CHAPTER 1

Introduction: Open Source Software and the Digital Commons

In March of 2012, The Linux Foundation released a report entitled, 'Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It'. The kernel is an essential part of an operating system that facilitates communication between computer hardware and software, and the Linux kernel development project is considered 'one of the largest cooperative software projects ever attempted' (The Linux Foundation, 2012: 1). Aside from a technical overview of how kernel development has changed over time, the authors included a curious note in the report's highlights: Microsoft was one of the top 20 contributors to the kernel. This marks the first time that Microsoft appeared as a top contributor, but it was not the only corporation in the top 20. Other corporate contributors included Intel, IBM, Google, Texas Instruments, Cisco, Hewlett-Packard, and Samsung, as well as others. The Linux operating system is a form of Free (Libre) and Open Source Software, or FLOSS, which allows users to freely study, use, copy, modify, adapt, or distribute the software. Why, then, would major corporations contribute directly to a FLOSS project, especially when that project seemingly does not directly contribute to corporate profits? This question becomes even more curious when one considers that many of the companies contributing to the kernel not only compete with one another in the market for information technology, but that companies like Microsoft and Google are direct competitors with Linux in the market for operating systems.

Indeed, Steve Ballmer, the Chief Operating Officer of Microsoft, once referred to Linux as 'a cancer that attaches itself in an intellectual property sense to everything it touches' (Greene, 2001). Ballmer was referencing the GNU General Public License, or GNU GPL, which is the most commonly used free software license. The GPL grants users of GPL-protected software the right to study, use, copy, modify, or adapt the software as they wish. In addition, users are granted the right to redistribute the software, as well as a modified version, and the user

How to cite this book chapter:
Birkinbine B. J. 2020. *Incorporating the Digital Commons: Corporate Involvement in Free and Open Source Software*. Pp. 1–32. London: University of Westminster Press. DOI: https://doi.org/10.16997/book39.a. License: CC-BY-NC-ND 4.0

may even charge a fee for the modified version, provided that the distributor does not place greater restrictions on the rights granted by the GPL. The GPL does not preclude corporations from modifying free software or charging a fee for their modified versions, but the corporation must still grant free software rights to end users. Ballmer's quote implies that free software is antithetical to commercial software companies. If this were the case, then Microsoft and other commercial software firms would have no incentive to contribute directly to one of the largest open source projects.

Furthermore, consider the fact that Ballmer made his denunciation of Linux on 1 June 2001. Merely 27 days later, on 28 June 2001, the United States Department of Justice found Microsoft guilty of monopolistic business practices in violation of the Sherman Antitrust Act primarily for bundling its Internet Explorer web browser with its Microsoft Windows operating system to rapidly increase its share of the market for web browsers. However, Microsoft has dramatically changed its position on Linux and open source since 2001, as signified by its inclusion in the top 20 contributors to the Linux kernel in 2012. That same year, Microsoft created Microsoft Open Technologies, Inc., a wholly owned subsidiary dedicated to facilitating interoperability between Microsoft and non-Microsoft technologies, while promoting open standards and open source. What changed during this 12-year period that Microsoft would so dramatically reposition itself in relation to FLOSS?

Microsoft is not alone. Indeed, corporate involvement in FLOSS has been increasing, especially since about 2007–2008. Table 1.1 provides an illustration of the companies that contributed to Linux kernel development for versions 4.8–4.13, which were released in 2017. The annual report for kernel development that year identified 225 companies that contributed to the project. While the Linux kernel is just one example of a FLOSS project to which corporations are contributing, other examples exist as well. This begs the question as to what motivates these companies to contribute to FLOSS projects. Furthermore, in what ways are they contributing to FLOSS projects? How do communities of FLOSS developers negotiate corporate involvement in their projects? Do communities of FLOSS developers have any recourse for unwanted corporate involvement or influence in their projects?

1.1. The Argument and Plan for the Book

The overall purpose of this book is to investigate the seemingly contradictory relationship between FLOSS communities and for-profit corporations. Working from a critical political economic perspective, I investigate the power dynamics that exist between communities of FLOSS developers and the corporations that sponsor FLOSS projects or appropriate the software production of FLOSS labourers. After all, FLOSS products and the productive process that make those products possible have been widely lauded as revolutionary changes that

Table 1.1: Top Companies Contributing to the Linux Kernel, Versions 4.8–4.13 (Corbet & Kroah-Hartman, 2017: 14).

Company	Changes	Percent
Intel	10,833	13.1%
none	6,819	8.2%
Red Hat	5,965	7.2%
Linaro	4,636	5.6%
unknown	3,408	4.1%
IBM	3,359	4.1%
consultants	2,743	3.3%
Samsung	2,633	3.2%
SUSE	2,481	3.0%
Google	2,477	3.0%
AMD	2,215	2.7%
Renesas Electronics	1,680	2.0%
Mellanox	1,649	2.0%
Oracle	1,402	1.7%
Huawei Technologies	1,275	1.5%

enable greater degrees of freedom and autonomy on behalf of users and contributors (Benkler, 2006; Raymond, 2000; Stallman, 2002). This project intervenes in these debates by tempering these claims. I position technology as a site of social struggle, and I contextualise commons-based peer production within a broader social context to illustrate how such production intersects with capitalist production. I do this by demonstrating how the purportedly revolutionary changes brought about by FLOSS and commons-based peer production are now becoming incorporated into corporate strategies and corporate structures.

The central argument presented here is that free and open source software is dialectically situated between capital and the commons. On the one hand, communities of programmers are actively working to create software as digital commons that can be accessed, used, adapted by others. By developing software iteratively this way, the pace and scale of software production increases. This represents a virtuous cycle whereby an association of software programmers actively contribute to a community that claims collective ownership over FLOSS projects. As such, FLOSS programmers can be framed as commoners insofar as they remain committed to ensuring the reproduction and sustainability of commons-based software projects over time. On the other hand, capital attempts to capture the value being produced by FLOSS communities. This includes harnessing the processes (i.e. the collective labour, or commons-based

peer production power) involved in FLOSS production as well as commodifying the products (i.e. specific FLOSS projects), which can provide a basis upon which to commercially exploit the collaborative production occurring in FLOSS communities.

This is not to say that the goals of the free software commoners and capitalist firms are always antagonistic. At times they are mutually beneficial, and researchers have demonstrated how commercial sponsorship of FLOSS projects tends to make those projects more likely to attract developers and, therefore, ensures the project's longevity (Santos, Kuk, Kon and Pearson, 2013). However, we also have other examples of these relationships breaking down, particularly when it concerns the unwanted encroachment of capital upon commonly held resources like the digital commons. In these situations, the interests of the FLOSS community diverge from those of a commercial sponsor, and the relationship becomes antagonistic. The FLOSS community is faced not only with the challenge of ensuring that their digital commons remain viable, but also with ensuring that the project maintains the sense of community that enabled the project to grow in the first place. How, then, to negotiate the relationship between their digital commons and the unwanted intrusion by capital into their projects? There are a variety of factors to consider when attempting to negotiate this relationship, and the subsequent chapters provide empirical evidence for how these dynamics manifest.

The commons, generally, and the digital commons, more specifically, can be understood as an alternative system of value that is emerging from within capitalism. At times, circuits of commons value can intersect with capital accumulation circuits. Therefore, understanding the relationship between free software and capital dialectically is useful for accounting for the contradictions between these two forces that operate according to differing logics. Chapter 2 outlines these differences more specifically by drawing on theories of capitalism, digital labour, and the commons. The purpose is to develop a critical theory of the digital commons by incorporating a critique of capitalism within theories of the commons.

In Chapters 3–5, I provide three detailed case studies that illustrate different aspects of the dynamics between FLOSS communities and corporations. I separate my discussion of corporate involvement in FLOSS into three thematic areas, with each case study providing an exemplary case of these themes. The three themes are *processes, products,* and *politics*. When considered together, these three case studies are indicative of more general tendencies of corporate involvement in FLOSS projects. Furthermore, each case study offers a nuanced understanding of the complex way these dynamics work, and they allow for a detailed unpacking of some of the contradictions inherent in the relationships.

To begin, Chapter 3 focuses on Microsoft's contentious relationship with FLOSS. This relationship is indicative of the ways in which the *processes* involved in FLOSS production effectively ushered in a new era of industrial software production. While other companies demonstrated a willingness to cooperate

with FLOSS communities, Microsoft's dominance of the software market for personal computing during the 1980s and 1990s makes it an instructive case for understanding how software production changed over time. The major historical event here is the antitrust ruling against Microsoft, which marked the end of an era in which software production was largely accomplished within a single firm that sought to exclude others from accessing its code. Indeed, one of the consent decrees in the Microsoft antitrust ruling was that Microsoft provide third parties access to its application programming interfaces (APIs). This was a radical departure from Microsoft's earlier practices, whereby the firm rose to power by using anticompetitive business practices.

Coinciding with Microsoft's dominance of the software market and its eventual antitrust conviction in the 1990s were other software firms trying to find a way to transform FLOSS products into successful commercial products. My analysis of Red Hat, Inc. in Chapter 4 is indicative of how FLOSS *products* get incorporated into a commercial firm's overall business strategy. Red Hat remains the largest and only publicly traded company providing software and services that are completely based on free software. As such, Red Hat cannot rely on traditional copyright protections to exclude others from using the underlying source code included in its software. Thus, my analysis of the firm explores how Red Hat has been able to create a profitable business based on free software.

Finally, the third case study in Chapter 5 focuses on how FLOSS communities cope with unwanted corporate influence in their projects. Sun Microsystems was an important corporate sponsor of FLOSS projects, but it was acquired by the Oracle Corporation, which had different plans for those projects. In that chapter, I focus on the diverse destinies of three such projects – the OpenSolaris operating system, the MySQL relational database management system, and the OpenOffice productivity software – and the ways that the communities involved in those projects resisted Oracle's encroachment into their projects. In effect, the case study illustrates the *politics* involved in negotiating boundaries between FLOSS communities and corporations, while also demonstrating some of the strategies FLOSS communities can use to protect their projects.

In the remainder of this introduction, I provide more context for understanding the significance of FLOSS. This includes historically situating FLOSS within a broader discussion of the commons, as well as some of the key historical moments in the development of software, generally, and FLOSS, more specifically. In each of these sections, I also offer some notes on the terminology used throughout the book, which will hopefully assist in avoiding conceptual confusion. Following those sections, I discuss the cultural significance of FLOSS. I conclude the chapter with a note on the methodology used for the current study. Readers who are already familiar with the history of FLOSS and its defining characteristics may wish to skip directly to the next chapter or the note on methodology at the end of this chapter.

1.2. Situating Free (Libre) and Open Source Software

Although free software and open source communities are related and, in some cases, not mutually exclusive, each of them has distinct characteristics that can best be described by reference to the ethos underlying each movement. To contextualise the emergence of FLOSS within the evolution of the computing and software industries, a brief history of these industries is provided below. Following that discussion, I focus on situating two key figures associated with FLOSS within their historical context: Richard Stallman and Linus Torvalds. These two figures represent free software and open source, respectively.

1.2.1. Historicising Free and Open Source Software

The use of machines for processing information or calculating differences in numbers, human beings performed such work. But human calculations were, at times, prone to errors. To reduce this uncertainty, Charles Babbage, a philosopher and mathematician working at the University of Cambridge in 1822, proposed that it was 'only by the mechanical fabrication of tables that such errors can be rendered impossible' (Gleick, 2011: 95). Such was the proposition for Babbage's Difference Engine, which performed routinised calculations mechanically, and was arguably the genesis for modern computers as we know them today. Later, Babbage expanded on his idea and planned a new type of machine that was capable of being controlled by instructions that could be encoded and stored to facilitate operation. The new iteration of the idea was called the Analytical Engine, but this still only provided the idea for the hardware or mechanisms necessary for such processes to occur. What was needed for this hardware was software.

The idea for software arguably originates with Augusta Ada Byron King, the Countess of Lovelace, otherwise known simply as Ada Lovelace. In 1843, she developed the idea that Babbage's Analytical Engine could perform a series of operations beyond the mere calculation of numbers. By abstracting from the differences between two things, Lovelace posited that the Analytical Engine could be programmed to perform operations that relied on symbols and meanings, which, in turn, could be communicated to the machine. Although Lovelace's idea was never realised in her lifetime, she is credited with developing the idea for software and is known as the first programmer.

While Babbage and Lovelace are credited as pioneers in developing the ideas for modern computers and software, the construction of such machines did not begin until World War II. Developments in the field of computer science and information theory – like Kurt Gödel's incompleteness theorem, Alan Turing's idea for a Universal Turing Machine, Claude Shannon's mathematical theory of communication, and Norbert Wiener's cybernetics – provided the intellectual inspiration for the development of such machines. Before, during, and after

World War II, many of the developments leading to modern computers were used for military purposes. Most notable, perhaps, were the German Enigma machine that was used to encrypt secret messages and the electromechanical bombes used by the United Kingdom to decipher those messages (Smith, 2011). However, in 1941, Konrad Zuse, a German electrical engineer, built the Z3, which is regarded as the first electro-mechanical, programmable, fully automatic digital computer (Zuse, 1993). The first comparable computer in the U.S. was developed by John Atanasoff at Iowa State University in 1942 (Copeland, 2006). Only one year later, the first fully functioning electronic digital computer was put to use by the cryptanalysts working at Bletchley Park in the U.K. as part of the Government Code and Cypher School. The Colossus, as the new machine was known, was programmed to decipher German communications during the war. By the end of the war, Bletchley Park had 10 Colossi working to decode German communications (Copeland, 2006).

Following these initial landmarks, the development of modern computers accelerated as many of the early pioneers began working for academic institutions and private companies after the war. In the United States, Grace Hopper, who served in the United States Navy Reserves as a member of the Women Accepted for Voluntary Emergency Service (WAVES) during World War II, was assigned to the Bureau of Ships Computation Project at Harvard University. While there, she worked on the Mark I computer project, which was built by IBM in 1944. Later, after she began working for private companies, Hopper popularised the idea of machine-independent programming languages. This led to the development of the Common Business-Oriented Language (COBOL) in 1959. Hopper is also credited with popularising 'debugging' as a term for removing defective material or code from a program. While Hopper may not have invented the term, she popularised it by literally removing a moth from a Mark II computer at Harvard University after it had caused the machine to short circuit (Deleris, 2006).[1]

During the 1960s, the creation of microprocessors drastically reduced the cost of computing. As a result, communities of hobbyist programmers and computer enthusiasts began to experiment with the technology in the following years. One notable example was the Homebrew Computer Club, started by Gordon French and Fred Moore in 1975 at the Community Computer Center in Menlo Park, California. The club provided an open forum for hobbyists to trade parts and advice about the construction of personal computers. The goal was to make computers more accessible to others. More will be said about this specific hobbyist community in Chapter 3, as it played an important role in the rise of Microsoft. Aside from these hobbyist communities, the majority of computer development occurred within the military, academic institutions, and private companies.

Most notable were the initial developments within the Defense Advanced Research Projects (DARPA), which was created in 1958, as well as the Artificial Intelligence Lab at the Massachusetts Institute of Technology (MIT), which was

founded in 1970.[2] Programmers working at the time were using a proprietary programming language called Unix, the intellectual property rights for which were owned by AT&T. One of the programmers working at MIT was Richard Stallman, who began working in the lab in 1971. Stallman found that when he wanted to work with the Unix programming language outside of officially sanctioned spheres, he was denied access to the code by AT&T. In protest, he posted messages to computer-based bulletin boards in 1983 announcing that he was developing a Unix-based language that would be available for free so that others could use the language however they saw fit. In 1985, Stallman published 'The GNU Manifesto', which outlined the goals of his new project, his reasons for developing the project, and what the project was aimed at fighting back against.[3] The programming language was called 'GNU', a recursive acronym standing for 'Gnu's Not Unix'. Along with the programming language, Stallman developed the GNU Public License (GPL), which stipulated that anyone could access the source code for free, and that anyone using the GPL agreed to make their contributions available under the same conditions. This would ensure that computer programmers could freely share their work with one another, thereby creating a common form of property that developed in opposition to its proprietary and closed counterparts.

Stallman became the figurehead of the movement against proprietary software. He viewed access to source code as a fundamental right, which he wanted others to believe in as well. He summed up this view in his famous dictum, 'Free as in freedom, not as in free beer', thus positioning free software as a moral right (Stallman, 2002). The free software definition stipulates that 'users have the freedom to run, copy, distribute, study, change and improve the software' (Free Software Foundation, 2012). As the principles of free software grew beyond the borders of the U.S., others have tried to reduce the confusion over the English term 'free' by using the French term *libre* rather than *gratis*. Stallman established the Free Software Foundation (FSF) to promote his movement against proprietary software, and he represents an impassioned counter-cultural figure who continues to espouse his free software philosophy.

While Stallman is generally considered to be the figurehead of the free software movement, open source software is generally associated with Linus Torvalds. In many ways, Torvalds and Stallman have similar stories, but differ on philosophical terms. During the 1980s, free software projects were being developed but generally on a smaller scale. Free software had not yet found a way to coordinate efforts on a larger scale. Torvalds wanted to work on kernel development for an open-source operating system. Rather than relying on numerous programmers all working independently on such a task, Torvalds released the source code for his project, which he was calling 'Linux', a portmanteau of his name, Linus, and the language he was working with, Minix (itself a simplified derivative of AT&T's Unix). Torvalds suggested that anyone who was interested in contributing to such a project was encouraged to do so, if they released their work back to the community so that others could progressively work toward

completing the kernel. The project proved to be successful, and eventually led to the creation of the open source operating system, Linux. Coordinating such a large-scale programming project was accomplished by asking those working on the code to release their work, no matter how small the changes seemed. The rationale was that coordinated efforts reduce the amount of redundant work, which was summed up in the adage 'with many eyes, all bugs are shallow', which Eric Raymond refers to as 'Linus's Law' (Raymond, 2000).

Stallman and Torvalds differ with respect to how they view the relationship between free software and proprietary software. Whereas Stallman tends to be somewhat more confrontational in his opposition to proprietary software, Torvalds is less so. Williams (2002) describes a decisive moment at a conference in 1996 where Stallman and Torvalds appeared on a discussion panel together. Torvalds expressed admiration for the work that Microsoft was doing and suggested that free software advocates could work together with companies. Such a suggestion was generally seen as taboo since Stallman was perceived with esteem by the programming community, and the Free Software Foundation generally took a very adamant stance against proprietary software companies. Powell (2012) frames this distinction between free software and open source similarly:

> open source software as an industrial process grew out of the culture of free software development, but departed from the latter's political focus on the value of sharing and the maintenance of a knowledge commons, and instead focused on the efficiency of open source processes for software production (692).

This moment at the 1996 conference thus marked a watershed moment in which the fervour of the free software movement thawed a bit, as Torvalds came to represent a more liberal approach to free software. By 'liberal' here, I am referring to the literal definition rather than a specific political position; the term should be understood as something that indicates an openness to new perspectives or behaviours while willing to abandon traditional values. In this regard, Linus's expression of support for the work that Microsoft was doing signalled an openness to working with Microsoft (or other commercial firms) simply to produce the best software rather than an adherence to the anti-corporate stance of Stallman and the Free Software Foundation.

In sum, then, we can understand the free software and open source movements with respect to these differing philosophical positions. Stallman and free software advocates tend to make moral claims against supporting proprietary software, while Torvalds and open source tend to be associated with a more liberal and inclusive stance. While Stallman and Torvalds have been used to illustrate the differences between free software communities and open source communities, they should not be viewed as mutually exclusive communities, nor should they be seen as representative of the entire free software

and open source communities. One of the peculiarities of the free and open source software community is that, although the overall community is united in their belief that software ought to be free for users to study, modify, adapt, or customise, its members will often vehemently defend their preferred free software project while deriding others. In a sense, this signals to others where their loyalties lie and engenders stronger ties within niche communities that exist within the larger FLOSS community. The present project is less concerned with these intra-group fissures than the relationship of the community to the corporations that rely on their labour. To that end, the combined term 'Free (Libre) and Open Source Software' or 'FLOSS' is used to refer to the overall community.[4]

1.2.2. The Unseen Ubiquity of Free and Open Source Software

From its beginnings in the 1980s and 1990s, FLOSS has proved to be an efficient and effective way of producing software. Whether we realise it or not, most of us rely on FLOSS in our everyday computing, as it provides critical infrastructure that enables the Internet to function. As an example of the size and scope of some FLOSS projects, consider the Linux kernel, which was discussed in the introduction to this chapter. When it was first released in 1991, the Linux kernel featured approximately 10,000 lines of code. Version 4.13 of the Linux kernel was released in September 2017 and featured nearly 25 million lines of code, which was produced by nearly 1,700 developers and 225 companies (Corbet and Kroah-Hartman, 2017: 11). Furthermore, Linux has become widely used as an operating system. For example, Linux (or other operating systems derived from Linux) holds 100% market share in the market for supercomputer operating systems (Top500.org, 2018a). These computers are the most powerful computers in the world, and all of them rely on Linux or Linux-based operating systems. This includes the United States Department of Energy's supercomputer at the Oak Ridge National Laboratory in Oak Ridge, Tennessee, which at the time of writing is home to the world's fastest and most powerful supercomputer (Top500.org, 2018b).[5] While Linux does not yet have a significant share of the personal computing desktop market, the operating system has been customised and used within a variety of contexts.

Within the United States, Linux is used for high-level military operations. For example, the United States Navy announced that its $3.5 billion warship, the USS *Zumwalt*, which has been described as 'the most technologically advanced surface ship in the world', will effectively serve as an armed floating data centre that features server hardware running various Linux distributions and more than 6 million lines of code (Mizokami, 2017, Gallagher, 2013). In addition, the International Space Station switched from the Windows operating system to Debian Linux, according to Keith Chuvala, the Manager of Space Operations Computing at NASA, because they wanted to have '…an operating

system that was stable and reliable – one that would give us in-house control' (Bridgewater, 2013).

Indeed, Linux and Linux-based systems also provide essential components for some of the most recognisable technology companies, which was discussed briefly at the beginning of this chapter. Despite the fact that I have only selected a few companies for detailed examination in the subsequent chapters, one could find other similarly intriguing case studies that would exemplify different dynamics between corporations and FLOSS communities. As such, it is worth mentioning some notable examples here simply to emphasise the ubiquity of Linux. Google's Android operating system, for example, is one of the world's most popular mobile platforms, and it is based on the Linux kernel. However, there are certain key components of the Android operating system that remain proprietary to Google (see Amadeo, 2018). Aside from Google, other companies like Canonical rely on Linux for creating customised operating system distributions. Canonical produces Ubuntu, which is one of the most widely used Linux distributions.

Linux has also seen widespread adoption around the world. Some countries have developed their own versions of Linux to meet specific needs, and some cities have even required that Linux be given preference over other operating systems. For example, between 1999–2001, four cities and municipalities in Brazil – Amparo, Solonópole, Recife, and Ribeirão Pires – passed laws that required government agencies to use or give preference to Linux (Tramontano and Trevisan, 2003; Festa, 2001). The decision to switch to free software systems was mainly economic, as Brazil reported spending nearly $1 billion on software licensing fees to Microsoft between 1999–2004 (Kaste, 2004). By switching to free and open source software, Brazil estimated that they could save approximately $120 million per year (Kingstone, 2005). Brazil remains one of the more progressive countries in its support of free software (see Birkinbine, 2016a; Schoonmaker, 2018; 2009). Many of the country's policy measures and initiatives related to FLOSS have been driven by communities of activists who have been able to intervene in policymaking processes to institute policies that seem to contradict the prevailing neoliberal ideology. In an excellent article on the subject, Shaw (2011) framed these activists as *insurgent experts*.

Similar measures to support free software were taken in Kerala, India, as the state adopted a policy to remove proprietary software from its educational system. According to one estimate, the switch saved the state of Kerala roughly $58 million each year (Prakash, 2017). The German city of Munich developed its own version of Linux called LiMux (Linux in Munich), which it used as an operating system for its 15,000 city council members before announcing a shift back to Microsoft in 2017 (Heath, 2017). The National University of Defense Technology in China has also developed its own Linux-based operating system called Kylin. In addition, the computers used for the One Laptop Per Child project, which was founded with the goal of bringing low-cost computers to developing countries for educational purposes, featured a free and open source

operating system based on Fedora, the free software project sponsored by Red Hat, Inc., which will be discussed in Chapter 4.

Beyond the increasing use of Linux, open-source principles have been used in areas outside of information technology. For example, open source hardware (see Söderberg, 2011) can increase access to physical goods, including furniture, musical instruments, construction materials, and wind turbines for generating renewable energy. Such projects are particularly attractive to those living in developing countries, where access to information, goods, and services may be restricted or limited. One of the more ambitious projects in this area is the Open Source Ecology project, which offers 'open source blueprints for civilization,' and includes instructions for building industrial machines with recycled or low-cost materials (Open Source Ecology, 2019). While this is just one notable example, it demonstrates the optimism and creativity involved in applying open source principles to a whole way of living rather than simply information technology. However, the core values inherent in these projects do not necessarily originate in open source software. Rather, the cultural values of openness, sharing, mutual aid, respect, and conviviality are foundational values for building a community. When applied on a broader scale, these principles hold the promise of a more sustainable future, especially when such principles are linked with environmental and ecological preservation practices. But these principles only become radical propositions in a system that discourages or provides little incentive for valuing them.

Despite the fact that FLOSS communities comprise a socio-technical system insofar as their activities are made possible by and exist within a technologically mediated realm, FLOSS enthusiasts also congregate and cooperate in-person through a network of Linux User Groups (LUGs) around the world. Regular meetings of LUGs are held to promote FLOSS, to assist new users with installing FLOSS, to troubleshoot any issues that may arise when using FLOSS, or to simply meet other people interested in FLOSS. In this sense, the social connections that exist within these groups are mediated by their mutual interest in technology. Because members of the FLOSS community are brought together by their mutual appreciation of technology, their cultural practices depend upon and are supported by interconnected network technologies. As more people become connected to the network, the opportunities for additional participants in these communities grow.

One final point deserves attention here too. It seems like an increasing amount of our social lives is spent on the Internet where we work, communicate with friends and colleagues, read news, watch movies and television, and listen to music, among other activities. When we connect to the Internet and visit websites, our requests for information are relayed through a network of interconnected servers that facilitate communication between other clients on the network. The operating systems running those servers are increasingly FLOSS projects like Linux or FreeBSD, but Microsoft also designs server software. This provides another example of FLOSS projects competing with proprietary

companies like Microsoft. Consequently, and whether we realise it or not, our ability to connect to the Internet may depend, in part, on the ability of FLOSS projects to work together with proprietary software. This further demonstrates the need for understanding the ways in which proprietary software and FLOSS projects work together, as well as what happens when these relationships break down. Unpacking the dynamics that exist in these relationships can help us understand either the enabling or constraining of our ability to connect with others online.

What these examples should illustrate is that Linux but also FLOSS more generally has become more than just a tool used within the computer hobbyist community. Its widespread and increasing adoption across the globe within a variety of high-level contexts demonstrates the power of the FLOSS production model as well as the effectiveness of its products. As FLOSS continues to be used within an increasing variety of contexts, understanding the ways in which corporations, governments, non-profit organisations, and other types of institutions are involved in FLOSS projects will become increasingly important. Therefore, FLOSS provides an important area for research not just because of its increasing ubiquity, but also because of the claims that have been made about the democratic, egalitarian, and non-market characteristics of its products and processes. This is precisely how this project seeks to contribute to such debates.

1.2.3. FLOSS and Hacker Culture

The term 'hacker' has taken on negative connotations recently, but the term is generally used to describe anyone who 'tinkers' with or makes changes to technology to create something new. Steven Levy (1984) outlined the principles of the hacker ethic. Among other elements, Levy claimed that computers can be used for creative purposes, hackers ought to be judged by the quality of their work rather than any other characteristic (gender, race, ethnicity, etc.), and that having the ability to hack is a prerequisite for hacking. This last caveat may seem obvious but, in order to perform a hack, a hacker must have access to the technology (in this case, the source code). In other words, closed, proprietary technologies that do not allow for tinkering may be viewed as unjust.

Indeed, when faced with closed, proprietary, or otherwise secured technologies, a hacker may attempt to circumvent or remove those restrictions. At times, this is done to make a point about information security, but it is also done to signal to others that they deserve credit for the sophistication of their hack. This signalling motivation is also recognised within open source software communities (Lakhani and Wolf, 2005), especially because FLOSS programmers are interested in remixing, modifying, adapting, or creating something new from a given product. The same signalling motivation has been used to understand why programmers contribute to FLOSS projects. Lakhani and Wolf (2005) explain that signalling can take place within at least a couple of levels. At

the level of the individual, a single hacker may perform a hack to signal his or her skills to others. Hackers might also use this type of signalling to communicate their skills to potential employers to secure paid employment. Gaining recognition within the broader community for performing certain programming tasks effectively can translate into increased job opportunities with companies looking for specific skills.

However, a different type of signalling takes place between groups of hackers. Groups or collectives may signal their prowess to others by shutting down a web site or otherwise disrupting services. Often, this is done in the spirit of competition, but can also be explicitly driven by a particular ideology. For example, nationally based hacker groups can be found in Syria where a pro-Syrian government hacking group called the Syrian Electronic Army has waged hacking battles against the pro-rebel hackers associated with the Free Syrian Army (Fitzpatrick 2012). In these situations, hacker groups strategically target the web sites of their opponents to signal the strength of their movement.

Although the signalling appears to be the most prevalent motivation, Weber (2004) identifies other motivations as well. In a survey of self-identified hackers, respondents reported their primary motivation for contributing to FLOSS development was a desire to challenge oneself and perform creative work. This seems to support what Levy (1984) identified as primary tenets of the hacker ethic: creativity and aesthetics. Weber (2004) also found additional motivations reported in the survey, including the belief that all software should be free, which echoes the philosophy of Richard Stallman and the Free Software Foundation. Weber concludes that motivations are diverse and that the results from these surveys need to be properly contextualised. For instance, many contributors to FLOSS development do not disclose their identity or any institutional affiliation. Indeed, a look at the credits file for users contributing to the development of the Linux kernel shows that most contributors are listed in the 'unknown' category. This means that a large portion of the FLOSS community simply chooses not to self-identify. Therefore, the results of any survey that claims to represent the entire FLOSS community must be approached somewhat sceptically.

While signalling and creativity are certainly important factors for understanding the motivations of hackers and FLOSS contributors, my own view is that the most robust scholarship on the cultural significance of free software and FLOSS production comes from Christopher Kelty. Kelty (2008) positions free software as a *recursive public*, which he defines as:

> 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 (Kelty, 2008: 3).

In other words, in the process of actively contributing to FLOSS projects, FLOSS programmers actively create, recreate, or reproduce the infrastructure that enables their activity to take place. This has conceptual links with other theories of the commons that position the commons as a process or a way of becoming (Dyer-Witheford, 2006; Linebaugh, 2008; Singh, 2017). Similarly, Rossiter and Zehle (2013) argue the commons are not purely 'given as a fragile heritage to be protected' against enclosure, but they must be actively constructed. FLOSS communities actively produce the digital commons as code, which is produced and licensed under intellectual property licenses that permit users to use the code and adapt it for their own purposes. These alternative intellectual property licenses take many different forms. The original copyleft licence to see widespread use was the GNU General Public License.[6] Other notable examples are the Creative Commons[7] licences, which allow varying levels of use for the protected property under conditions set by the creator. For example, users may make their creation freely available and permit others to use it, if those users provide attribution to the original author.

Kelty (2008) furthermore claims that FLOSS programmers 'do not start with ideologies, but instead come to them through their involvement in the practices of creating Free Software and its derivatives' (7–8). Coleman (2004) makes similar claims when she refers to the 'political agnosticism' of FLOSS. The complex forces at play in this agnosticism stem from an outward denial of specific political affiliations even while 'political denial is culturally orchestrated through a rearticulation of free speech principles, a cultural positioning that simultaneously is informed by the computing techniques and outwardly expresses and thus constitutes hacker values' (Coleman, 2004: 509). Coleman continues by explaining that the core of the moral philosophy espoused by the FLOSS community is a 'commitment to prevent limiting the freedom of others' (509). This utilitarian ethic of openness is what is necessary for FLOSS programmers to continue building state-of-the-art computer programs because it is precisely the ability to tinker, adapt, and improve upon software that enables innovation to occur within software development.

These principles, as well as the outward denial of a specific political position, are, in part, what has enabled the FLOSS community to attract such a large community. Of course, this is not to say that all members of the FLOSS community reject specifically political ideologies. One needs to look no further than Eben Moglen's (2003) 'dotCommunist Manifesto', which offers a polemic against the regimes of private property. Indeed, he concludes the manifesto with the following seven principles in the struggle for 'free speech, free knowledge, and free technology' as well as a concluding note on how this struggle will bring about a more just society:

1. Abolition of all forms of private property in ideas.
2. Withdrawal of all exclusive licences, privileges and rights to use electromagnetic spectrum. Nullification of all conveyances of permanent title to electromagnetic frequencies.

3. Development of electromagnetic spectrum infrastructure that implements every person's equal right to communicate.
4. Common social development of computer programs and all other forms of software, including genetic information, as public goods.
5. Full respect for freedom of speech, including all forms of technical speech.
6. Protection for the integrity of creative works.
7. Free and equal access to all publicly produced information and all educational material used in all branches of the public education system.

By these and other means, we commit ourselves to the revolution that liberates the human mind. In overthrowing the system of private property in ideas, we bring into existence a truly just society, in which the free development of each is the condition for the free development of all.

(Moglen, 2003)

Similarly, Dmitry Kleiner's (2010) *Telekomunist Manifesto* outlines proposals for developing a working class politics online. His proposals for venture communism as well as a copyfarleft licensing regime offer concrete proposals for developing alternatives within existing frameworks, but doing so in a way that is guided by radical politics. Both of his proposals are aimed at preserving and protecting the commonly held property of independent producers from capitalist exploitation or co-optation.

It is precisely because the collective productive activity of the FLOSS community is so valuable for software production that capitalist firms are interested in harnessing this power. At the same time, this is also the reason that critical scholars like Kleiner have sought ways to preserve that value within the communities who create such value, even if they offer different proposals for how to do so. Taken as a whole, then, this community holds tremendous value for software production. The authors discussed above, particularly the work of Kelty (2008) and Coleman (2004; 2013), offer some of the best work for understanding the cultural significance of FLOSS as well as the ethics underlying the FLOSS community. However, there is still the pressing question of what happens when the specific cultural, political, and economic values of the FLOSS community intersect with circuits of capital accumulation. This was one of the tensions that Kleiner (2010) was trying to address when developing his proposals for alternatives. Moreover, in what ways does the FLOSS community negotiate and justify the dual position of advocating for open knowledge and market success simultaneously? Some of the best work exploring the complex set of dynamics at work in this regard has been that of Alison Powell (2012; 2016; 2018). In exploring the ways that participants in peer production communities negotiate competing moral visions for their projects, Powell (2018) argues that participants often engage in 'operational pragmatics' that are used to justify various design decisions. In doing so, participants collapse

distinctions between advocacy for open knowledge and market success even if these distinctions seem to be at odds with one another. In effect, both are viewed as 'good' or virtuous, that function as 'regimes of justification' when making decisions about design (Powell, 2018: 514).

How, then, can we understand these complex and intertwined ways of negotiating cultural differences both within peer production communities as well as their intersection with capital accumulation circuits? Is it possible for peer production communities to be exploited by capital if they are willing participants in designing products for market success? After all, corporations are keenly interested in harnessing the productive power of the FLOSS community. The following section discusses one way to theorise the ways in which companies relate to FLOSS communities. However, the following chapter will discuss these specific dynamics in greater detail by drawing from theories of capitalism, digital labour, and the commons, while exploring the ways in which exploitation occurs when capital and the commons intersect.

1.3. Open Source Business Models

The previous section demonstrates how the specific cultural dynamics at play in FLOSS communities have been explored quite effectively by other scholars, including the significance of those dynamics for cultural production more broadly. However, the economic arrangements between corporate firms and FLOSS communities have been explored comparatively less. This book aims to offer some greater descriptive detail as to how these dynamics specifically manifest as FLOSS communities and corporations negotiate the boundaries between their respective organisations. However, one attempt to develop a typology of open source business models is worth mentioning here.

As part of their broader treatment of open source software, Deek and McHugh (2008) develop a typology of open source business models. The typology contains five different models that have been used in trying to profit from FLOSS. Table 1.2 provides an illustration of this typology, providing the types of business strategies employed, a description of the strategy, and an example of a company or product that is representative of the strategy.

The first business model relies on dual licensing, in which the owner of copyrighted software provides free and open distributions for non-profit users but requires for-profit customers to pay a fee to use the software. The exemplary case here is MySQL, which is an open source database management system. The company provides a free version of its software under the General Public License (GPL), which stipulates that any derivative software using the GPL-licensed software must also be made available under the same licence. MySQL also provides an advanced commercial version of its software to for-profit corporations, which can be customised to the users' specific needs or integrated with that company's proprietary software.

Table 1.2: Types of Open Source Business Strategies, adapted from Deek and McHugh (2008: 272).

Business Strategy	Description	Examples
Dual Licensing	Owner of copyrighted software provides a free and open distribution for non-profit users but requires for-profit customers to pay a fee to use the software.	MySQL
Consulting	Company assists other companies with planning, strategy, and implementing appropriate open source solutions within their business.	Olliance Consulting (division of Black Duck Software), LQ Consulting
Distribution & Services	Company provides services for non-expert computer users by handling the compilation of stable, updated, and prepackaged software suites that are distributed to users (clients).	Red Hat, Canonical
Hybrid open/ proprietary – Vertical Development	Using open source as a base upon which proprietary software can be built.	Google
Hybrid open/ proprietary – Horizontal Arrangements	For-profit company becomes directly involved in supporting open source projects to supplement its own business operations.	IBM, Microsoft

The second type of business model is one in which a company provides consulting services for FLOSS. Quite simply, companies that adopt this model assist other companies with planning, strategy, and implementing appropriate open source solutions within their business models. Among other things, Black Duck Software provides consulting services through its Olliance Consulting division.

The third business model is one in which a company provides FLOSS distributions and services, and the exemplary company here is Red Hat. Unlike MySQL, which owns the copyrights for its software, Red Hat creates and provides its own distribution of Linux. In addition, Red Hat provides training, education, documentation, and support for its Linux distribution. In other words, Red Hat provides a service for non-expert computer users by handling the compilation of stable, updated, and prepackaged software suites to be distributed to users. In some ways, then, Red Hat behaves similarly to a proprietary

software provider, except that it does not own the intellectual property rights for the software it sells and services. Rather, the company sells and provides its own Linux distribution, which it can do because of the open licensing model of Linux.

Whereas the first three business models are solely related to FLOSS, the remaining two rely on a hybrid of both open and proprietary software. The fourth model is a hybrid of both proprietary and open software that relies on vertical development with FLOSS. Vertical development means using open source software as a base upon which proprietary software can be built. One of the major corporations that uses this model is Google. In fact, Google does not sell its software at all; it develops and maintains its own software in-house, while selling services provided by its software to other customers. Of course, Google's search engine is proprietary, but Google uses the Linux core to support its proprietary search services.

The final model is a hybrid of proprietary and open software, but one in which the company relies on horizontal arrangements. This is the business model that lies at the heart of this book project. In these relationships, for-profit corporations become involved in open source projects. Drawing from Fogel (2005), Deek and McHugh (2008) claim that the reasons for corporate involvement are diverse, but include everything from spreading 'the burden, cost, and risk of software development across multiple enterprises to allowing companies to support open source projects that play a supportive or complementary role to their own commercial products' (277). IBM is one example of this type of business model. For example, IBM's WebSphere application, which enables end-users to create their own applications, was built using the Apache web server, which is open source. Thus, by supporting open source projects like Apache, IBM is indirectly supporting its own interests. Furthermore, IBM directly competes with Microsoft as a platform for applications. Because IBM supports Linux, it is not only investing in the reliability of its own products but may simultaneously weaken Microsoft's market position, especially because Linux is also a direct competitor of Microsoft.

In sum, then, this section has discussed how FLOSS has been used in differing ways by drawing on the typology developed by Deek and McHugh (2008). The most fruitful area of study for the purposes of this project was the hybrid open/proprietary model that relies on horizontal arrangements, although other projects are discussed, like MySQL, which represents other types of business strategies. The corporations that rely on horizontal arrangements are most interesting because of their direct involvement in FLOSS projects. Thus, these companies need to maintain a good relationship with the broader FLOSS community. When the norms of the community are violated by a company, the community can abandon a project, which can effectively end commons-based production on the project. In this sense, the FLOSS community leverages its collective labour power against undue corporate influence in its commons-based resources. This was the case when the Oracle Corporation acquired Sun

Microsystems. This case will be discussed in greater detail in Chapter 5. For now, however, it is important to note the two different examples of companies using hybrid horizontal agreements to two different ends. In the case of IBM, the company maintained a relatively stable relationship with the open source community. In the other, Oracle overstepped its bounds by violating the norms of the community. As more and more corporations become involved in FLOSS projects, the relationships that exist between the community and the corporations that rely on their collective labour power will be subject to changes.

1.4. FLOSS as Digital Commons

The seemingly contradictory relationship between FLOSS communities and corporations is further exacerbated by the fact that FLOSS has consistently been held up as the primary example of a digital commons. In medieval England, the commons referred to a portion of land owned by the lord of the manor, which certain tenants had the right to use for their needs. These rights included the right to cultivate soil, produce crops, feed livestock, and other activities. The concept has since been expanded from this very specific meaning to encompass any resource that is owned by a community or a resource that may be accessed by a broader community of people.

In tracing the roots of scholarship on the commons, most scholars bookmark the work of Elinor Ostrom (2005; 1990). The narrative often begins with Ostrom's work, and focuses on how her ideas developed and influenced subsequent generations of scholars.

While Ostrom is a towering figure in scholarship on the commons, this simple narrative tends to obfuscate the broader history and context within which Ostrom's work is situated. Locher (2016) clarifies this history by demonstrating how Ostrom's work can be contextualised within a broader history of scholarly debates within economic, political, and anthropological scholarship concerned with the best way to achieve development. These debates were concerned with the role of the state, the market, and local communities in the project of development during the post-World War II period. This scholarship can be linked with the United States' international development projects through its flagship institution, USAID, in the 1970s–80s.

Two assumptions in the approach to development dominated this period. One was the assumption of the 'tragedy of the commons' or the fallacy of collective action, based primarily on the work of Garrett Hardin (1968). Hardin argued that the commons were ultimately unsustainable because they were at risk of overexploitation as members of the community acted in their self-interest to maximise personal gain. Thus, there was a fallacy in the logic of collective action; it was simply impossible for communities to govern collective resources without overexploiting them. The second assumption was that the liberal technocratic state ought to be the central agent in development through

economic planning and coordinating large-scale development projects. This assumption was driven by the success of the New Deal and the welfare state in the post-war period. As such, the model was viewed as the primary means for developing countries in the Global South, where traditional practices would give way to modernisation to boost economic productivity.

During the 1970s, these assumptions were challenged by development anthropology, which analysed 'adaptive socio-ecological strategies' used by local communities to ensure the survival of ecological resources (Locher 2016, 313). Often, these decision-making strategies were situated within complex systems of customs and social rules that developed from local communities' historical experiences with their broader environment. Challenges to these assumptions continued in the 1980s as neoliberal economics emerged as an alternative to welfare state capitalism. Informed by rational choice theory, which privileged calculating and efficient economic decision-making by profit-maximising individuals, the goal was to unleash productive capacity in the private sector through deregulation and privatisation. Neoliberal doctrine thus argued for dismantling state regulation and withdrawing the state from social provision. As such, neoliberalism represented not just an economic doctrine but also 'an ethic in itself, capable of acting as a guide for all human action, and substituting for all previously existing ethical beliefs' (Treanor, 2005: n.p.).

It was within this context that Ostrom's scholarship, in collaboration with others, sought to illuminate the ways that local communities govern common-pool resources outside of the binary of either state provision or market relations. For example, Hess and Ostrom (2007) argued against the tragedy of the commons thesis by focusing primarily on two points: first, Hardin assumes that the sheep herders are acting according to the principles of neoclassical economics and are individually acting in their self-interest rather than allowing for forms of common governance, whereby concessions are made to the other sheep herders. Second, Hardin frames the issue within the binary choice between socialism and capitalism. However, the framing is fallacious for a couple of reasons. The commons under feudalism were owned by a private individual and not the state. Furthermore, Ostrom (1990) demonstrates how different types of commons can be governed collectively so individual short-term gains can be compromised for the long-term survival of the common resource. In effect, Ostrom (1990) provided some nuance to the way that we understand commons, especially because they were often placed in a binary opposition that was representative of Cold War-era assumptions about social development: either state provision of common property (socialism) or private property ownership (capitalism).

Ostrom focused on the diverse ways that different commons are managed by those communities that claim some sort of association to the resource. The types of common-pool resources governed in this way vary, but the initial focus was on natural resources like fisheries, grazing pastures, groundwater basins, and irrigation systems. Later, Hess and Ostrom (2007) would expand the study

of the commons to non-tangible resources like knowledge and information. Table 1.3 illustrates different types of property by providing a simple matrix of two factors: rivalry and excludability. Rivalry refers to the extent to which a resource is finite or requires reproduction. Highly rivalrous goods tend to be finite objects like apples, which need to be planted again to reproduce the crop, while low rivalry goods tend to be intangible goods that can be reproduced without much additional cost, like ideas, information, or knowledge. Excludability refers to the extent to which an owner of such goods can exclude others from accessing or using that good. Highly excludable goods are protected by private property rights, whereas goods with low excludability may be used by anyone. Following from these terms, the matrix for rivalry and excludability would look something like this:

Table 1.3: Typology of Property, adapted from Hess and Ostrom (2007) and Frischmann (2012).

		Excludability	
		High	**Low**
Rivalry	*High*	Individual Property (finite resource)	Common Property (infrastructure)
	Low	Intellectual Property (books, music, consulting)	Knowledge Commons or Digital Commons (language, knowledge, code, free software)

Within this typology, FLOSS is positioned as a knowledge or digital commons. Digitised knowledge – in the form of source code, README files, software packages, and the shared documentation required in collaborative production – is freely available for anyone to use and at no additional cost for reproduction. One of the unique characteristics of free software as digital commons is that it avoids the free-rider problem, whereby someone who consumes or uses a resource does not give back to the community. Even if a user of FLOSS projects does not have the capability to modify code, that person can still contribute to the community simply by using the software. As an example, consider someone using the Linux-based operating system, Ubuntu. That person would not need to pay for Ubuntu or any of the software included with the operating system, but the person can still use programs and report any flaws or 'bugs' they encounter when using the software. These can be reported back to the development community so someone within the community can work on fixing the issue. Ultimately, the fix to the software can be submitted to the project manager for inclusion in a subsequent release of the software, or the fix may be distributed as an update to all users. This process is reflective of the adage 'with many eyes, all bugs are shallow' (Raymond, 2000) which makes it possible for the programs and operating system to maintain a high quality over time. In effect, the use of free software serves as a form of quality control.

Thus, free software may be positioned as a digital commons. However, there are different approaches for understanding the ontology of the commons. Antonios Broumas (2017a) offers a useful framework for understanding these differences when he identifies four different approaches: *resource-based*, *property-based*, *relational/institutional*, and *processual*. Ostrom's (1990) approach tends to position commons as *resources* or *resource systems* that are shared by a group of people, which make them susceptible to social dilemmas. In *property-based* approaches the collective property of the commons is differentiated from private and public property. *Institutional/relational* approaches attempt to account for a 'wider set of instituted social relationships between communities and resources' (Broumas, 2017a: 1509; see also Dardot and Laval, 2019). Finally, in a *processual* approach, 'commons are defined as fluid systems of social relationships and sets of social practices for governing the (re)production of, access to, and use of resources' (Broumas, 2017a: 1509). In the processual approach, commons are understood as a process or a state of becoming. This process has also been summarised by Linebaugh (2008) when he proposed the use of *commoning* as a verb, which will be discussed in greater detail in the following chapter. For the time being, however, it is worth noting my own understanding of the commons tends to fall more clearly within the *processual* or *dialectical* understanding of the commons. This approach is also nicely summarised by Broumas (2017a) when he explains the complex interaction that takes place between a producing subject and its interrelationship with an external objective environment:

> the interaction of subject and object takes the form of a subject/object, an entity that preserves certain elements of subject and object, eliminates others, and sublates the status of such an entity through the emergence of novel properties that did not exist in its generating entities (1510).

In building on this general discussion of how free software and the digital commons can be understood through different approaches, the following section will outline one of the primary threats to the commons, which is enclosure. I offer a clarification of why I have opted for a different term to describe the complex dynamics taking place between FLOSS communities and corporations.

1.4.1. Incorporation vs. Enclosure

Within certain approaches to understanding the commons – most notably the *property-based* approach – the commons are generally held in contradistinction to private property. In other words, once the commons become commodified or privatised, they cease to be commons and are in the service of capital. Even within more recent work on the revolutionary potential of the commons

and commoning activities, the commons are positioned as a potential alternative to capitalism (see Dardot and Laval, 2019). The process by which commons become transformed into private property is known as enclosure. Historically, the enclosure of common land in England took place in varying degrees between the 15th century and the 19th century.[8] Enclosure took various forms throughout this period, including voluntary enclosures, forced enclosure, parliamentary legislation, and others. Throughout this process, ownership of common land was transferred to private owners, who then claimed the right to restrict access to the land. This effectively ended the open field system, whereby commoners held traditional rights to use open fields for feeding livestock, farming, or harvesting from the land. While historians still debate the extent to which enclosure exacerbated class divisions and played an integral role in the development of capitalism in general, the process nonetheless affected the relationship between commoners, capitalists, and the commonly held resources that once provided a means of subsistence for commoners. Moreover, the state played a crucial role in facilitating enclosure through the Enclosure Acts, which were passed between the 18th and 19th centuries in England and Wales (see Polanyi, 2001).

The enclosure of common land was accomplished by literally erecting fences around previously open fields. Enclosure of knowledge commons, however, depends on restricting access or prohibiting certain uses of informational resources. James Boyle (2003) refers to the process of enclosing the knowledge commons as the Second Enclosure Movement, whereby intellectual property rights restrict access to those things which were once considered common property.

Similarly, Mark Andrejevic (2007) uses the term 'digital enclosure' to refer to the process by which two distinct classes are formed online: 'those who control privatised interactive spaces (virtual or otherwise), and those who submit to particular forms of monitoring to gain access to goods, services, and conveniences' (3). In other words, Internet users, as a class, have nothing to sell but their data, which serves as a form of value production for Internet Service Providers (ISPs), which represent a class that controls the means of digital production. In this sense, the ISPs can restrict access to their sites unless users agree to the Terms of Service (ToS) or End User Licensing Agreement (EULA). These non-negotiable contracts place restrictions on how users may interact with the site. The effect of these agreements is to enclose informational resources, which are controlled by ISPs. This type of value capture has also been critiqued in debates about digital labour (see Jarrett, 2016; Fuchs, 2015; Scholz, 2013), which will be discussed further in the following chapter.

In this book, I use the term 'incorporation' rather than 'enclosure'. The term 'enclosure' implies either a physical barrier or other restriction (i.e. intellectual property rights) placed upon the commons. In effect, the 'enclosure' of digital commons typically refers to the process of imposing higher degrees of excludability on the collective resource. However, as the case studies in this

book demonstrate, corporations have developed unique ways of transforming the products and processes of commons-based peer production into commercial offerings without placing restrictions on FLOSS communities' access to their common resources. This is qualitatively different from other forms of 'enclosure' discussed above. For this reason, I have opted for the term 'incorporation' because I think it more accurately describes what is happening when corporations get involved in FLOSS projects, and this will be made clear by the case studies provided in subsequent chapters. Incorporation is generally defined as the inclusion of something as part of the whole, but it also carries the specific legal definition of formally establishing an organisation as a corporation. In what follows, however, I discuss one more notable contribution for understanding the dynamics between FLOSS communities and corporations.

1.4.2. Commons-Based Peer Production

The work of Yochai Benkler (2006) is useful for understanding the broader social dynamics at work in communities of peer producers as well as how those communities intersect with existing institutions. One of the most notable contributions in this regard is his concept of *commons-based peer production* and its consequences for a broader set of social relationships. Benkler (2006) argues that commons-based peer production constitutes a new form of organisation that is 'radically decentralized, collaborative, and nonproprietary; based on sharing resources and outputs among widely distributed, loosely connected individuals who cooperate with each other without relying on either market signals or managerial commands' (60). Benkler positions social production in general and peer production specifically in contradistinction to market-based production, arguing that these forms of production constitute a form of non-market production. While these spheres are not mutually exclusive, Benkler argues that diverse forms of non-market production, like FLOSS, have the capability to influence market production.

Peer production can challenge market-based production in at least a couple of ways. First, peer production can develop products that will compete directly with those produced by commercial firms. In this case, the commercial firm has a few different options: compete, do nothing, or adopt and adapt. If the firm chooses to compete, it will be required to somehow create a better product than that offered by the nonmarket rival, although this may come at considerable cost to the firm. Alternatively, the firm can do nothing. In this case, the firm is basically relying on the belief that its products are superior to the non-market option and that the non-market option will not gain additional market share. This is a risky strategy for the commercial firm. If the non-market option does gain an increasing share of the market, the commercial firm, or at least its product that directly competes with the peer-produced option, runs the risk of becoming obsolete. The third option is to adapt to the changing forces in the

market by adopting some of the strategies of the non-market forces. This type of strategic reorientation to non-market forces can have the consequence of altering the basic structure of an organisation. As Benkler (2006) notes:

> As the companies that adopt this strategic reorientation become more integrated into the peer-production process itself, the boundary of the firm becomes more porous. Participation in the discussions and governance of open source development projects creates new ambiguity as to where, in relation to what is 'inside' and 'outside' of the firm boundary, the social process is (125).

Altering the firm's position in relation to peer production, which exists outside the firm, arguably offers a higher form of risk for the firm. The firm gives up a certain level of control over the production process. The traditional view of a firm's control over its informational resources or, more specifically, knowledge, is that knowledge can be viewed as an asset to be managed as an investment (Machlup, 1962). However, the peer production process in general is far more innovative and efficient than centralised production, including outside the realm of software production (Von Hippel, 2005).

Fritz Machlup (1962) was one of the first scholars to propose that knowledge could serve as an economic resource, and his work was one of the first to popularise the idea of the information society. However, knowledge and information are typically viewed from a supply-side perspective, especially in economics literature that treats these factors as investment costs for the firm. Arguing from an alternative perspective, Frischmann (2012) suggests that we can view knowledge, information, and cultural resources as a form of intellectual infrastructure. Doing so positions these resources as 'basic inputs into a wide variety of productive activities,' which 'often produce public and social goods that generate spillovers that benefit society as a whole' (Frischmann 2012, xii). Such an argument resonates nicely with the arguments in favour of promoting commons-based peer production for enabling greater innovation (Benkler, 2006; Von Hippel, 2005). By framing knowledge and information as an infrastructural component of social development, protecting the knowledge commons becomes crucially important to the survival of commons-based peer production.

The concept of the commons is useful for thinking about informational resources. Given the increasing interconnectivity between people across vast spatial boundaries with the ability to communicate and collaborate in online environments, maintaining a base of commonly held resources that can be used for peer-production remains a central concern for facilitating more open and democratic forms of communication. This is particularly the case because the commons are subjected to the threat of enclosure or incorporation, which can threaten a community's rights of access to the commons or the collective governance of the commons.

1.4.3. Summarising Different Approaches to the Commons

The previous sections introduced the commons and commons-based peer production. Those sections drew heavily from the work of two scholars: Elinor Ostrom and Yochai Benkler. However, these scholars take different approaches to their ontological understanding of the commons. Drawing from Broumas's (2017a) framework, I positioned Ostrom's work as a *resource-based* ontology of the commons. This is because Ostrom began her analysis with the collectively governed resource, then examined the ways that communities governed those resources. The value of Ostrom's scholarship, then, was to provide a framework for understanding how communities can manage common resources outside of market relations or state provision. Rather than offering a prescriptive argument for how all communities ought to govern common resources, Ostrom's framework accounts for the diverse and varied ways that communities establish adaptable institutions of governance for managing complex problems. As such, Ostrom's project builds a 'bottom-up' approach for understanding community governance as well as the community's relationship to common-pool resources.

The work of Yochai Benkler (2006) can also be understood within the emergence of the commons paradigm, although his approach differs from Ostrom. Benkler's ontological positioning of the commons falls more within the *relational/institutional* approach, as defined by Broumas (2017a). Such an approach abstracts from simply focusing on communities or resources, and instead focuses on the social relations and structures that exist between the two. In this regard, his work focuses on the broader implications of the digital commons for economics, politics, and culture. Ultimately, he explores the greater degrees of freedom, autonomy, and creativity that are made possible by digital technologies, including the ways in which digitally networked practices of production would alter the relationship between communities and capitalist firms. In this regard, Benkler's work is also more conducive to a critical or, in Broumas's terms, a *processual* or *dialectical*, understanding of the dynamics existing between FLOSS communities and corporations.

Broumas (2017b) also offers another framework for differentiating between social democratic and critical theories of the intellectual commons that is useful in this regard. Although his framework was used to discuss the intellectual commons, the framework may also be mapped onto the digital commons. According to Broumas, social democratic theories of the commons 'employ political economic methodologies to analyse the dynamics that unfold between the commons, the market and the state with the aim to propose reconfigurations of these relations which will best serve social welfare' (103). Such theorists argue that by making progressive changes to existing structures, we can bring about a more just and egalitarian society. As it concerns the digital commons, the goal is to build repositories and platforms for commons-based knowledge and peer-to-peer production that can, in turn, bring about greater degrees of personal freedom as well as democratic decision-making (Bauwens 2005; Benkler 2006).

Table 1.4: Social Democratic and Critical Theories of the Commons (Broumas, 2017b: 121).

	Social Democratic Theories	**Critical Theories**
Epistemology	Political Economy	Critical Political Economy
Agency	Social Individuals	Social Intellect
Structure	Productive Community	Community of Struggle
Internal Dynamics	Bottom-Up/Top-Down Emergence	n/a
External Dynamics	Co-Existence of Commons with Capital	Commons/Capital Antagonism and Sublation
Normative Criteria	Deontological [reformist]	Deontological [subversive]
Social Change	The Commons as Substitute for the Welfare State	The Commons as Alternative to Capitalism

In the framework visualised in Table 1.4 above, Broumas (2017b) examines some of the foundational characteristics of each approach, focusing on epistemology, agency, structure, internal/external dynamics, normative criteria, and social change. Of particular interest in Table 1.4 is the relationship between the external dynamics and social change sections. The section on external dynamics in the table represents a large portion of the subsequent chapters, in which I explore the relationship between capitalism and the commons. One of the pressing questions for FLOSS specifically but for the commons more generally is whether these movements are capable of constituting alternatives to capitalism. Indeed, some recent scholarship by Massimo De Angelis (2017) specifically attempts to frame the commons as an alternative value system that is emerging from within capitalism but also one that has the potential to usher in a post-capitalist future, and this will be discussed in greater detail in the following chapter.

My goal for the next chapter is to specifically outline the contours of a critical political economy of the digital commons. To begin the transition to that task, however, the final section of this introduction discusses some of the methodology used by critical political economists in general and in this study specifically.

1.5. A Note on Methodology

The following quote from Marx (1845) comes from a section of *The German Ideology* that discusses the essence of historical materialism:

> Empirical observation must in each separate instance bring out empirically, and without any mystification and speculation, the connection of the social and political structure with production. The social structure and the State are continually evolving out of the life-process of definite individuals, but of individuals, not as they may appear in their own or other people's imagination, but as they really are; i.e. as they operate, produce materially, and hence as they work under definite material limits, presuppositions and conditions independent of their will (Marx, 1998, 41).

The quote represents a methodological approach to inquiry that is guided by assumptions about how reality can be understood and described. The quote also nicely summarises the goals of researchers working within the critical political economy of communication – that is, to connect the definite processes of material production with broader social and political structures. Most often, the inquiries of critical political economists of communication are directed at large corporations that hold extensive market power and the ability to influence the production, distribution, exhibition of, or access to, communication resources. In the process of investigation, the aim of critical political economists is to empirically investigate the material operations of corporations and connect those operations to the broader social system. The connections made to the social system can be situated within national boundaries while accounting for the attendant institutions (religious, legal, cultural, etc.) that encourage or discourage certain types of behaviour, but can also be made across those boundaries (internationally, regionally, globally).

By making these connections, political economists search for the general tendencies of capitalism rather than seeking to establish absolute laws. This allows the inquiry to remain open to the possibility of contradictory factors, while also allowing for an account of diverse practices both within and across media industries. Indeed, the contradictory factors provide the illuminating moments for critical researchers, particularly because they provide opportunity for critique and resistance. To this end, critical political economists of communication have provided important critiques of corporations, especially the ways in which they operate in conjunction with the general tendencies of a broader capitalist system. As Meehan (1999) notes, 'critical scholars share an ethical obligation to produce knowledge that accurately describes the media and reveals the hidden dynamics whereby media corporations attempt to commercialise and control expression in service to advertisers and ultimately to capital' (162).

To search for these 'hidden dynamics', the current study employed a *critical interpretive* methodological approach. Maxwell (2003) describes this approach as used by Herbert I. Schiller, a pioneering scholar working within the critical political economy of communication tradition. When working from a critical perspective, one situates research findings within broader bodies of knowledge and looks for disjunctures or contradictions arising from within the field of

study. These contradictions or disjunctures can provide germane moments for research, from which previously accepted understandings can be challenged and refined. In this sense, CPEC scholars resist interpreting research findings according to their face value or as *prima facie* evidence. Rather, the research findings are brushed against the grain of alternative bodies of knowledge to situate the results within a broader set of relationships. Similarly, Mosco (2009) describes his epistemological stance as being *constitutive*. That is, critical political economy scholars resist causal, linear determinations as well as the assumption that units of analysis are fully formed wholes. Instead, critical political economists favour an epistemological position that is based on mutually constitutive processes, which act on one another throughout various stages of formation. In this sense, the approach is dialectical in that it considers both particular and more general phenomena as part of a totality of processes. These concerns are carried with the researcher throughout the research process, regardless of what type of evidence is being investigated or how it is being gathered.

To facilitate this type of investigation, critical political economists use a variety of methods. However, the selection of method is often driven by the amount of access that the researcher has to the subject being studied. When direct access to corporations is available, critical political economists rely on research methods such as interviewing, participant observation, ethnographic methods, and other methods that allow for direct observation of the life-processes of definite individuals as they operate or produce materially. In turn, these observations can be linked with the 'definite material limits, presuppositions and conditions independent of their will' (Marx 1998: 41). When we do not have direct access to corporations, critical political economists rely on documentary evidence of corporate operations and the material production taking place within the corporation. Most often, this data comes from documents that are produced by and about the corporation. To that end, the following section discusses the specific methods used in this study.

FLOSS projects depend on extensive and accurate documentation to make the development of projects run effectively and efficiently, and these documents are made publicly available so that other developers can work on the project. The source code is one form of documentation, which enables users to understand how a project works, but many FLOSS projects also contain credits files, licensing disclosures, README files, and other documents that provide essential information to users. This information, as well as the information found on publicly available discussion lists, was combined with my experiences using Linux and attending a variety of different events and meetings focused on FLOSS, including local LUG meetings and the Open Source Convention (OSCON). The aim of these documentary and first-hand experiences was to understand the dynamics between the corporations and the community of software developers, specifically how the latter negotiate their relationship with those corporations.

The advantage of researching FLOSS communities is that nearly all FLOSS projects have unique forums, bulletin boards, or wikis dedicated to providing documentation and facilitating communication about the project. These sources typically contain repositories of the project itself, but they also offer community discussion and historical data about the project's development. This, in turn, can provide documentary evidence of ongoing and past events in a way that is open to the public. For example, the Fedora Project, which is discussed in Chapter 4, features a wiki that contains extensive documentation about the project, including news, events, recent changes, user guides, and links to various sub-projects associated with the main Fedora Project. Similar sources can be found for all the FLOSS projects discussed in this study.

This introductory chapter identified the central concerns of this project by highlighting the seemingly contradictory goals of free and open source software communities and capitalist firms. Furthermore, I situated FLOSS historically by discussing some of the foundational moments in both the development of software as well as the rise of free software specifically. This discussion also included a consideration of FLOSS's cultural significance. Finally, I outlined the specific methodological approach used in the study. Now that the broad outlines and contours of the study have been established, the following chapter discusses more specifically the theoretical frameworks used to understand the complex relationships between FLOSS communities, their commons-based peer production, and capitalist accumulation.

Notes

[1] A photo of the moth that was removed from the machine is available from the Naval Historical Center at https://www.history.navy.mil/our-collections/photography/numerical-list-of-images/nhhc-series/nh-series/NH-96000/NH-96566-KN.html

[2] There is a longer history of computing research at Harvard that traces back to the 1930s, including Vannevar Bush's differential analyzer and Claude Shannon's electronic Boolean algebra. Shannon is also well known within the field of communication studies for his landmark, *A Mathematical Theory of Communication*, which was published in 1948. However, research on computing at Harvard became specifically focused on artificial intelligence in the late 1950s.

[3] The GNU Manifesto is available at http://www.gnu.org/gnu/manifesto.html (last accessed 4 January 2019).

[4] The use of the combined term 'FLOSS' is mostly pragmatic, as I am interested in exploring dynamics between the communities producing a free and/or open source software project and those corporations that sponsor or otherwise use that software. I'm interested in these dynamics regardless of whether those communities identify as free software communities, open

source communities, or some combination thereof. In certain places in the book, I specify one or the other when a distinction will be important. Otherwise, I use the FLOSS acronym for more general discussion.

[5] The supercomputer at the Oak Ridge National Laboratory is known as Summit and was built by IBM. When this computer took over the top position as the world's fastest supercomputer in June 2018, it marked the first time that a computer in the United States held that position since November 2012. In the interim, the top position was held by computers in China.

[6] The text of the GNU General Public License (GPL) can be found at http://www.gnu.org/copyleft/gpl.html (last accessed 4 January 2019).

[7] The Creative Commons Licenses can be found at http://creativecommons.org/licenses/ (last accessed 4 January 2019).

[8] A detailed account of the English enclosures is not provided here, but those interested in a more detailed treatment should see Neeson, 1993; Thompson, 1966; and Marx, 1906, especially Chapter 27: 'Expropriation of the Agricultural Population from the Land,' which is freely available at http://www.marxists.org/archive/marx/works/1867-c1/ch27.htm.