations of the GPL is a testimony to its self-organisational capacity. It provides an example of what Rushkoff finds so promising in "Open Source" as a model for democracy, because the Free Software movement's engagement with the law and its self-legislative capacity:

> "...marks a profound shift in our relationship to law and governance. We move from simply following the law, to understanding the law, to actually feeling capable of writing the law: adhering to the map, to understanding the map, to drawing our own. At the very least, we are aware that the choices made on our behalf have the ability to shape our future reality and that these choices are not ordained but implemented by people just like us" (2004: 58).

Not long after the GPL was first used in 1989, enforcement activities commenced as informal community efforts often in public Usenet discussions. The next ten years the Free Software Foundation was the only established organisation defending the GPL and "their enforcement was generally a private process; the FSF contacted violators confidentially and helped them to comply with the license". It was not until the early 2000's that things changed. "By that time, Linux-based systems had become very common, particularly in embedded devices such as wireless routers" - in a realm where non-free software is generally prohibitively expensive to implement and customise - and a new dimension of enforcement began: "public ridicule of violators in the press and on Internet fora supplemented ongoing private enforcement and increased pressure on businesses to comply". The GPL Compliance Lab was established by the FSF in 2003,

as more and more cases became known, with a view to building "community coalitions to encourage copyright holders to together settle amicably with violators" (SFLC 2008).

In 2004, a German Free Software programmer called Harald Welte commenced a more organised enforcement approach with a project called GPL-violations.org. In late 2003 he had discovered that "a bunch of companies" were using code from a GPL'ed project - on which he was working - in a manner that breached the GPL. He became active in the legal realm and in the same way as it is said that Free Software often begins with an itch, a need to solve a personal, specific right here and right now problem, Welte set up the web site GPL-violations.org with an accompanying mailing list for sharing reports, analyses and advice on alleged, potential and definitive breaches of the GPL. They have been busy ever since - in the "About" section on their web site it reads:

> "By June 2006, the project has hit the magic "100 cases finished" mark, at an exciting equal "100% legal success" mark. Every GPL infringement that we started to enforce was resolved in a legal success, either in-court or out of court" (GPL-Violations.org 2009).

The GPL-violations.org project has expanded accordingly, there are several busy mailing lists, in addition to the site, where people consult each other – that is, discuss as software commoners if a particular act is a violation or not. To frame it in terms of property, the relating-subject (A+C) is developing its own enforcement mechanisms and through discussions about enforcement they refine their own understanding of the relational modalities of their community and reflect upon what is permitted in the commons and what is not. For instance, discussing the

grey areas of the GPL in a fast developing field of embedded devices is a perennial task.

When the GPL was finally fully tested in a court of law, in September 2006 in Frankfurt am Main, the judgement read that because a device incorporating GPL'ed code was brought to market without proper GPL compliance, the:

> "Defendant is ordered to pay to Plaintiff 2,871.44 EUR, plus interest on this amount of 5 percentage points above the base interest rate since February 25, 2006; regarding the amount of 141.34 EUR, payment shall be made in exchange for the transfer of ownership of the data storage unit "[…] Wireless G Network Media Storage DSM-G6000" which is owned by Plaintiff."(GPL-Violations.org 2006).

Another crucial element of the ruling in the German court asserts that the GPL is a valid software license, a proper instance of copyright, and that in effect it is a contractual relation, accepted by the defendant and therefore the plaintiff's demands are ruled in favour of:

> "The GPL applies to the legal relationship between the authors and Defendant. The three software programs are undisputedly licensed only under the terms of the GPL. In the case of free software it is to be assumed that the copyright holder by putting the program under the GPL makes an offer to a determinable or definite circle of people and that this offer is accepted by users [of the software] through an act that requires consent under copyright law; in this respect, it can be assumed that the copyright holder enters into this legal relationship

> without receiving an actual declaration of acceptance [from the users] (Section 151 of the German Civil Code (BGB)).” (ibid.)

The ruling went against the argument of the defence which was loosely based on and attempted to mobilise anti-trust laws. The German court is clear and the logical aspect of the verdict reflects the analysis of the preceding section, which stated that if breaching the GPL, the code in question reverts to be protected under conventional copyright:

> “It need not be decided whether, as Defendant argues, the provisions of the GPL violate Article 81 EC and Section 1 of the German Antitrust Act (GWB), in particular the prohibition against price fixing and of predetermining the conditions of secondary contracts in the first contract. This would, according to Section 139 of the German Civil Code (BGB), result in the invalidity of the entire license agreement with the consequence that Defendant would not have a right of use in the software at all, so that Plaintiff could file a copyright infringement claim for that reason.” (ibid.)

If you invalidate the GPL you are left with the foundation upon which it rests: copyright, and copyright is per default an exclusive right of the creator; thus contesting the validity of the GPL is practically useless, since an invalidation of the GPL at any rate will render the copyleft holder an exclusive copyright owner. Those who do not comply are left with but one choice, apart from paying up and withdrawing the device, and that is to play along. The rulings of this kind have had profound effects and there is now a proliferation of what is called “third-party

firmware" projects for wireless network devices, adding features and capabilities beyond what was originally intended by the manufacturers[75]. The history of the GPL in court and the mechanisms of enforcement is so far a successful one.

During 2006 a range of even firmer defence mechanisms emerged. The Free Software Foundation Europe set down a Freedom Task Force, which provides licensing services to individuals, projects and businesses which use Free Software, working with GPL-violations.org and complementary to the Software Freedom Law Center (SFLC), which provides "legal representation and other law-related services to protect Free Open Source Software" (SFLC n.d.).

A significant conflict that was recently concluded began when the *Association pour la formation professionnelle des adultes* (AFPA), a French educational organisation, ordered and purchased some software, which turned out to be in breach of the GPL:

> "The events of the case go back to early 2000, when Edu4 was hired to provide new computer equipment in AFPA's classrooms. Shortly thereafter, AFPA discovered that VNC was distributed with this equipment. Despite repeated requests, with mediation from the Free Software Foundation France, Edu4 refused to provide AFPA with the source code to this version of VNC. Furthermore, FSF France later discovered that Edu4 had removed

75 One of these projects has become a general purpose GNU/Linux distribution for embedded devices and the Free Software commons were not only defended by the court, but expanded it into the realm of routers, switches and embedded devices of all kinds. See http://openwrt.org/

> copyright and license notices in the software. All of these activities violate the terms of the GNU GPL. AFPA filed suit in 2002 to protect its rights and obtain the source code" (FSF France 2009).

When the case was finally concluded in the Paris Court of Appeals on September 22, 2009, with no further appeals possible, the GPL was once again upheld on the basis of copyright law. However, in this ruling a new aspect to the defence of Free Software was established. It was not a developer, whose code and freedoms had been violated, but an end-user who filed suit and won:

> "[W]hat makes this ruling unique is the fact that the suit was filed by a user of the software, instead of a copyright holder. It's a commonly held belief that only the copyright holder of a work can enforce the license's terms - but that's not true in France. People who received software under the GNU GPL can also request compliance, since the license grants them rights from the authors" (ibid.).

This illustrates that users are as much part of the software commons as the developers, in legal terms, since they too are granted the rights articulated in the GPL and can act on them and have their claims validated in a court of law. Moreover, since no further appeals are possible, this sets a legal precedent: in future legal proceedings in France, the GPL, on the basis of its clever relation to copyright law, ought to be automatically upheld with reference to this case. Given that intellectual property laws, through particularly TRIPs and WIPO (see Chapter 1), are increasingly global in nature and sought to be harmonised across national borders, a precedent in a leading industrial nation like

France might also carry a certain weight in other jurisdictions. Certainly the argument can be recycled in different national contexts by software commoners.

We can conclude that on the basis of the hacker customs and cyberspace values that self-organised voluntary associations are emerging to protect the four freedoms of software. The defence and enforcement of the GPL helps build a sustainable community which is capable of interfacing with the external judiciary to successfully translate the visions from within in relation to the old ways without.

The Statute of Anne, from which there is straight line to the modern concept of copyright, reflected the revolution of the printing press. The GPL and Copyleft reflect the revolutionary way in which information can be shared in cyberspace. It is an expression of needs and desires in a new technological environment:

> "Once copying is a useful and practical activity for ordinary people, they are no longer so willing to give up the freedom to do it. They want to keep this freedom and exercise it instead of trading it away. The copyright bargain that we have is no longer a good deal for the public, and it is time to revise it—time for the law to recognize the public benefit that comes from making and sharing copies" (Stallman 1996).

Ten years later Stallman's brain child prevailed in court.