Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–147

DOI: xxxxxx

now
the essence of knowledge

Open Source Software: A Survey from 10,000 Feet

Stephanos Androutsellis-Theotokis[1], Diomidis Spinellis[2], Maria Kechagia[3] and Georgios Gousios[4]

[1] *Patision 76, Athens, GR-104 34, Greece, stheotok@aueb.gr*
[2] *Patision 76, Athens, GR-104 34, Greece, dds@aueb.gr*
[3] *Patision 76, Athens, GR-104 34, Greece, mkehagia@dmst.aueb.gr*
[4] *Patision 76, Athens, GR-104 34, Greece, gousiosg@aueb.gr*

Abstract

Open source software (OSS), the origins of which can be traced back to the 1950s, is software distributed with a license that allows access to its source code, free redistribution, the creation of derived works, and unrestricted use. OSS applications cover most areas of consumer and business software and their study touches many disciplines, including computer science, information systems, economics, psychology, and law. Behind a successful OSS project lies a community of actors, ranging from core developers to passive users, held together by a flexible governance structure and membership, leadership and contribution policies that align their interests. The motivation behind individuals participating in OSS projects can be, among others, social, ideological, hedonistic, or signaling, while companies gain from their access to high-quality, innovative projects and an increase in their reputation and visibility. Nowadays many business models rely on OSS as a product through the provision of associated services, or in coexistence with proprietary software, hardware, services, or licensing. The numerous OSS licenses mainly differ

on how they treat derived software: some contain provisions that maintain its availability in open source form while others allow more flexibility. Through its widespread adoption, OSS is affecting the software industry, science, engineering, research, teaching, the developing countries, and the society at large through its ability to democratize technology and innovation.

Contents

1

Introduction

Open Source Software (OSS) is software distributed with a license allowing access to its source code, free redistribution, the creation of derived works, and unrestricted use. The history of open source software can be traced back to the 1950s SHARE user group, the academic distribution of Unix, and the GNU project.

Open source applications cover most areas of consumer and business software. Prominent application areas include systems infrastructures like operating systems and databases, software development, personal productivity, desktop, entertainment, graphics, publishing, education, scientific, engineering, content management, and business software.

The organization of open source development projects often differs from proprietary ones in terms of their organizational structure, membership, leadership, contribution policies and quality control. Lean, distributed, and often informal operations make it easy to start or participate in an OSS project, but also isolate projects from market pressures allowing many to languish or fizzle.

Behind a successful OSS project lies its community. Its actors range from core developers to passive users. Although a community's governance structure is typically flexible, many processes and mechanisms align the interests

of the community's members. Initiative, teamwork, communication, and cooperation are generally more important than in business software development.

The key defining element of open source software is its license, which must satisfy a list of important requirements. There are numerous open source licenses, and they mainly differ in how they treat derived software: some contain provisions that maintain its availability in open source form, while others allow more flexibility. Selecting an appropriate license for a new open source project is important, as is studying an open source project's license before incorporating it into a proprietary system.

Nowadays many business models rely on open source software, either as a product or through the provision of associated services. Revenue can be obtained from the complementarity of a proprietary product with an open source one, support and training, subscriptions, and advertising. The strategic dimensions behind a move to open source software include not only opportunities related to marketing and innovation, but also risks associated with a loss of profits and the lowering of competition barriers. On a tactical level an open source-based business model can lower development costs, enable end-user customization, but will also demand new organizational structures, a higher short-term investment, and the continuous nurturing of an open source ecosystem.

Open source software can be reused as a (low cost) product, as an adaptable component, or as code and other elements that are morphed into another system. Increasingly, open source systems form complete stacks used as infrastructure for other applications. In specific categories, such as web applications, the adoption level of open source software is near or even higher than that of proprietary offerings. The impacts and effects from open source adoption affect an organization's bottom line, its management, the software's quality, and the software development process.

An often asked question regards the motivation behind individual and organizational participation in open source projects. The incentives for individuals can be social, political, ideological, hedonistic, as well as the allure of a flexible, stress-free, and bleeding edge technological environment. Companies seem to gain from their participation as well, through privileged access to a high-quality product and its development process, as well as exposure to user-driven innovation, higher reputation and visibility, human capital im-

provement, and improved employee morale.

The emergence of open source software is fueling the economy as a whole through its widespread adoption as a cheap alternative to pricey proprietary products and as a driver behind many successful e-business ventures. Open source is also directly affecting specific sectors: the software development industry through competition and new business opportunities; hardware development through lower cost and barriers of entry, consumer-led innovation and policy enforcement difficulties; academia through valuable opportunities for research and student involvement in real-world applications, as well as the availability of software tools and the provision of pioneering new courses.

The future of OSS appears to be as exciting as its past. It can lead to new design, production, marketing, and business models, as well as ways to develop large complex software systems in an organic manner. Challenges lie ahead, and problems still need to be overcome, so the potential for future research on OSS is large. For instance, the comparison between open source and proprietary products and processes is still an area lacking solid empirical evidence. More important however is the ability of open source development models to democratize technology and innovation.

1.1 Open Source Software and Other Types of Software Distribution

Up to the late 1980s most packaged software was almost exclusively sold and distributed as a complete and finished product (a so called "precompiled binary"), which was installed on a user's computer and then ran [24].

With the evolution of software development, computers and the internet, new models and types of software distribution appeared. These differed in aspects such as the degree of openness of the software product (i.e. how much information about the inner workings of it is exposed to the user), the possibility for the end user to modify it or use parts of it in other, derivative software works, and the cost and licensing model.

According to the classification put forward by the Free Software Foundation (FSF) [76] and elaborated by Perens in reference [181], the main types of packaged software distributions used are the following:

Proprietary or commercial software is typically distributed in binary form only, with the source code closed, i.e. not available to the public. Payment is required and the terms of use are very restrictive, not allowing modification or redistribution.

Public domain software lies at the other end of the spectrum. The authors of this type of software give up all copyright, the source code is freely available for modification or redistribution, and no fees are required. In fact it is even allowed to obtain public domain software and re-distribute it under other, non-open licensing schemes, or even remove the author's name and treat it as one's own work.

Freeware and shareware products do not require upfront payment and can generally also be duplicated, as is the case with public domain software, however modifications are typically not allowed as the source code is not distributed with the product.

The difference between Freeware and Shareware is that with Shareware only limited usage of the product is allowed without payment, either for a fixed evaluation period, or with reduced functionality. Shareware is generally regarded as more of a marketing concept than a licensing option.

Open source software is the distribution and licensing approach that is the topic of this survey. The main characteristics of OSS are outlined within the Open Source Definition[1] and can be summarized as follows.

- Free distribution: No licensing fees are charged for this type of software.
- Source code availability: The source code is distributed together with the product.
- Modifications and derivative works: The users of the software can modify the source code to create derivate software products, or reuse (parts of) the source code in other products. However this may be subject to specific restrictions dictated by the OSS license used.

[1] `http://www.opensource.org/docs/osd`. Note: All internet URLs in this survey, including in the references section, were last accessed in March '10.

- No discrimination: Either against persons, groups or fields of endeavour.
- Licensing: OSS products are copyrighted, and distributed with a particular license that outlines the terms of their use. There are various OSS licensing options, which differ in their degree of permissiveness and other aspects.

One of the most important aspects of an OSS license is whether any derivative work that is based on the source code of this particular software product can be distributed under different licensing schemes (either OSS or proprietary), or whether it is only allowed to be distributed under the same license as the original product.

The OSS licenses enforcing the latter condition are known as restrictive, or "copyleft" licenses, and their goal is to ensure that the source code will remain available to the public. The different types of OSS licenses are discussed in more detail in Chapter 6 and summarized in Table 6.1.

OSS development is based on the formation of large, open and distributed communities of developers who are guided by a common belief in the freedom of software and information, and who follow collaborative practices such as sharing information, helping others, and studying and peer-reviewing each other's work. Such developers are motivated by their own interest in the project and the urge to learn from it, and they are rewarded by the acknowledgement of their contributions, the resulting reputation they gain, and the success of the project itself.

1.2 Research, Related Disciplines, and Publications

The study of OSS is inherently multidisciplinary, encompassing various research and scientific disciplines [78]. In the following list, as well as in Table 1.1, we provide indicative examples of research efforts spanning two or more research fields, including OSS:

- Computer and information system sciences study the technical aspects of OSS development [201, 163, 200].
- Management and organizational sciences deal with the management, organizational and governance aspects of OSS project [164, 127].

- Social science addresses areas related to the communities formed around OSS efforts, their motivation, behaviour, and evolution [229, 48, 176].
- Psychology delves into issues relevant to the individual participants in OSS projects, what drives and motivates them, and how they are rewarded [254, 138, 16, 69].
- Economics studies the business models that OSS projects are based on, the involvement of corporations in OSS efforts, as well as the ecosystems and collaborations built around them [146, 17, 110].
- Law focuses on the various legal, licensing and copyright issues around OSS distribution [149, 148, 203].
- A multitude of other scientific fields (such as medicine, biology, and engineering) benefit by using OSS products, and by applying OSS ideas and methods in their domain [10, 28].

Interest in OSS spans many professional areas and domains, including software development, business, research and government. In Chapter 10 we discuss in more detail the impact of OSS in all these domains of our society and global economy. We feel that this survey provides not only an overview of the field, but also considerable practical information for those wishing to get involved in OSS as developers or project members, by adopting OSS in their products, or by gaining insight from the OSS practices, ideas and experience.

Within this survey there are numerous references to works from many different scientific domains and disciplines. We highlight some in Table 1.1, which itemizes some of the most informative relevant works, grouped by subject. We separate empirical studies, surveys and overviews, and articles focusing on specific subjects. We also recommend:

- the collected works in [54, 67],
- the 2004 theme issue of *IEEE Software* [216], the 2004 issue of *Research Policy* [239], the 2006 issue of *Management Science* [240], the 2010 special issue of the *Journal of the Association for Information Systems* [42], and
- the proceedings of the International Conference on OSS, and the FLOSS ICSE Workshop.

	Empirical studies	Surveys / Overviews	Specific topics
Project communities	[53][127] [229] [46][109][128] [133][170]		[21][70][205]
Motivations	[22][75][101] [102][137][138] [238]		[21][190][257] [11]
Success factors			[21][37]
Software development	[163][218] [200][179]	[66][67][201]	
Innovation and future	[75][238][71]		[235][240][232]
Adoption and reuse	[98]	[222]	[216]
Business models	[204]	[248]	[21][205][72]
Licenses			[140][146][148]
Generic	[56]	[51][54][83] [94][185][245] [129]	

Table 1.1 A collection of informative publications on different aspects of Open Source Software.

1.3 Organization of this Paper

In this survey we aim to cover most aspects of OSS, including technical, social, organizational, economic, and legal, as well as provide an outlook to the future of OSS by identifying current shortcomings and research directions.

In particular, Chapter 2 overviews the history and evolution of OSS, from the first free software development efforts to the latest OSS business and financial models.

Chapter 3 deals with the organization of OSS efforts into projects, their comparison with proprietary software development efforts, and particular characteristics and potential indicators for project success.

In Chapter 4 we examine in more detail the characteristics of the communities that are formed around OSS projects, the different actors and par-

ticipants, the leadership and governance mechanisms that are employed, and their evolution.

Chapter 5 focuses on the more technical aspects of OSS, and in particular the software development practices and processes. It presents the main characteristics of OSS software development and how it differs from other domains.

Chapter 6 on the other hand analyses the legal and licensing perspective, which is crucial as it characterises the permissiveness and often the impact of each OSS effort. We briefly outline the main OSS movements, the different licensing options and we provide some guidance into selecting the most appropriate licensing scheme depending on an OSS project's characteristics.

In Chapter 7 we focus on the economic and financial nature of OSS projects, and what business models can be adopted to extract business value from an OSS effort. We discuss the business ecosystems that are formed around successful OSS efforts, and the various roles that companies and organizations can play within them.

Chapter 8 then focuses on the important issue of adoption and reuse of OSS software into other products and domains. It examines the criteria for an OSS product to be a good candidate for reuse, the process of adopting and reusing OSS code, and benefits but also the potential risks and concerns that accompany this practice.

In Chapter 9 we discuss the motivational aspects for engaging in an OSS effort, both for individuals and for businesses and organizations.

We conclude in Chapter 10 with an overview of the impact that the OSS process and ideology has had on the software business and our society, closing with a discussion of the current research directions, and where they may lead the future of OSS.

2

History and Evolution

The concept of open or free software is old. Its roots lie in the 1960s and 1970s, when early computers were used in universities for research, and software programs were freely circulated among scientists. Building upon each other's software and giving back the modifications was considered a normal communal practice and became a feature of what was known as "hacker culture" hacker being a term used in communities of programmers to characterize skilled and passionate programmers (although later it also acquired a negative significance by the public). Raymond [186] and Bretthauer [24] provide excellent historical prespectives of the early days of the hacking movement.

In the following paragraphs we briefly trace the evolution of OSS through the main systems, events, applications and movements that shaped it. Figure 2.1 outlines this information, while figure 2.2 offers a more diagramatic view of the transformations of the OSS landscape as one event, movement or initiative led to another, reaching the current state of the OSS domain.

2.1 The Early Years

The beginning of OSS is associated with the appearance of the first large computer systems. As programmers invented programming tools, techniques and

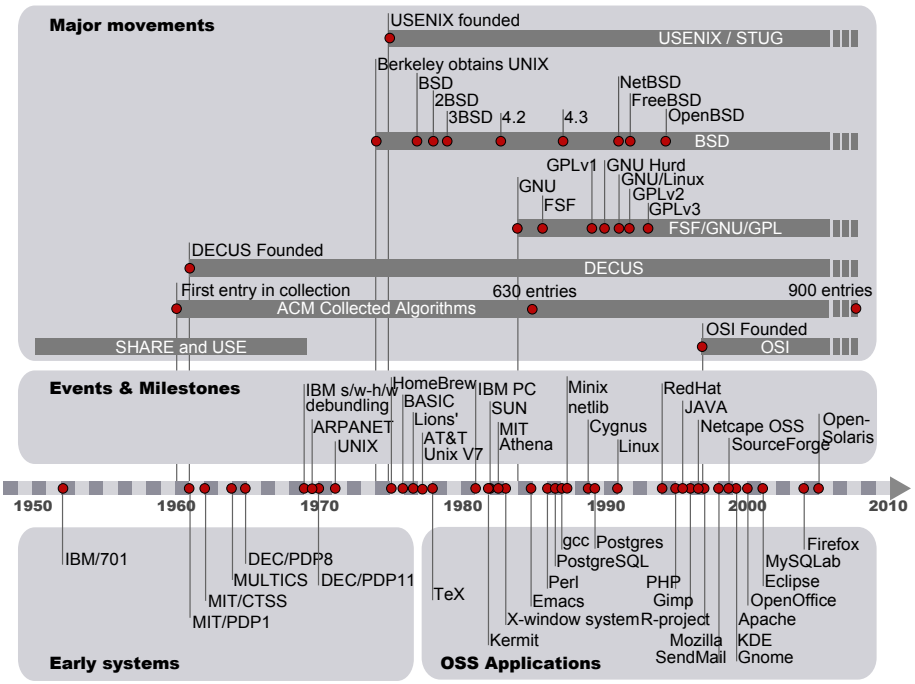


Fig. 2.1 The history and evolution of OSS. Main systems, applications, events and movements.

applications to render them productive and useful, they freely shared them among themselves, often contributing to each other's efforts.

Although the first commercial computer put on the market that required programming was IBM's 701 in 1952 [245], it was MIT's acquiring of the first PDP-1 in 1961 that gave rise to the hacker culture that is still recognizable today. This was the heart of MIT's Artificial Intelligence Laboratory. Later came DEC's PDP-8 in 1965, followed by the innovative PDP-11 in 1970, which was much more affordable and had good enough performance both for universities and corporate research [198].

These early computer systems established the need for multiuser support, and the solution proposed by MIT in the early 1960s was the Compatible Time-Sharing System (CTSS). This was followed by a joint endeavour between MIT AT&T's Bell Labs and General Electric in 1964 to build the Multics system, which was expected to be released with its source code openly

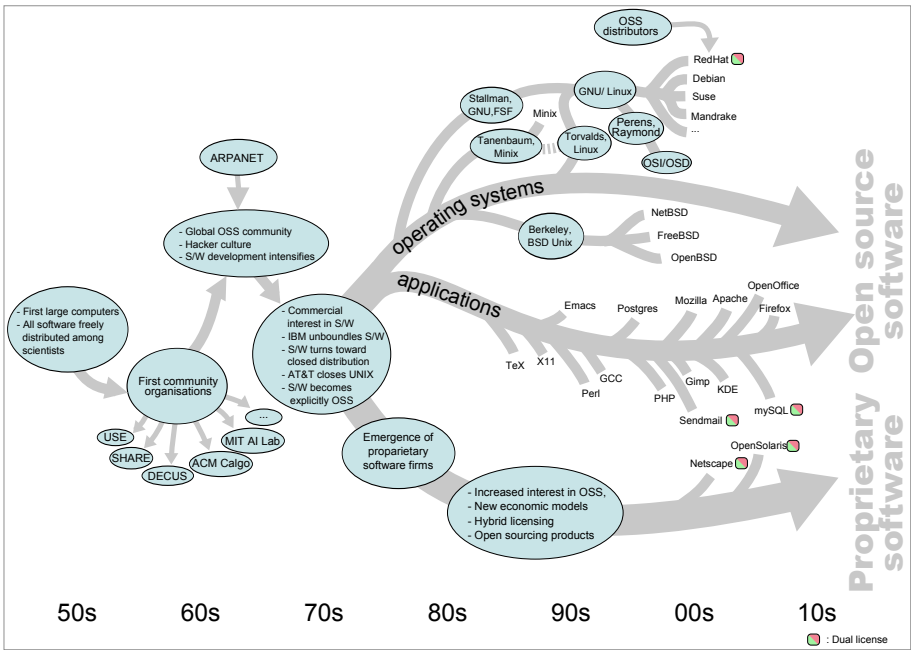


Fig. 2.2 A diagramatic view of the evolution and transformation of the OSS domain.

available to the public. The project ran into problems, but this led two Bell Labs researchers, Ken Thompson and Dennis Ritchie, to work on a new type of operating system (which later became Unix), whose brief history and basic operating principles are described in a seminal paper [188].

From the very beginning the communities of users that revolved around the development and use of open software felt the need to organize themselves, often in ways resembling current OSS project communities.

Some of the first such instances were the SHARE user group of IBM and the Univac Scientific Exchange (USE) in the 1950s and 1960s, which were ran by volunteers to facilitate the distribution of software [64]. Similarly DECUS, founded in 1961 as a society for users of Digital Equipment Corporation's computers, promoted the open exchange of user-developed software; initially through listings and later through magnetic media [55].

The ACM Collected Algorithms,[1] initiated in 1960, archives software associated with papers published in the *Transactions on Mathematical Software*, as well as other ACM journals. It is a family of publications including a collection of around 600 software algorithms that were freely distributed for non-commercial use.[2] It is one of the first organized instances of public distribution of software source code and algorithms.

Sharing software and building on each other's work was still the only way known to the software community.

2.2 Unbundling of Software from Hardware

Up to the 1970s software was commonly distributed for free by hardware manufacturers, who were "bundling" it with their hardware as one product and including the cost of the software with the hardware. However, as software evolved and became more complex, so did the cost of developing it. An emerging software production industry was competing with this trend, seeking to sell their software independently.

A series of antitrust suits filed against IBM in 1969 resulted in a change in IBM's pricing policies and the offering of computer programs for a charge [195], thus paving the way for proprietary software development. Software then started being sold, often under restrictive licenses.

Six years later Bill Gates, having created a BASIC interpreter for the Altair 8800 microcomputer, found himself fighting to establish a presence on what would later become the home and personal computer software market. In an open letter to hobbyists [82], Gates argued that the notion of sharing software without paying for it was hurting software developers, and that royalty payments were critical for the development of high-quality software.

Both events, by restricting the availability of software, sowed the seeds of the OSS movement.

2.3 The Early Years of Unix

Ritchie and Thompson published the key ideas underlying the Unix operating system in 1974 [188]. Coincidentally, a 1956 "consent degree' prohibited

[1] http://calgo.acm.org
[2] http://www.acm.org/publications/policies/softwarecrnotice

AT&T, the legal entity behind Unix, from engaging in any business other than the furnishing of common carrier communications services. As a result, Unix was supplied royalty-free and without any formal support [198, pp. 56–60]. The combination of Unix and the C programming language developed by Ritchie proved very popular, and spread very quickly to a large number of university and research computer labs (the "Hackerdom", as dubbed by Raymond [186]).

With the source code in their possession, users including programmers, students and researchers were able to use Unix as a tool for learning, enhance it to cover their particular needs, and extend it to support their specific hardware. With no formal support from AT&T users were forced to collaborate and share ideas, information, programs, and bug fixes [198, p. 65].

A related movement sprang out of Kernighan and Plauger's work to popularize Unix's programming tools by publishing a book with readable and concise implementations of some key utilities [122]. The tools' source code was made available in executable form by the book's publisher [122, contents page]. Programmers in universities and computer vendors collaborated to enhance those tools and port them to various architectures, and in 1978 they formally grouped together as the Software Tools User Group [198, pp. 78–90].

In 1976 John Lions, an Australian computer scientist, published the complete source code of the 6th Edition Unix kernel together with a commentary of it [150]. It is considered an excellent description of the high-quality Unix kernel code, and for years it was the only public documentation available.

Although the first versions of Unix were freely distributed, as AT&T realised its value as a commercial product it decided to distribute Version 7 in 1979 with a license that restricted access to the system's source code [198, p. 151]. A side-effect of this was that Unix could no longer be used in university courses for teaching (and similarly the Lions book was no longer allowed to circulate). This was the main reason Andrew Tanenbaum later decided to develop the Minix operating system, as discussed in Section 2.5.

2.4 Workstations, Networking and the Hacker Culture

The late 1970s and 1980s saw the flourishing of two key drivers of OSS development, namely computer networking [184] and personal workstations (see

Ceruzzi's account of the influence of both technologies on hacker culture development in reference [30]). Both allowed the exchange and distribution of information as well as software between programmers at a new, global scale, boosting worldwide collaboration and productivity. Programmers and hackers from many parts of the United States and the world became connected, forming a networked group with its own culture, discussions, norms of behaviour, slang, and eventually beliefs and ethics .

Two notable OSS distribution efforts that were made possible by networking advances are

- the Usenet newsgroups *net.sources* (1982)[3] and *mod.sources* (moderated—1984),[4] which were later renamed into the comp.sources hierarchy, and
- the email-based *netlib* repository of software serving the numerical and scientific computing communities [57].

2.5 Notable Events in OSS History

The increasing trend toward proprietary software further fuelled the emerging OSS movement, whose advocates felt the need for free, open source operating systems and applications to be available to the public, leading to several core development efforts.

The Berkeley Software Distribution The University of California at Berkeley and Bell Labs collaborated to help Unix flourish between 1974 and 1977. This gave rise to the Berkeley Software Distribution, or BSD, a version of Unix with improved features, tools and utilities [144]. The BSD was shared with many research centers worldwide with the provision that they first obtain a source license from AT&T, thus encouraging them to view and contribute to the source code.

A second, more advanced version of the BSD was ready in 1978, while the use of an advanced version of Unix on the new 32-bit VAX machine at Berkeley led to the 3BSD distribution in 1979 [158].

It was realised at that point that as the research community continuously

[3] http://groups.google.com/group/net.sources/msg/d2bbe4e01cfd64c6
[4] http://groups.google.com/group/mod.sources/msg/39c786363ae144c9

modified the Unix system, an organization was required to manage and coordinate the new releases. This role was undertaken by Berkeley, as a result of its involvement thus far [158].

DARPA also decided to unify its activities at the operating system level, and chose Unix as the standard to use. The new 4BSD distribution thus came up in 1980 with a per-institution licensing arrangement, followed by a series of major releases, such as 4.3BSD the first release to include a full TCP/IP stack, as well as the more recent NetBSD, FreeBSD and OpenBSD.

During the 1980s, Berkeley and AT&T released their respective new Unix versions, which gradually became harder to tell apart.

The GNU Project, the Free Software Foundation and the GPL When the computer systems used at MIT's AI lab were replaced with new hardware that ran a proprietary operating system, the OSS programmer community that had formed around it gradually collapsed. Richard Stallman, a member of the lab, started looking for an alternative that could make the OSS community possible again, and this led him to the concept of free software, and in particular to the decision to create a new, Unix-compatible operating system that he called GNU (Stalman's article collection [83] contains many details on the early history of GNU). In 1985 he released the GNU Emacs editor, which he distributed for free over the network, or packaged on tape for $150. The GNU system included code from other projects that were also free software, such as the GNU Compiler Suite (GCC).

To support the GNU effort, Stallman initiated the FSF in 1985, to promote "the freedom to share and change software". He famously explained that "when I speak of free software, I'm referring to freedom, not price. So think of free speech, not free beer" [83, Chapter 1]. Most of the income of the FSF came from the sales of copies of free software and other related services, and secondarily from donations. Emacs and other free software, as well as manuals, were sold on tape, and later on other media. Part of this income was used to hire developers to work on basic GNU projects, such as the shell and the C library [83, Chapter 18].

One of Stallman's and the FSF's concerns was that their work could be taken and used in proprietary packages. To protect from this, the concept of copyleft was developed as a mid-way in between public and proprietary

software. This concept was the basis for the GNU General Public License, released in 1989 [24] (see Section 6.3.1).

The combination of GNU with the Linux kernel developed by Linus Torvalds (see below) in 1991, led to a complete GNU operating system [83, pp28].

Minix In 1987 Andrew Tanenbaum of the Vrije University of Amsterdam released Minix, an open source operating system based on a microkernel architecture which he created mostly for educational purposes [224]. Its source code was made available to universities for study and research. The design of the Linux operating system is considered to have been influenced by the Minix design principles, although Linux is not a microkernel per se.

First OSS applications Of the OSS applications that were developed around that time, some are still in widespread use today, notably the X-Window System, and TEX.

The X-Window System, developed at MIT but including contributions from numerous other sites, is the longest living foundation for developing graphical user interfaces. It is argued that its success was in great part due to the developers' willingness to open its source code and distribute it for free over the network with a very permissive license [186, Chapter 4].

The TEX [125] typesetting system was written by Donald Knuth in 1978, and rewritten from scratch and published in 1982. It is considered as one of the most sophisticated typesetting systems. The TEX Users Group (TUG) [5] was founded in 1980 as an organization for users of TEX and people interested in typography and font design.

2.6 OSS Meets Proprietary Software

The OSS and proprietary software worlds and cultures often met, affected each other's course, and sometimes clashed. IBM's un-bundling of software from hardware and AT&T's closing of the Unix source were two such examples.

In 1975 the Homebrew Computer Club was formed by computer hobbyists and enthusiasts. The rule of the club was that anyone could take a copy

[5] http://www.tug.org/

of software out, as long as they brought back two copies in the next meeting [115].

At that time, Bill Gates together with partner Paul Allen had written a version of the BASIC computer language that became very popular and was being wildly copied from one user to the other. Practices like this led Gates to write an "open letter to hobbyists", in which he addressed the issue of intellectual property rights and innovation.

This was one of the first cases of confrontation between two different cultures, which carried on for decades and is still ongoing.

Several years later, OSS started influencing proprietary systems. Various software vendors added the X-Window System to their proprietary offerings, and released the end product under a non-disclosure agreement. Paradoxically, soon most of the users of X-Windows were not running the free MIT version, but the proprietary versions that came from these software vendors [83]. OSS however was already establishing itself at an increasing pace.

2.7 OSS Becomes Mainstream

The Emergence of OSS Vendors As OSS became a mainstream phenomenon new companies were formed to benefit from this trend, and many of the existing proprietary software firms started looking toward the OSS movement, adopting it and often learning from it.

OSS software vendors and distributors appeared, OSS licensing was adapted to allow for hybrid solutions based on new financial and business models, new categories of services revolving around OSS were offered, and proprietary firms started opening (at least partly) their products' source code.

Founded in 1989, Cygnus Solutions was the biggest vendor of OSS at the time. It offered technical support services, with the GNUPro Developers Kit being one of its primary products [225].

At the same time, a multitide of OSS software infrastructures and applications appeared and gained widespread reputation.

Linux and OSS Distributors Linux is a free Unix-like kernel developed under the lead of Linus Torvalds in 1991. One difference between the Linux project and other concurrent efforts was that since its inception, it was based on a large community of developers whose work was coordinated only

through the internet. Frequent, almost weekly releases ensured that plenty of feedback was received within days from hundreds of users, and this was the main quality control mechanism for selecting which code changes to keep and which to reject [186, 227].

By 1993 Linux was stable enough to compete with many commercial Unix releases, and hosted more software applications. It was distributed commercially on CD-ROM with enormous success, and it was one of the focal points of hacking activity on the internet.

The Linux kernel is often bundled with other software, such as the shell and associated utility programs developed under the GNU effort, a graphical environment based on the X-Window System with the KDE or GNOME desktops running on top, as well as various application and server programs, such as the Firefox browser and the Apache web server respectively.

By 1994 [6] various Linux distributions were available at small cost, including SuSE, Debian, and RedHat. Notably RedHat Software bundled together hundreds of OSS software packages licensed into a so-called Linux distribution for retail sale [255].

Licensing Issues and The Open Source Initiative In July 1997, Bruce Perens, the leader of the Debian GNU/Linux Distribution project, addressed the problem of many different licenses for software that claimed to be "free" by proposing the Debian Social Contract and the Debian Free Software Guidelines (described by their author in reference [181]). Based on these, together with Eric Raymond who had studied and published work on the free software phenomenon and culture, they formed the Open Source Initiative (OSI).[7] Their goal was to establish a practical approach to software licensing and to promote commercial use of OSS [207].

The OSI developed the Open Source Definition (OSD) set of guidelines for OSS licenses that guarantees several freedoms for software users. The term Open Source Software, instead of Free Software, was proposed in order to assuage the reservations of business users. The term was met with resistance by the Free Software Foundation [83, Chapter 14], on grounds of not appropriately protecting the user's freedom.

---

[6] A comprehensive history of Linux distributions is provided by Wikipedia:`http://en.wikipedia.org/wiki/Linux_distribution`

[7] `http://www.opensource.org/`

The Open-Sourcing of Mozilla In 1998, Netscape announced that it would release the source code of the current version of its Navigator web browser under the name Mozilla. A licensing team wrote the software license and Mozilla.org was formed to coordinate the entire project. A group of Open Source community leaders (including Linus Torvalds and Eric Raymond) met with the Netscape legal team to discuss the existing licenses. The result was to craft a new license, the Netscape Public License (or NPL) in an attempt to protect OSS developers and at the same time promote development by commercial enterprises [100].

The first draft was published for comment by the public on the internet, and the feedback received led to the design of a new version of the NPL, as well as the similar Mozilla Public License (MPL).

Mozilla.org set up an organization, accumulated funds and resources and started functioning.

The Mozilla Firefox browser came about in 2004, multiply licensed under the GPL, LGPL (a version of GPL, see Section 6.3.2) and the MPL at the developer's choice [51].

Sendmail and Dual Licensing Sendmail is a major, successful open source application (it carries most of the world's email traffic) that was licensed under a dual license, i.e. both open and commercial.

The project started in 1981 at Berkeley by Eric Allman (author of the ARPANET delivermail application that was part of the BSD) as an open development project for a Unix-based mail transfer system. The open source version is licensed under an OSI-approved BSD-like license [251]. However the company Sendmail Inc. was eventually established in 1998 by Allman to develop and sell an enhanced, commercial version of the software that includes, among other features, a GUI for facilitating installation and configuration.

So two versions of fundamentally the same software were distributed under an open source and a proprietary license. This example was later followed by other software vendors, such as Trolltech and MySQL AB that commercialized their products, the Qt widget library and the MySQL database respectively, under a dual licensing scheme, enabling them to be very successful in both the commercial and the OSS application ecosystem. (An account of the reasons that may lead dual-licensed software to success is provided in

reference [230].)

2.8 Mainstream OSS Applications

A number of operating systems and a multitude of OSS applications emerged over the last three decades, including systems applications and infrastructure software, desktop applications, software for publishing, graphics and entertainment, scientific and business applications, and of course a whole range of tools for software development including compilers, interpreters, editors, IDEs, and version control systems. In terms of user base many OSS systems occupy the first or second place in their respective category.

Some of the most notable and successful such applications are outlined in the Appendix of this article.

3

Projects

This chapter introduces what is commonly referred to as an open source project. This loose term encompasses a collective effort whose goal is the production and support of software products. An open source project generally consists of the following:

- A community of developers, users and other actors, based upon an organizational and governance structure, driven by specific goals, and unified by shared values and cultural characteristics.
- Various software production processes for the management, development, release and maintenance of the software products.
- Technologies, infrastructures, platforms and tools utilized in the software production.
- A licensing model that abides by the general OSS guidelines, while at the same time ensuring the viability and success of the project.

We discuss many of the above aspects of OSS projects (as well as others) in other chapters of this survey. In this chapter we focus on open source as a development methodology, and on cross-cutting characteristics and concerns that distinguish open source projects from proprietary endeavors and affect their success.

	OSS **projects**	**Proprietary projects**
Membership	Substantial voluntary participation	Paid staff
	Large number	Limited number
	Open to all, voluntary	Closed (company)
	Virtual boundaries, fluid	Bound by contracts
	Independent community	Belongs to company
Decision making	Core developers or project leader	Company / organization
	No formal structure, voting consensus	Strict and rigid structure
	Meritocratically distributed	Centralized control
Motivation	Intrinsic and extrinsic, social / political	Monetary rewards / career
Actors	Usually skilled and motivated	Company employees
	Devote part of their time to project	Main occupation, full time
	Developers and users	No users employed

Table 3.1 Differences usually encountered between OSS and proprietary software projects. Part a: Communities

3.1 Open Source vs Proprietary Software Projects

Several key distinguishing factors highlight the differences between OSS and proprietary software projects. We compare the two from the following perspectives.

Community Issues relevant to the communities formed around the projects, the different actors, motivation and membership, and decision making processes.

Software Production How code changes are managed, the testing processes followed, the release management approaches, and the technical environment and infrastructure.

Business Issues relevant to licensing, business models and decisions, and how adoption and reuse of the project artifacts is encouraged.

Tables 3.1, 3.2 and 3.3 summarize some significant differentiating factors under the above three categories. These are discussed in more detail in the relevant sections of this survey. Note that in these tables we try to present the average picture, as it is reported in the literature, but clearly there are exceptions and this comparison is not representative of every single OSS project.

	OSS **projects**	**Proprietary projects**
Environment	Decentralized and geographically distributed	Mostly centralized in one or few locations
	Rare face-to-face communication, asynchronous means	Regular face-to-face communication and meetings
	Scarce project plans or schedules	Project planning, scheduling
	Non-assigned jobs and tasks, voluntary participation and selection	Work assignmed and coordinated by project leaders
	Massively parallel development	Smaller sized teams
	No explicit system-level design	Rigorous design processes
	No list of deliverables	Strict development guidelines
	Process and code open to all	Closed communication
	Tasks selected based on developers' expertise and interestes	Developers trained and assigned specific tasks
Change management	Implementation and patch submission before final review	Documentation-level changes and review before implementation
Testing processes	Informal	Formal, reporting
	Peer reviewing	By testers, QA engineers
	Community and user participation	Within company/organization
	Strong community involvement after release	Mostly completed before release (or during beta-testing)
Release management	Frequent releases of minor improvements, no fixed dates	Infrequent major releases on fixed dates
	Minimal promotion and marketing	Promotion and marketing
	Scarce documentation and community-based technical support	Documentation and formal technical support

Table 3.2 Differences usually encountered between OSS and proprietary software projects. Part b: Software production

OSS and proprietary software are the two extremes of a spectrum of potential approaches. As a result, a variety of hybrid models emerged that encompass elements from both. The goal of these models is to capitalize on the strong points that each perspective has to offer.

Typically, proprietary and commercial software projects are characterized by strong emphasis on the user requirements elicitation and design phases, rigorous documentation, strict scheduling and assessment processes, high level technical support to the client base, and of course business mechanisms that facilitate adequate profitability.

OSS on the other hand is mostly characterized by open processes, fluid and self-organizing communities based on widespread communication, a development culture stemming from collaboration, self-assignment of tasks and

	OSS **projects**	**Proprietary projects**
Licensing	Some version of copyleft licensing No cost of acquiring software	Proprietary license, copyright Software sold at a cost
Business models	Value-added packaging Services and Support Loss-Leader Widget Frosting Accessorizing Dual Licensing	Product, service or hybrid, Licensing, royalties, maintenance
Adoption / Reuse	Permitted at all levels, based on specific license	Incorporation and support of reused element's production processes

Table 3.3 Differences usually encountered between OSS and proprietary software projects. Part c: Business

peer-reviewing, participation of the user base in the project, and frequent releases to reflect the outcomes of different development tasks carried out by independent groups of developers.

3.2 Project Success

We briefly discuss project success from the following three perspectives.

- What prerequisites should a project fulfill in order to have a good chance of being successful.
- How to ensure the sustainability of a project that has reached a certain level of success.
- What indicators can be used to distinguish a successful project from other, less successful ones.

3.2.1 Important Prerequisites for Success

There are certain preconditions that should be taken into account at the first steps of an OSS project's organization, and are key to its success. We briefly discuss the main ones that have been identified in the literature.

Project Starting Points Whether the project is initially conceived in the mind of one person, or among a group of developers, it starts off as an idea within a topic of interest. The strength and originality of this idea, together with any initial hints as to how it may evolve into a software artifact are key to

the entire endeavour [78, 36]. Whereas with proprietary software the market pressures are likely to make projects based on mediocre ideas quickly cease to exist, with OSS such projects could be allowed to languish for a long period of time.

The audience that the project will be targeted toward should also be clear from the beginning. End users, software vendors integrating a component, system administrators, or other developers are possible users. The project should identify valid and important needs and provide effective solutions for them [36, 40], otherwise it may not attract real interest.

Finally, though secondary with respect to other decisions, the natural language of the project may play an important role in the exposure it will have and the user base it will reach [36, 148]. English of course is the language that appeals to the largest audience.

Community Organization The decision making processes, hierarchical structure and leadership model for the project community should be decided upon in a way that reflects the mentality of the project creators. A meritocratic culture should also be inspired and enforced to the extent possible [79].

Technical Issues An appropriate choice of technical environment and infrastructure (operating system, internet connectivity, tools, etc.) [148] as well as programming languages [36] is important for an efficient software development process. With OSS additional issues may include hosting providers for the project repository (such as SourceForge[1] and GitHub [2]). Moreover, the choice of external project dependencies has been shown to affect the success of a project under development [95].

Open Source Perspective The choice of licensing scheme [148, 40, 79, 21] and the definition of an effective and reasonable business model [79] are both issues that should be tackled from the beginning of the project. For example a license or business model that does not reflect the aims of the project community in terms of freedom or plans to use the project outcomes

[1] http://sourceforge.net/
[2] http://github.com

may lead to internal conflicts, forking (see Section 4.3.4), or even demise of the project after considerable effort has been invested.

3.2.2 Factors in Project Sustainability

Ensuring the sustainability of a project after it has started, and making sure that it will not stagnate or loose its momentum requires further attention. We briefly examine the evolution and sustainability of both the software system, and the project community.

Software System Sustainability As the software evolves to achieve the identified goals and cover its specifications, its documentation and testing material has to be constantly developed and adjusted as well [175, 185]. Over time, software maintainability will become an increasingly more complex problem. This phenomenon, defined as "software aging" [178], has been shown to equally affect OSS and proprietary software in [200, 256, 131]). Consequently, a clear definition and visibility of the entire software architecture (a task that is often beyond the scope of individual developers) will greatly aid the maintenance tasks. This is a particularly sensitive matter for OSS projects, which are based on a largely distributed developer base, and have also been criticized for poor documentation quality (see also Section 10.4).

Re-engineering parts of the software may be required as well [200], although due to the distributed development nature of OSS projects this needs to be carefully coordinated. Operational support of the resulting product (including tasks such as configuration management) will also aid in making the system maintainable in the long term.

Community Sustainability The sustainability of the project community also needs particular attention. Initially, it is important for the project's success to build a good first impression for the product or service offering. (For example, Choi et al. show that first impression plays a critical role in attracting more developers [32].) Afterwards, it is crucial to keep a high level of developer and user satisfaction [37], which can be achieved through continuously evolving and taking advantage of the developers' skills and competencies [36] and carefully balancing and matching the project development tasks

with the skills and areas of interest of the developers. In OSS efforts these skills and competencies may be the main motivating factors for the developer's contributions, as well as an effective safeguard against the possibility of forking.

The relationships between project members also need to be carefully managed and coordinated, and care has to be taken in maintaining a widely accepted leadership model,[3] while at the same time providing adequate recognition of the achievements and efforts of all community members.

3.2.3 Success Indicators

In identifying projects that stand out as particularly successful, the following indicators could be used (a more comprehensive study is presented in reference [143]).

Development Stage A project that has progressed into more advanced development stages (e.g. from alpha to beta and then to stable [37], or in a SourceForge development stage of 4 or higher [168]) is more likely to be a healthy project, backed by an active community of developers.

Popularity A frequently downloaded project, (e.g. one with a download ranking of 85% or higher [168]) is likely to be a successful one [206].

Activity and Visibility Frequent and visible development [206] or bug fixing activities are indicators of a healthy and successful project. The availability of defect logs, and the average time elapsing between bugs being reported and fixed are very strong such indicators [168]. A project with accumulating bug reports that are not acted upon is likely one that is no longer actively supported.

Tools and Software Environment The software tools that are made available by the project (e.g. for automated unit testing), the portability of its code across different platforms, its compatibility with other programs, or signifi-

[3] See Section 4.2 for more on the role of leadership in OSS projects.

cant efforts to internationalize and localize the project's user interfaces and documentation, may also be indicators of a robust project.

3.3 Representative Examples

We have already introduced various OSS applications and projects, and in the rest of this survey we will examine the characteristics of many of them in more detail. Examples of successful and popular projects that have helped shape the OSS domain include the Apache[4] web server, the OpenOffice[5] productivity suite, the Mozilla[6] web browser, the MySQL[7] relational database system, the Eclipse[8] development platform, Linux[9], the GNU tool suite[10], and various BSD[11] operating systems.

All of these projects share various common defining characteristics.

- They are supported by large, active communities of developers.
- They abide by strict guidelines concerning their software development and governance processes.
- They have set up foundations to manage the projects.
- They are partly financially supported through sponsorships and donations.
- They include working groups responsible for the promotion of the project, and the translation of the software into different languages.
- They organize conferences and workshops to promote the project results and advances.
- Many contributions come from paid developers.

More details about the above, as well as other projects can be found in the relevant chapters of this survey, as well as in their online web sites.

---

[4] `http://www.apache.org`
[5] `http://www.openoffice.org`
[6] `http://www.mozilla.org`
[7] `http://www.mysql.com`
[8] `http://www.eclipse.org`
[9] `http://www.linux.org`
[10] `http://www.gnu.org`
[11] `http://www.freebsd.org`, `http://www.netbsd.org`, `http://www.openbsd.org`

4

Communities

Around any successful OSS project an extensive community of people is formed. This dynamic group features members at different roles, capacities, degrees of involvement and responsibilities. In this chapter we examine the structure, elements and characteristics of OSS project communities.

4.1 Actors

Several studies have investigated the community structures [229, 253, 155] and ad-hoc team formation properties [41, 173, 99] that emerge in OSS projects as a result of the distributed collective processes required when developing software. A notable community structure that emerges in most such studies is the so-called *onion* model: the project development team is organized in concentric circles, where the most inner circle includes developers that have a managerial and leadership role while outer circles have gradually less control on the project. For instance, in reference [38] the authors study project success through a survey of SourceForge projects based on the number of developers, project activity, bug fixing and the number of downloads. Promotion strategies permit community members to transcend community roles [58, 114].

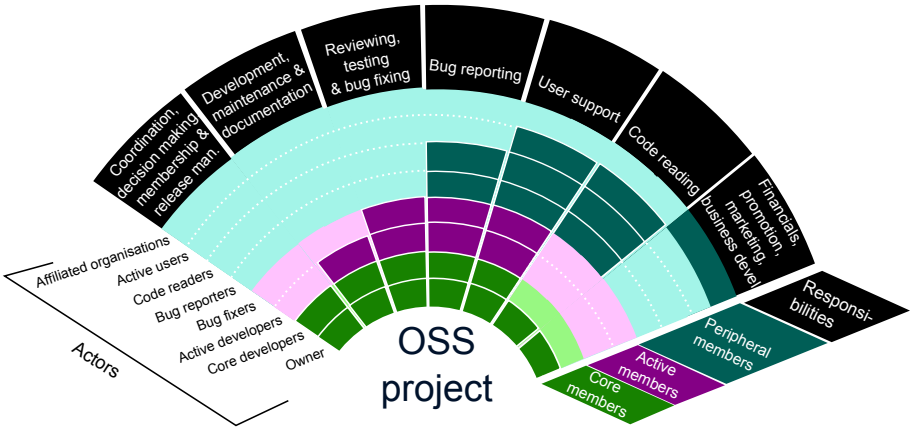


Fig. 4.1 The main actors of OSS project communities grouped as core, active and peripheral members, and their main roles and responsibilities. Additionally passive users, who can contribute to the projects' motivation through download and use, complement the entire projects community.

A more detailed view of project organization in concentric circles is provided in Figure 4.1. The figure analyzes the typical structure of an OSS project community, highlighting the main actors, how they are grouped in membership categories and what responsibilities they undertake. Note that although this subdivision of responsibilities and roles may give the impression of a strict hierarchical organization, the OSS community does not usually perceive it as such, but rather as a natural extension of the division of labour [19].

The size of the core/active parts of these project communities can vary significantly, from many projects consisting of a single person to relatively large projects involving several hundreds of active members, while the number of core members is typically considerably smaller than the number of peripheral members. Empirical evidence can be found in studies of large OSS projects, like Apache, Linux, Mozilla, and Gnome, based on data retreived from online repositories [162, 129, 163, 128].

The various actor categories are outlined below, while Chapter 5 includes more details regarding many of their functions in the software production process.

4.1.1 Core Members

Often OSS projects are inspired and initiated by one person, the project owner, as opposed to an already formed community [185, 11]. In these cases, even if a number of core developers join the effort, the owner keeps setting the vision and often maintains most of the roles and responsibilities for the project, including, apart from the obvious technical development tasks, making decisions regarding the direction the project will take, managing the releases and the licensing schemes to be adopted, and any business models required to ensure the project viability [19, 145, 21, 129, 174].

In OSS projects that consist of more than a very small number of members, however, the main strategic decisions about the overall project direction are the responsibility of a subgroup of active and frequently contributing members, often referred to as the core developers [80, 11, 170, 254, 191, 19], who are generally responsible for a large part of the project outcome [162, 80, 163, 126]. This "inner circle" of programmers [128] varies in size, and may also vary in time (e.g. for GNOME the size was at a certain point in time around 50 developers, for Apache around 15 at a different time [128], and for FreeBSD 9 [117]).

Core developers typically have voting rights and code commit privileges [162, 163, 21, 19]. Apart from their valuable technical contributions to the code, they are also responsible for setting up and/or participating in committees with more specific responsibilities, most notably the release management committee. This committee is responsible for deciding what will be included in future releases of the project. Decisions are made through a voting process (although in various cases instead of a committee there may in fact be only one person)[73].

Some of the core developers who are responsible for handling specific modules or code areas (see Section 5.1) and their related commits are also known as maintainers [117, 53, 73, 39].

4.1.2 Active Members

The active members of an OSS project consist of active developers and bug fixers [80, 170, 254, 19, 73].

Active developers are programmers who regularly contribute bug fixes and code for new features, as well as relevant documentation [11, 170, 254, 19]. They need to have a good understanding of large parts of the source code and the project architecture [170, 254]. As a result they often constitute the most important development force, along with core members.

Bug fixers fix bugs reported by bug reporters (or sometimes bugs they come across themselves), and therefore need to have some understanding of parts of the source code of the project [11, 170, 254]. Often bug fixers do not have commit privileges, but they submit a patch to an active developer.

4.1.3 Peripheral Members

Peripheral members have a more sporadic participation in the project, yet they constitute the majority in OSS project communities [80, 170, 254]. These members share a common interest in the project and identify with it, and are in touch through the different communication channels used, thus both benefiting from and contributing to knowledge exchange and build-up. Reference [258] contains an extensive literature review as well as some relevant empirical findings.

As OSS project communities tend to be of a particularly dynamic nature, peripheral members who maintain active involvement and consistently offer valued contributions may be granted greater systems access and more central project roles by the project's core members. For a more detailed discussion of the these role transformations see also [170, 254, 58, 114].

The following are the main categories of peripheral members.

Peripheral developers contribute to the code from time to time, usually with some localised bug fixes or minor new functionalities, as evidenced by a number of case studies [170, 254].

Bug and problem reporters identify and report bugs or other issues [80, 11, 170, 254, 19].

Code readers are involved users of the project software with an interest in the source code, which they study and sometimes also review and comment

on [254].

Active users are users of the project software who typically participate in the relevant fora, offer support to other users [11], and may also occasionally act as bug reporters or participate in requirements elicitation and testing. For a discussion of these activities see also [137].

Affiliated organisations or businesses offer various kinds of support, such as financial, business development, and promotion [68, 11].

4.1.4 Passive users

Although they cannot be considered members of the project community in the strict sense, the passive users of the resulting software system have an involved attitude toward the project, and may offer their critique and comments [68, 11, 170, 254].

4.2 Leadership

The OSS project community is organized around and led by a governance and leadership structure.

4.2.1 Leadership Models

Despite the broad nature of OSS projects, clearly defined and well-structured governance models are often found in their communities [154, 174], and can be roughly distinguished in two broad categories: monarchical and federal [154, 129, 73]. [154] in particular describes a case study that was performed with focus on the motivation for and coordination of software development work in OSS projects, and was based on literature reviews, interviews with OSS developers, and other research findings.

In **monarchical** (or "star-shaped" [156]) leadership models the initial project inspirer (and owner) maintains a central role, and the entire project is based on his/her decisions and visions [154, 129, 73]. A variation of the monarchical model is the *hierarchical* model [35]. In larger projects, such as the Linux kernel, to facilitate fast work turnover, the project leader transfers portions of its authority to a selected group of core team members; in such

cases, the leadership model effectively resembles that of a military organization.[1] Well known examples of this include Linus Torvalds of the Linux project [52] (sometimes humorously described as a "benevolent dictator") and Larry Wall of Perl [154].

In **federal** leadership models, the responsibilities are more "democratically" distributed among the core members of the project.

Examples include the FreeBSD[117], the Apache Group [154], the Debian's Core which consists of the Debian Project Leader and a periphery of maintainers [197, 156], and Mozilla's Core Team which consisted of Netscape Employees [154].

A project's leadership model evolves as the project matures or when new requirements emerge [174]. Moreover, the project's source code structure has a measurable effect on the potential of a developer to become the project's leader. [90].

4.2.2 Skills

The project leadership naturally arises through a meritocratic process, based on factors such as technical proficiency, knowledge, dedication to the project goals, as well as the number and importance of the contributions made [21]. In the monarchical model, only leaders that are perceived as outstanding manage to generate significant project communities. As a result, trust and respect are reflected in the formation of the leadership team. In practical terms, both reputation and peer review processes are utilised [70, 205, 128, 53], whereby the role of developers includes reviewing the participation and code contributions of other developers (their peers) [156]. For more details see reference [156], which discusses the relationship between institutions and the projects' sustainability and evolution both from a theoretical perspective, and empirically based on the Debian project.

Based on the above observations, and as evidenced by many OSS projects, important prerequisites for a developer to become a core member of the project include a perception of fairness and trustworthiness, dedication without ego-based or political bias [145], and active participation and long-term contributions to the project. Paola Giuri and her colleagues through an em-

[1] In fact, Linus Torvalds refers to his trusted team of patch integrators as lieutenants

pirical study of the roles, skill profiles and activities of individuals registered with OSS projects in SourceForge.net found that a good balance between technical and social/leadership skills is important [90]. Nevertheless, purely technical skills and talent are in some cases given more weight [145, 205, 73, 53]. Good communication and social skills are also important and allow for "leadership without coercion" [103], clear communication of the project's vision and goals, attracting new members to the project and securing funding and support [73].

4.2.3 Structure

Ultimately, the organisational structure of OSS projects stems from the combination of (or conflict between) a flat, anarchistic and free nature [185] that is inherent to the project (as there is no enforcing institution, developers are geographically distributed, and do not often meet face to face), and a clear, layered, hierarchical leadership model that the project community agrees on and tries to impose.

The organisational boundaries themselves are usually open to everyone, in contrast to traditional organisations, and membership is fluid. This allows new contributors to enter, and constantly innovative ideas and perspectives to be introduced to the project [145, 170, 205, 163], forming what are described as "horizontal innovation networks" [234].

Finally, whatever governance structure ends up being defined and followed, it is likely to change with time and need as a result of project evolution and the actions of individuals [205] (see also Section 4.5).

4.3 Governance Processes

Some of the main governance processes of OSS project communities include management of membership, allocation of tasks, and decision making.

4.3.1 Membership Acceptance

Membership in OSS projects is generally open to all. However the management of acceptance into and level of participation within the various technical or governance groups, as well as in the core team, is handled either by the core members of the project (e.g. the Core Team in the Apache project

[70, 163, 205, 154]) or by a designated group (e.g. in the Linux project [154]). These groups have total authority over admitting members based on their contributions and on an assessment of their skills, usually through a voting or consensus procedure (this process can take as long as 6 months in the Apache project). In more extreme cases members can be demoted or expelled if they are judged to have misbehaved.

A model that is closer to proprietary software development was followed in the case of the Mozilla project, as many of the contributing members were paid by the project for their work, and the project authority and control was largely maintained by the Mozilla organization [163].

In another example, a "joining script" is followed by candidates to the Freenet project, whereby a set of significant contributions have to be made, such as offering technical advice or developing high-quality code, in order to be accepted as a developer [238].

4.3.2 Membership Promotion

Projects that maintain community membership processes or requirements, usually prescribe promotion or demotion rules for development team members [56, 114, 35]. Such rules may require from project members to actively contribute to the source code repository during regular intervals [56], to introduce significant (based on the community's judgement) work [58], and finally to be a long-lasting outstanding member of the community [35] in order to join the project's governance board.

4.3.3 Labour Division

Labour division refers to the allocation of different tasks to developers. Due to the open and voluntary nature of the projects and communities, this process is based on trying to balance various often conflicting considerations.

The developer's preferences are taken into account and, to the extent possible, respected [185, 163, 170, 21]. Developers can always choose what to work on in open source, but the project may not accept contributions that are not solicited/desired. The skills and area of expertise of the developers are important, and the goal is to couple them with the open tasks requiring work [21]. However in some cases a more loose approach is followed, where developers can pretty-much choose what they will work on, and are not strictly

assigned tasks [205, 170], sometimes leading to a "problem of choice" This is discussed in reference [46], which presents a stochastic simulation of the allocation of resources.

The notion of code ownership [163], whereby one developer is mostly responsible for a module or section of the project, and other developers make secondary contributions to it, is usually strong. This is usually not a predetermined role, but emerges as a result of particular expertise or recognition in a particular area, whereas in other cases it is actually enforced through the task allocation process.

Stefan Koch and Georg Schneider, through data retrieved from the Gnome project's CVS repository, found that a small number of developers usually work on the same file [128], which is considered an indication of a high degree of labour division. The "stigmergic" theory [53], inspired by self-organization theories, further examines this by arguing that the community reacts to "stigmas", or signs in the code base (such as the existence of a specific code tree), and that the hierarchical nature of the project architecture is a means of revealing a related hierarchy of preferences among developers. Finally an association often exists between the degree of collaboration and the code complexity [53].

The process followed to achieve the desired division of labour and allocation of tasks starts with the pending jobs being communicated to the core team and the rest of the community, usually in mailing lists, or through bug and issue repositories. For example, in the Mozilla project the Bugzilla bug tracking system is used by developers to seek help for issues they are having with their code. The selection of tasks is based on personal interests, skills and capabilities, and the allocation of tasks to volunteers follows the considerations described above. It is not possible to force developers to work on a task if they do not want to. In case a certain task finds no developers willing to or capable of working on it, the task may be set aside temporarily, at the risk of the entire project being jeopardised if it is crucial [21].

This form of labour division is not as inefficient as it may sound. Allocating development resources according to the developers' interest ensures that only features for which there is genuine interest get implemented, while elements lacking community support languish and die. This is commonly expressed in OSS projects through phrases such as "put your code where your mouth is" or "shut up and code". A more detailed discussion of the motiva-

tion driving the developer contribution can be found in Chapter 9. Furthermore, when there are no developers willing to work on an essential feature, this often indicates that the feature's code base is in a bad shape. A common outcome in such a situation is a developer stepping in for a complete rewrite, which benefits the whole project's code quality. Thus a dynamic process helps to ensure that an OSS project's essential code parts are in a state where developers can actively maintain them.

4.3.4 Technical Decision Making

The most important decisions related to the project code are made by the core developer team. Such decisions may include code development orientation, what to include in future releases, architectural issues, how to handle critical bugs, etc.

Such decisions are taken through a process of voting and consensus [163, 90]. Usually any developer can vote or express an opinion (for example through the project mailing lists), but only the votes cast by the core group are considered binding [70, 205]. This may lead to disagreements, for example if the "owner" of a particular part of the code does not agree with decisions made by the core team.

In these cases the owners may try to exert pressure to the core team to adopt their opinion [156], but if this fails they may in some cases decide to abandon the project, or even start a different project adopting their own approach. This is called "forking", and it is one of the risks faced by OSS projects [129] (see also Section 4.4.1).

4.4 Coordination Challenges and Mechanisms

OSS project coordination is subject to a variety of challenges, due to the large and variable nature of their communities.

4.4.1 Co-operation Challenges

OSS project development and coordination is faced with the following main challenges.

Geographically Distributed Development Due to the wide, variable and decentralised nature of the development community, face-to-face interaction is rare [163, 205, 21, 197], the multicultural profile of the team may introduce communication difficulties, and language, customs, even different time zones make coordinated development even harder. The communication distances have been found to lead to lower development performance and overall delays [108], though there is also conflicting evidence. In particular, see the empirical study of the effect the global nature of the FreeBSD project has on productivity, quality and cooperation [212].

Modularized Code Production Decentralization, and the fact that voluntary developers usually cannot devote large amounts of time to the project [205], result in development tasks being highly modularized [145] and requiring even more careful coordination to avoid "stepping on each other's toes" [21]. The notion of "code ownership" helps in this respect, but introduces other risks (see below).

Heterogeneity in Opinions and Aims, and Risk of Forking Conflicts, differences in opinions and arguments are more frequent than in traditional projects, due to the difficulties in coordination and communication [156]. These may lead to forking, as discussed above, which may be initiated by code owners who feel they should have decision making power and believe that the project is not going in the direction they want [156]. A notorious example of a forked project is the 386BSD operating system, which forked into FreeBSD and NetBSD. Later on OpenBSD was forked from NetBSD, and DragonFlyBSD was forked from FreeBSD. All systems but 386BSD are actively maintained, have a loyal user base, and regularly exchange technical advice and code.

Free-Riding A common problem is that of community members not contributing anything to the project, while benefiting from the contributions of others [61, 130], e.g. through code, solutions to problems, algorithms, design features etc. that they can use for their own benefit. Note that this is an issue with project developers and not passive users, who are always welcome and beneficial to a project's community.

Handling Code Complexity The complexity of the development process itself puts considerable pressure on the developers and may lead them to coding solutions, approaches or styles that are not recommended and may render the code base hard to debug or maintain [197, 156].

A further challenge is related to two activities that have been termed *exploration* (the extension of existing technologies or the development of new features) and *exploitation* (the use of already developed knowledge without changing its nature) [197]. For example, in the Debian project core developers are mostly responsible for exploratory activities, whereas peripheral developers mostly focus on exploitation of existing code. Although the two are not necessarily mutually exclusive, finding the right balance between them is often a difficult task.

4.4.2 Cooperation Tools and Mechanisms

The lack of a project plan or deliverables schedule [163, 205] means that the remedies to the above issues are mostly based on either synchronous or asynchronous communication and cooperation avenues. The most frequent mechanisms are the following (see also Section 5.6).

- Free-form discussions based on email, or IRC channels
- Structured discussions based on discussion forums, bug tracking systems,
- Mailing lists
- Newsgroups (USENET)
- Chat rooms for direct communications
- Wikis

4.4.3 Conflict Resolution

OSS project communities are generally characterized by common, pervasive "cultural" characteristics [154, 185, 205, 221] (the "hacker" or "geek" culture) that unify the group of technically oriented and passionate people through a set of common beliefs (e.g. in the freedom of software and the choice of work), values (e.g. community and cooperative work) and norms (e.g. open disclosure and acceptance of outside critique) [61]. However pitfalls still exist, and conflicts or arguments often need to be addressed and dealt

with. These are usually the result of issues with compliance (e.g. behaviour, fairness to other, respect for property rights and discipline), reciprocity (e.g. with respect to code reviews and help) or social pressure (e.g. hostile emails, shunning, spam etc.) [205, 70, 162, 204, 154]. Conflicts are thus dealt with through the following approaches.

Rules and Institutions Generally accepted rules, guidelines and protocols are often employed [130, 21, 205, 156] that encompass the common notions of validity, describe responsibilities, rules, processes such as voting, communication, code submission, notification of new issues, and licensing issues. Examples include the Debian Social Contract[2] and Free Software Guidelines [156].

Monitoring and Reputation Some projects include procedures to share information about the actions of developers [130], such as the volume of contributed code, members that are inactive over a long period of time, or code changes that resulted in a build failure. Reputation mechanisms may be used to reward good work and correct behaviour [154, 205], and maintain or attract developers, while peer review and supervision, as well as parallel debugging [128], are also widely employed [205].

Hierarchy and Authority Authority, at all levels, plays a role in the avoidance and resolution of conflicts either through applying governance procedures, or through regulation and control [156, 130]. Quorum voting systems can be set up (e.g. in Apache) [163, 205], while the administration hierarchy and code ownership structure also impose control over the intensity of controversies, as they are based on trust and experience and are therefore usually respected [156].

When required, sanctions or different forms of punishment, usually to do with their participation in (or expulsion from) different groups, may be applied to shape the developers' behaviour [130].

[2] `http://www.debian.org/social_contract`

4.5 Evolution

There are conflicting theories regarding the adequate size of the project's development community. There are those who believe that "given enough eyeballs, all bugs are shallow" [185], whereas others caution us of the risk of "too many cooks in the kitchen" [160]. And although it is generally believed that an increased number of developers does not lead to a proportional increase in productivity, studies such as Koch's [126], which analyses both small and large, successful and failed projects and their programmers using version-control data, have shown that the attraction of a larger number of participants in OSS projects is generally beneficial.

In any event, the project's sustainability depends on the evolution of its community [128, 47]. We thus briefly overview the processes and requirements for membership augmentation and retention.

4.5.1 Membership Augmentation

Many OSS projects have a specific marketing and promotion agenda [172], which usually involves publicizing the projects' attractive features and the benefits it may bring to new members. These may include job opportunities and knowledge acquisition [37, 167], and specialization in particular areas of development work [221].

New members are also likely to be more attracted to projects that are healthy and successful. Indicators of this may be frequent releases [103], good project management and communication avenues [221], and a strong development community (good programmers attract good programmers [18]).

Important attractive factors also include open access to mailing lists and involvement in the project for new members [37] as well as a pleasant environment and a culture of providing support and mentoring to new members [137, 5, 167, 221].

4.5.2 Membership Retention

Equally important to the augmentation of the community with new members is the retention of the current ones. This favours projects that maintain an atmosphere of fun and motivation [172, 5] (see also Chapter 9), and place em-

phasis on trust, quality of communication between members [221], a sense of membership and identity [167, 221] and a categorization of roles that relates recognition of the members' contribution [167].

Fairness and reciprocity are also important and affect the willingness of volunteers to contribute [204]. These can be enhanced through reputation [37, 221] as well as reward and penalty [205] mechanisms.

The community must be kept balanced in terms of the required participation in different roles, in order to be kept sustainable [170, 162].

Finally members should be allowed to engage in relationships with outside entities such as customers or vendors, and not be given the impression that they are "locked" inside the project and not free to leave the community if they wish to [205].

4.5.3 Technical Requirements

There are also a series of more technical requirements that favour the evolution of an OSS project community. The interaction between developers, and especially new ones, should be facilitated through the use of shared repositories with clearly defined access rights [167], as well as the use of content meta-data marking to make it easier to search [1]. The code should be constantly maintained so that it is fresh, sanitized and well documented for new developers to extend [1, 103]. And finally, within a variable and distributed development community a more modular codebase and architecture is easier to work with than a monolithic one [6]. In this last article the authors devise a model to show that the architecture of a codebase affects developers' incentives to work within the framework of the open source development process, and thus argue that it can have a major impact on the sustainability and value of such processes.

5

Production Process

The software production process within an OSS project depends on the project's organization, governance, community structure and goals. As has already been stated, not all OSS projects are the same, and therefore they don't all adopt the same software production processes. However if we focus on established, successful projects with communities of considerable sizes, we can identify some common distinguishing characteristics and traits. These are the focus of this chapter.

The open and collaborative nature of OSS project communities, with public contributions, freely distributed documentation, and wide participation in technical and managerial decisions, offers significant opportunities for learning and skills development (known as the "technology spillover" effect [19]), as well as practices such as massively parallel debugging [93], whereby developers utilize and peer-review each other's code[1].

Various phases of the OSS software development cycle also exhibit specific characteristics. The requirements definition and elicitation phase can involve both end users (who are closely related to the project community) and project developers, often resulting in more direct understanding of user needs

[1] Practices such as this have been found to result in improved code quality [129, 157, 231], although this view has been repeatedly questioned in empirical studies [77, 200, 179, 152, 214, 49].

and requirements, or problems that need to be addressed. The incorporation of new features and the integration of newly developed code follows specific multi-stage procedures including prototyping, voting, reviews and testing within specially set up developer groups. Finally the management of releases, including timing, procedures, packaging and distribution, which usually includes the project source code, is also undertaken by developer entities collaborating at different roles and responsibilities.

The OSS development process is also characterized by increased modularity, which is key in producing a clear and understandable design and allows autonomous, contained, and independent contributions by separate groups of developers. As the community participates in product development decisions, technological considerations and innovation are taken into account and incorporated in the end product. Often technologically superior alternatives are chosen, even if they are not economically the most favorable solutions. A risky experiment with a new promising, or even controversial, technology is at most one branch away from the main project trunk.

Table 5.1 summarizes some key differences between software production within an OSS project and a proprietary software development firm. These are expanded upon in the following paragraphs.

5.1 Modular Development Methodology

Empirical studies, such as the one on the design structure of Mozilla and Linux by Alan MacCormack and his colleagues [152], have shown that a fundamental characteristic of many OSS projects and development methodologies is the decomposition of the system design into separate modules. Modularity characterizes a system whose parts can be designed and implemented independently, but will work together to support the whole [6]. The compatibility of the different modules is ensured by a set of horizontal architectural design rules. There is a trend for more modular design in proprietary software products as well, and it can be argued that to some degree this is the result of learnings taken from the OSS domain.

Various degrees of modularity and coupling between components are possible, with monolithic designs at one end of the spectrum, and loosely coupled ones at the other.

The core architecture of many large OSS projects can generally be de-

Open Source Software	Proprietary Software
Developer community organization	
Individuals or large, open, variable community	Company employees
Widespread ad hoc collaboration	Hierarchical organization
Enthousiastic, motivated, hacker culture	Corporate culture
No conflicts of interest	Potential conflicts, friction
End-user involvement high	End-user involvement low (but rising)
Governance and project management	
Informal management structure	Strict hierarchical management
Decisions taken through code contributions, voting, discussions, disagreements	Decisions taken by management
Responsibilities and tasks allocated through fluid, informal procedures	Responsibilities and tasks strictly assigned by management
Possibilities for tasks without developers, forking, duplicate efforts	Organized control of resource allocation and effort management
Software development procedures	
Innovation and technological considerations more than financial ones affect decisions	Decisions mostly economically driven
Massively parallel debugging and peer reviewing processes	More isolated efforts, source code not shared
Design modularity and loose coupling are critical	Design modularity more optional
Requirements arising from project community and associated users	Requirements arising primarily from the market and formal studies
Documentation sometimes not formally developed, sometimes of lesser quality	More rigorous documentation enforced by company standards
Usability issues not always addressed, user interfaces sometimes poor	Considerable emphasis on usability and user interfaces
Technical infrastructure	
Needs infrastructure for collaboration, communication, distributed development	Distributed development infrastructure not always critical
Internet-based repositories used	Code and documentation held within company limits
Project evaluation and monitoring	
Project status assessment involves project and community health, evolution, contributions, code quality, and resolved issues	Project status assessment based on lists of tasks, functionalities to be implemented, open bug reports, and expended effort
Software release and distribution	
Frequent releases, loose release planning, feedback from community seeked	Rigorous release plan, infrequent releases
Web-based and community-based distribution channels	Software directed to market through standard sales channels

Table 5.1 The main difference between the software production process followed within an OSS project and a proprietary software firm.

scribed as a platform that supports modules that are essential to the system, and a set of distinct modules on top of it. For example, in the Linux operating system the kernel is part of the platform, and the device drivers are independent modules [6, 93].

At the time of writing, version 2.6 of the Linux kernel included almost 2500 loadable kernel modules, supporting device drivers, file systems, sound hardware, networking, cryptography, processor architectures, compression, and security. Indeed, one of the key success factor of Linux is considered to be the efficient modularity of its design [226]. At one extreme end the Eclipse integrated development environment consists entirely of plug-in modules [81].

Apart from the general advantages of a modular design, the following features make it particularly important for OSS projects.

- The design is clear, distinct and understandable [152].
- There is loose coupling between different modules and development tasks, which allows work on a given module to be carried out without affecting other modules in the design [152]. This offers more autonomy and requires less interaction between contributors (as shown at least in the case of the GNOME development [84]). As dependencies are minimized, development can take place at a global scale, around the clock [212] with minimal effects on product quality (as shown in the case of the Windows Vista development cycle [15]).
- It attracts more voluntary contributions (as opposed to monolithic codebases [6, 152]). Individual code contributions can be small and contained, while still allowing their sum to be very valuable [116, 183]
- It promotes synergies and cooperation opportunities [19].
- Experimentations and exploratory implementation attempts can be accommodated more safely, and changes and improvements can be performed without jeopardising the overall system [6].

5.2 Requirements Definition

The first step in the software production process consists of the requirements elicitation, analysis and definition (the requirements usually stem both from

the developers' interests and the users' needs), analysis and definition. In order for OSS projects to be sustainable and successful, requirements of two types should be considered [201, 84].

Technical requirements are the typical requirements that the design and implementation of any software project is based on, and include the desired functional and non-functional characteristics and features of the resulting product. The project leaders will usually outline these in a vision statement, and they may be subsequently enriched with post-hoc features during the development and evolution of the project, as a result of interaction with users and testers, or based on the input received by the project community.

Environment requirements do not refer to the actual technical artifact, but to the nature of the project and the community surrounding it. For an OSS project to be able to retain the interest of its development community, it must provide constant motivation and incentives for participation through vehicles such as research interest, challenging technical problem-solving aspects, and involvement in new technologies that may allow new career opportunities (see also Section 9). It is doubtful that projects lacking such characteristics will be able to maintain the required momentum and retain their developer base.

The participants in the requirements elicitation phase of the project are mostly developers and documenters, but may include other peripheral members and volunteers, investors and other stakeholders who have an interest in the project, potential customers or prospective end-users, scientists, etc. [201, 231, 189, 84, 72].

In general the requirements elicitation in OSS projects requires less interaction with external end users, as they are mostly understood within the project community [72]. Indeed often the developers intend to be users of the system themselves, and have specific needs and requirements that need to be addressed. Moreover, requirements in OSS projects are usually not formally documented [202]; it is very common for requirements to be lurking in email documents, TODO lists in the project's repository or bug reports and feature requests in the project's issue tracking databases [163].

The sources of information from which the specifications will stem usu-

ally include technical reports and system documentation, mailing lists, newsgroups and discussion forums, as well as accounts of developer's own needs and perspectives [201].

5.3 Incorporation of New Features

The incorporation of new features into the project usually follows a procedure that may involve creating a short description or even a proof-of-concept prototype, voting to select among different candidate features, gathering specific requirements and finally designing and implementing them (see also [201, 84, 4]).

The initial requirements emerge through interactions between the project developers, users and other members, while the feature selection and prioritization process is mostly a responsibility of the core team and the code maintainers. Final validation is achieved through the actual implementation of the new features.

5.4 Code Integration

The procedure typically followed for integrating new pieces of code in the repository is outlined below (see also discussions in [51, 163, 162, 84, 117, 72, 56]).

Code Reading and Familiarisation The developer first studies the existing relevant code in the project repository.

Development and Testing The developer carries out the required code modifications, and runs tests to verify their implementation. This is done in the developer's private workspace. Submitting untested changes that disrupt the project's build (compilation process) or even the software's operation is strongly avoided (and particularly frowned upon [73]).

Systematic testing is usually not formally prescribed in most OSS projects, with the exception of some very large and complex ones [220, 56].

Patch Submission If the developer does not have commit privileges, the code changes are submitted to a core developer or maintainer for review and

eventual integration to the project's code base. This is often done through email or through some distributed version control system (see also Section 5.6).

Review and Pre-Commit Testing The core developer or maintainer reviews the code changes and performs further testing before committing the changes to the code repository. Even developers with commit privileges may publish or submit their code for third party review if they are performing a tricky or critical change, or if they are working on unfamiliar code. Some projects, such as FreeBSD, formally assign a mentor to new members of the developer team, and make it the mentor's responsibility to sign-off the new developer's code [56]. Others, such as Linux, employ a multi-level governance hierarchy [35] for gradually reviewing patch submissions.

Code Commitment The (core) developer commits the changed code to the project's version control system repository. This often triggers various ancillary actions that promote collaboration and code stability [59], such as running automated tests or code quality checks, automatic emailing of the change on a mailing list, updating information in the project's issue database (based on information that associates the commit to an identified issue [246]), and informing users who had registered their interest on the given issue about the change.

Once the patch is accepted into the core repository, it will be ready for inclusion in the next release.

Figure 5.1 schematically illustrates the above procedure.

5.5 Release Management

The release of new versions of the project software, including new features, functionality or bug fixes, are crucial moments of the project life-cycle. Various elements of the release management process can also be found in the development of proprietary projects. Our description in this section serves mainly to document the adoption of commonly accepted best practices by OSS projects. An important differentiating factor between the release management of OSS and proprietary projects, is that the source code (and often even the binary executable form) of an OSS project is typically available as a

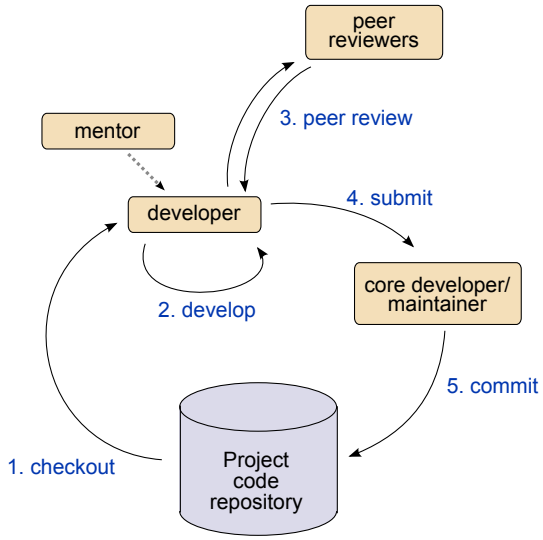


Fig. 5.1 Illustration of the typical procedure followed for the integration of newly developed code in an OSS project.

snapshot of its current state.

Therefore, release management in OSS is often simply the strengthening of various attributes like reliability targets, extent of permitted changes, documentation, and public relations. Specifically, many OSS projects don't base their release on targets for a predefined feature set or target date. Instead, they often aim at generating a stable release by closing all pending critical bugs, prohibiting disruptive changes, documenting the improvements over the previous version, and announcing the new version's availability.

5.5.1 Time for Release

A new release usually takes place under the following conditions (see also [51, 73]).

- A sufficiently important number of bugs has been repaired.
- A significant number of new features has been added to the project.
- Important documentation updates are available for the users.

- Certain promotional or political reasons dictate it (e.g. to present a new feature at some event, or to attract funding or commercial interest).
- A fixed release schedule has been previously agreed on.

5.5.2 Responsibilities

The responsibility for the release is usually shared between two entities: The project core team, or code owner, and an appointed release manager [51, 73, 84, 56].

The core team is responsible for deciding the release time and has the final say on what features will be included in the release. They will then proceed to appoint a release manager, i.e. a project member that will be responsible for the planning and coordination of the release. Usually the selection of release manager is based on previous experience with the project as well as appropriate technical and communication skills.

5.5.3 Release Procedure

The procedure followed during a release, under the supervision and coordination of the release manager, consists of the following phases.

Release Stabilisation This usually consists of three main steps (see also [56, 73, 220, 117, 65, 7]).

(1) All new code patches that are relevant to the release are merged into the code stream to be released.
(2) The code stream is "frozen", meaning that no new features can be added to it. Different degrees of freezing can be applied, for example a soft freeze will allow minor changes, whereas a hard freeze will not allow any changes except critical bug fixes that need to be included.
(3) A final review approval is given for the release go-ahead.

Packaging and Format The material to be released can be packaged in different formats, that may include single self-contained archives, compressed

source code, binary packages, or patch files (addressed to users of the previous versions of the software). The necessary files with installation instructions, copyright and licensing information, along with a list of changes addressed by the current release are also included in the package. A link to the release location on the version control system server can also be supplied.

Naming The naming and numbering of the release is important as it conveys information about its contents and its relation to previous releases, development streams, etc. This is particularly relevant in the modularised and parallel development environment of OSS projects, as it allows organizing and keeping track of changes in a hierarchical manner (and doing this in accordance with the project's version control system) [73].

Based on their degree of maturity, releases are commonly categorised as pre-alpha (not feature complete), alpha (for testing purposes), beta (further testing before release), and release-candidate (ready to release unless fatal bugs emerge).

The stage of development is further characterized as stable (assumes that there are no significant undocumented problems) or unstable (includes enhancements that have not undergone thorough testing, or more changes are expected). In contrast to proprietary software projects, many users of OSS systems decide to run software from the unstable (development) branch, in order to be the first to use new features and bleeding edge technologies, and also to assist debugging and testing a project they want to help.

The numbering of the releases also follows widely agreed-on mechanisms, to declare the main development trunk that the release originates from and the relevant brunches and sub-branches, as well as the development stage.

Pre-Release Testing A final pre-release testing phase takes place within the project community, and may involve selected end-users and volunteers. This goes beyond the basic functionality and includes all new features and installation scripts. Based on the development stage and the feedback received from this testing phase, the release manager will then decide whether to make the release available to the public.

Distribution The new release is most commonly distributed from the project's main distribution server as well as possibly secondary servers (mirrors). Alternatively it can be distributed through peer-to-peer file sharing networks, or even among end-users who forward it to each other [1].

5.6 Technical Infrastructure and Collaboration Facilities

Due to their wide geographic distribution, the large number of developers and other active members involved, and their constant evolution and enhancement with new features and functionalities, OSS projects depend crucially on the right technical infrastructure to support collaboration and development tasks.

We briefly overview the software tools and systems that are most frequently utilized.

5.6.1 Version Control Systems

A version control system (VCS) is crucial for keeping track of the evolution of the project code and documentation in the decentralized OSS project environment. Such systems may be centralized or decentralized. Two commonly used centralized (and OSS) VCS systems are CVS[2] and Subversion.[3] Their cross-platform clients can concurrently access and modify the project files providing features particularly appealing to OSS, such as support for peer reviews and conflict resolution, multiple development branches, data mining, automated notifications, etc.

Decentralized VCSs (such as BitKeeper,[4] Git,[5] Mercurial,[6] Bazaar,[7] etc.), use local repositories on the client systems. These offer better scalability, and more direct member interaction. Centralized VCS's offer more controllable security and privacy, accessibility to history, and better repository management capabilities [177], however several projects are moving to distributed VCS solutions [161].

See [7, 209, 73, 161] for more on the use of VCSs in OSS projects.

---

[2] `http://www.nongnu.org/cvs/`
[3] `http://subversion.apache.org/`
[4] `http://www.bitkeeper.com/`
[5] `http://git-scm.com/`
[6] `http://mercurial.selenic.com/`
[7] `http://bazaar.canonical.com/`

5.6.2 Issue Tracking Systems

Issue tracking systems allow the project members to report bugs, request enhancements and new features, and keep track of pending jobs. Responsibilities for different issues or bugs can be explicitly assigned to specific project members. All of these issues are accessible to the relevant members, and can also be automatically communicated through mailing lists. A complete history of the handling and status updates, including comment exchanges in the form of discussions, are also supported for each issue and maintained by the system. Some systems even allow project members and users to "vote" for fixing a particular bug or implementing a feature. This allows the prioritization of issues according to the view of the project's users.

The current status of issues can be determined at any time, and lists of pending issues (usually prioritized according to their status, urgency or impact to the project) can be produced and distributed within the project community.

Bugzilla[8] is probably the most widely used such system (also in proprietary development environments). Other systems include GNATS,[9] and Trac[10].

5.6.3 Support for Technical Discussions and Communications

The considerable amount of remote communication and collaboration that takes place among OSS project members is usually based on three types of systems and infrastructures.

Synchronous communication applications, such as instant messaging and IRC, allow real-time discussions and instant responses to questions, thus boosting the turnaround time for problem resolution, while at the same time helping establish more informal relationships between the project members.

Asynchronous communication is usually based on mailing lists, Usenet groups, discussion forums and blogs. These allow a more structured form of

[8] http://www.bugzilla.org/
[9] http://www.gnu.org/software/gnats/index.html
[10] http://trac.edgewall.org/

communication, which also leaves behind a history trail that can be searched in the future. Such methods can be used either for technical or non-technical issues, as well as for the broadcast and discussion of ideas and opinions, or the creation of an informal repository of information related to the project.

Web-based dissemination platforms such as wikis and the project's web site usually include varying amounts of user documentation, technical data, and organizational information.

5.6.4 Repositories and hosting facilities

Often OSS projects are hosted on web-based repositories that can be either of a generic nature (such as SourceForge[11] and GitHub[12]) or thematic (such as Java.net,[13] CPAN,[14] and CTAN,[15]). These provide various facilities such as file storage, documentation authoring and presentation, mailing list hosting, on-line forums, source code browsing clients, a build farm of diverse operating systems and processor architectures, version control systems, an issue tracking database, and downloading support [51, 34, 192, 73].

Hosting OSS projects in such widely known sites offers the projects considerable visibility and promotion. On the other hand, projects hosted on independent web sites have a more distinct presence and are more autonomous.

5.7 Assessing Open Source Software Projects

As with any software development effort, but even more so for OSS projects due to their distributed nature, it is important to be able to assess their status and health from the technical and software production perspective. Various software engineering criteria, metrics and tools can be used to evaluate the project, the artifact, as well as the development process.

Project status and health The quality of the project can be examined at an abstract level, e.g. by considering issues such as its testability, simplicity,

[11] `http://sourceforge.net/`
[12] `http://www.github.com`
[13] `http://java.net/`
[14] `http://www.cpan.org/`
[15] `http://www.ctan.org/`

readability and self-descriptiveness [218, 210].

Additionally, software engineering methods have been used in empirical OSS studies concerning community issues such as project structure, governance, coordination and cooperation (see for example [220, 128, 126, 84, 97, 152, 183, 58, 193, 246, 160, 113]).

Project evolution From a dynamic point-of-view, the types and frequency of contributions made to the project can provide an indication of how active the project is. These can be tracked through the project VCS and issue tracking systems, or alternatively by directly comparing source code versions [183, 118, 113]. Similarly, the changes in the project architecture can provide an indication of the project's evolution [92].

Design and architecture Various metrics based on the design structure [152], including its object-oriented nature [97, 139] can give a measure of the quality of the project design and architecture. A wide range of known software engineering metrics[16] have been used in various OSS studies (see also [92, 162, 163, 218, 113]).

Software One can also focus on the software artifact itself, and examine the adequacy of its functionality, reliability, usability, efficiency, maintainability and portability, based on the project specifications, to reveal exemplary or poor coding practices [210, 215]. Specific characteristics of low-quality software can also be used as signs of a project that may end up in trouble (see [210] for various such cases). A variety of tools for software quality assessment are available for both researchers and developers.

Open source projects additionally constitute ideal cases for performing studies and analyses as above, as they provide researchers with a plethora of public data through version control and issue tracking systems, mailing lists, documentation, and the code itself. This data can be analyzed and mined by using or constructing appropriate software tools and platforms [92, 218, 97, 56, 113, 199, 215].

[16] Such as Chidamber and Kemerer's metrics for object-oriented systems, McCabe's cyclomatic complexity, and Halstead's Volume [119], to mention but a few.

5.8 Concerns

Concerns have also been expressed regarding the OSS software production process.

The informal organisation and management of the OSS development process may result in problems such as an inability to achieve the necessary match between resources (mostly developers) and tasks [46]. If the development process is not carefully directed, it may lead to either developer redundancy and duplicate efforts, or incomplete efforts and unimplemented tasks. Additionally, extensive reuse of code by large numbers of developers requires very careful coordination, and may make large-scale changes very painful in terms of synchronisation with the source.

The loose planning and scheduling approach of OSS [93] may also cause difficulties. The frequent release schedule [117, 103, 73] may motivate users and improve code tracking, however it may also lead to unstable code, and it is not necessarily consistent with the development of complex and demanding new features [117].

OSS is also considered to suffer from poor documentation [23], and in some cases a lack of tools for tasks such as requirements management, project management, metrics and estimation of project health, scheduling, and suite design [46].

Finally, the OSS community structure and organization may also entail some risks. The danger of forking [16, 60] due to differences in the priorities or perspectives of core team members or incompatibilities between participating developers can lead to code base splitting and unmaintained code, while developers may also be tempted to keep parts of their work proprietary rather than contribute to the public project in a so-called “war of attrition” [18].

6

Licensing

Open source software can be freely used, modified or distributed, provided certain restrictions are observed regarding its copyright and the protection of its status as OSS. These rights and restrictions are expressed through the software's license, i.e. a contract between the software owner(s) (the licensors) and its prospective users (the licensees) [51]. OSS licenses come in different flavors, but in general they make available the software source code and they permit the creation of derivative works as well as the non-exclusive commercial exploitation of both the original and the derivatives [140].

The licensor of the OSS software (typically the owner or author) may be a single developer, a group of developers, or an organization, and holds the copyright to the software [196]. By assessing and combining factors such as the motivations behind their work, the project's characteristics, its intended audience and its likely success, the licensor decides whether to make their work available under an open source license, and if so what type of license to employ [148].

The licensee, on the other hand, is either an end user of the OSS, or someone who has embedded it in their product or application, which is then further distributed or licensed [196].

6.1 Concepts and Definitions

Before discussing the different types of OSS licenses, we briefly introduce the background concepts delineating the degrees of freedom available while distributing the product of any intellectual activity, including software.

6.1.1 Intellectual Property, Copyrights and Patents

The term intellectual property is used to encompass a wide range of areas of law, including copyrights, patents and even trademarks [83]. These are all means used to encourage private investment in research, technology and innovation, by ensuring that innovators will be able to get individual returns for their work.

Copyright is a form of legal protection that can be applied to a wide range of intellectual works, including software. Often copyrighted software allows no access to the source code, and is distributed with a licence agreement that severely restricts the copying, modification or further distribution of the covered software. However the software's authors may also choose to publish the source code by placing it under a software license that conveys the rights of the parties accessing the source [236].

Patents are written descriptions of inventions, used as property claims covering the core ideas and their use [83], so that only the inventors can extract economic returns from them. Patents constitute permissions on the use of an idea, granted to the authors for a limited amount of time [149]. Applying for a patent is a process that may take years, and involves a substantial financial investment [83, 236, 101, 51].

Since software code can be easily copied and reproduced, many companies argue for its strong patent and copyright protection. However, others express considerable concerns about the use of such protection, claiming that the software community and society in general benefit less from this restrictive approach, relative to keeping the knowledge that the innovators have created free and available to all [101]. In particular, the Free Software Foundation, a non-profit advocacy group, states that the use of patents will severely undermine the free software movement [51, 26]. The concern of the FSF and other members of the OSS community is that most OSS projects lack the financial and institutional resources required to investigate patented prior

art that may cover their work, and to defend themselves against patent litigations, which are notoriously costly [26]. Furthermore, there are significant differences between different countries in patent law, especially for software [51], which poses an additional difficulty for the typical global OSS project. In general, excessive patent protection has been criticized throughout many scientific and technological fields, as it can impede the development of scientific research and render access to crucial resources (such as medicins) more difficult or costly [10].

6.1.2 Public Domain

In contrast to patented software, collaborative models that allow many members to freely and actively participate in a project or development effort has been referred to as free revealing [185]. OSS projects are characteristic examples of this. Control of knowledge, innovation, or in this case source code is thus relinquished, and so they become public domain goods [236].

By labelling software as Public Domain the owner declares that there is no copyright protection and no distribution or licensing restrictions. Anyone is free to copy, modify, distribute or sell the software, without any permission being required [26, 51, 141]. Even if parts of a public domain software product are incorporated into a copyrighted work, then that copy of the material will be covered by the copyright, but the original work is still in the public domain, free and available to all [73].

There is a major misconception equating OSS with public domain software [181]. OSS is not public domain software. It is copyrighted and distributed under a license — just a license that gives the users more rights than they are typically used to.

6.1.3 Open Source and Copyleft

Open source lies in between allowing a software work to fall completely in the public domain (thus relinquishing any notion of ownership), and protecting it under copyright or patent law. All open source software licenses share two characteristics: They waive the right to earn license fees from distributing the software, and they incorporate the condition that the source code will be made available to licensees.

Copyleft (a play on the word copyright) is a form of open source licensing

that grants the right to reproduce, adapt or distribute software. However, it imposes the restriction that any derivative work will be released under the same license. In this way the software and the freedoms applied to it become inseparable [83].

Copyleft licenses are therefore a subset of OSS licenses, further distinguished according to how restrictive they are, and often labelled as strong-copyleft or weak-copyleft.

6.2 Open Source Software Movements

Two main movements and organizations that promote OSS and certify licenses as open source or free software are the Free Software Foundation (FSF) and the Open Source Initiative (OSI).

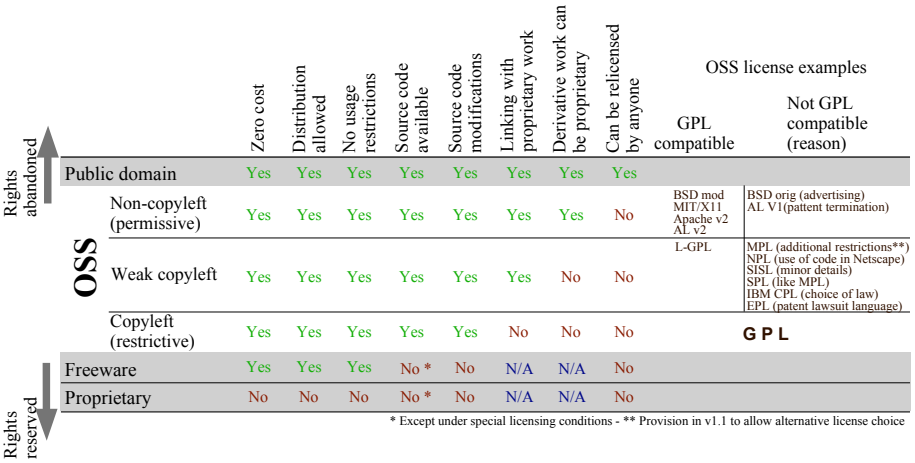
The FSF was founded in 1984 by Richard Stallman of the GNU project, and introduced the GNU General Public License (GPL), as well as the term copyleft. The FSF advocates that "free software is a matter of users' freedom to run, copy, distribute, study, change and improve the software". Indeed the main goal of the FSF is to keep software free by allowing others to build on one's code, and return their changes to the community [141, 83].

The term Open Source was coined in 1997 by Eric Raymond and Bruce Perens, who also wrote the Open Source Definition,[1] consisting of ten criteria for determining whether a license is open source or not [181]. The OSI was subsequently formed in 1998 as Netscape decided to release their web browser's source code to the public. This decision created concern among the developers' community as their creative work would circulate freely, and it was not yet clear what the term free meant. In an attempt to explain the concept, Stallman famously said they should "think of 'free' as in 'free speech', not as in 'free beer"' [83, Chapter 1].

6.3 License Types

Table 6.1 summarizes the main OSS licence categories, their relationship to other software license types, their main features and some representative examples that will be discussed in more detail.

[1] `http://www.opensource.org/docs/osd`



		Zero cost	Distribution allowed	No usage restrictions	Source code available	Source code modifications	Linking with proprietary work	Derivative work can be proprietary	Can be relicensed by anyone	OSS license examples: GPL compatible	OSS license examples: Not GPL compatible (reason)
Rights abandoned	Public domain	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
OSS	Non-copyleft (permissive)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	BSD mod MIT/X11 Apache v2 AL v2	BSD orig (advertising) AL V1(pattent termination)
OSS	Weak copyleft	Yes	Yes	Yes	Yes	Yes	Yes	No	No	L-GPL	MPL (additional restrictions**) NPL (use of code in Netscape) SISL (minor details) SPL (like MPL) IBM CPL (choice of law) EPL (patent lawsuit language)
OSS	Copyleft (restrictive)	Yes	Yes	Yes	Yes	Yes	No	No	No		**GPL**
Rights reserved	Freeware	Yes	Yes	Yes	No *	No	N/A	N/A	No		
Rights reserved	Proprietary	No	No	No	No *	No	N/A	N/A	No		

* Except under special licensing conditions - ** Provision in v1.1 to allow alternative license choice

Table 6.1 A categorisation of OSS license types, their main properties, and some characteristic examples. Based on material from [207, 249, 26, 203, 181]. Original source: [121] — ©2010 IEEE.

6.3.1 The GPL and Copyleft Licenses

The GNU[2] Public License (GPL) was created in the mid 1980s by Richard Stallman and its terms provided much of the foundation for free software development. A key feature of this license, which contributed to its widespread adoption, is the license's "viral" nature, as it enforces the source code of any derivative work from a GPL-licensed software to also be released under the GPL. As a result, developers working on GPL projects are assured that their code will never be used in proprietary software [94]. This is the essence of the notion of copyleft.

Furthermore, while the license allows the creation of derivative works, it does not allow the creation of derivative licenses from the GPL [140, 181]. The only stipulation on pricing is that anyone requesting the source code may be charged for the physical cost of the copy [94].

Large swaths of open source software to date have been distributed under the GPL [72]. Examples include the Linux operating system kernel, the GNU Emacs Editor and C Compiler, among many others. According to a 2005 study 68% and 69% of all projects hosted by Freshmeat.net and SourceForge.net respectively (two prominent online OSS repositories) were licensed

[2] GNU is a recursive acronym, standing for "GNU's Not Unix!".

under the GPL [26].

GPLv3, released in 2007, also inlcuded patent protection clauses. Until then, patent protection was only implicitly provided by the GPL. These new clauses were aimed at explicitly protecting OSS developers from the risk of being sued by companies distributing code under the GPL for patent infringements.

6.3.2 The Lesser-GPL and Other Weak Copyleft Licenses

The GNU Lesser General Public License (LGPL), [3] also known as the Library GPL, is a derivative of the GPL proposed by the FSF, intended for use mainly with software libraries. Its main differentiator from the GPL is that an unmodified LGPL licensed program or library can be incorporated within a proprietary program, or more generally one that is not licensed under the LGPL.

For example, if a library licensed under the GPL is incorporated into a proprietary program, and the two are distributed together, this would be a violation of the GPL as the distributed program and the library would be considered a derivative work subject to the limitations imposed by the GPL [140]. The goal of the LGPL is to overcome this obstacle.

This gives rise to the notion of weak copyleft, i.e. a less restrictive approach to licensing OSS. Weak copyleft permits the use of the covered code within larger works covered by other licenses. It therefore establishes a middle ground between the GPL license that does not allow this combination of licenses, and the more permissive non-copyleft licenses that freely permit this (see Section 6.3.3).

The FSF has developed the LGPL as a strategy to defend the ground of free software libraries against incursion by libraries licensed under less restrictive terms, such as the Apache license. By allowing their distribution with proprietary software, these libraries increase their chance of becoming widely adopted, and thereby furthering the FSF's goals. However, the FSF encourages the use of the more restrictive GPL license in cases where a library offers a unique advantage not found in competing libraries licensed with less restrictive licenses.[4]

[3] `http://www.gnu.org/copyleft/lesser.html`
[4] `http://www.gnu.org/licenses/why-not-lgpl.html`

The NPL/MPL Licenses , proposed by Netscape in 1998, also share the weak copyleft approach. Netscape's intention was to allow its source code to be used in larger, proprietary derivative work, but at the same time ensure that any modifications to their code would be contributed back to them and the rest of the community [94]. To that end, Netscape proposed a beta version of the Netscape Public License (NPL) for public comment and, based on feedback received, they refined it into a second license, the Mozilla Public License (MPL[5]) [140, 94].

Similar to the LGPL, the MPL allows the creation of larger, derivative work, including proprietary code that is not required to be published in source code form. Still, any changes to the original source code must be made available to the community [94]. Some particularities of the MPL license are discussed in Section 6.5.

The Artistic License The Artistic License (AL) was created by Larry Wall in 1991 for Perl, as he felt that the terms of the GPL (under which Perl was released until then) were too restrictive. The goal of the AL was to allow Perl to be used in commercial packages [94].

Its name is due to the intention to allow the initial developer to maintain "artistic" control over the licensed software and derivative works created from it [140]. Specifically, it allows the programmer to do anything they want as long as the changes are published and described in the source code, or all executables are renamed and the differences are documented, thus allowing the original author to maintain artistic control [94].

The AL is very similar to the GPL, but being a weak copyleft license it doesn't require distributing derivative works under the same terms [251].

Other Weak Copyleft Licenses Various other licenses have been proposed that embrace the weak copyleft approach, including the Sun Industry Standards Source License (SISSL),[6] the Sun Public License (SPL),[7] the IBM Common Public License(CPL) [8] and its derivative Eclipse Public License (EPL).[9]

[5] http://www.mozilla.org/mpl
[6] http://www.opensource.org/licenses/sisslpl.php
[7] http://java.sun.com/spl.html
[8] http://www.ibm.com/developerworks/library/os-cpl.html
[9] http://www.eclipse.org/legal/epl-v10.html

6.3.3 The BSD and Other no Copyleft Licenses

The BSD license was originally used for the release of significant portions of a Unix-related code by the University of California. Since then, a fair amount of open source software is distributed under this license. It allows covering derivative works under different terms or licenses, as long as the necessary credit is given to the original work.

Being one of the main no-copyleft licenses, it imposes no requirements on developers working with source code released under a BSD license. In contrast to weak copyright licenses, there are no incentives or requirements to contribute the modifications back to the community [94]. Originally it included a clause requiring that all acknowledgments of previous contributors' work be retained, however this clause was objected to and dropped in 1999.

Finally the BSD license also includes a no-endorsement clause saying that the names of the originators and contributors cannot be used to endorse products derived from the source code [94].

The Apache License is a derivative of the BSD license used by the Apache Software Foundation.[10] A rewriting of this gave rise to version 2.0 in 2004. It is very similar to the BSD and MIT/X11 licenses. It makes clear however, that the licensing of derivative works under other licenses is permitted so long as the terms of the Apache License v2.0, are complied with (this is implied but not specifically spelled out in the MIT and BSD Licenses).

The Apache License furthermore helps in the distinction between open source and closed source software development. Licensees are free to take their derivative work and license it under a different license. If, however they choose to label their addition to the work as a Contribution, then they are effectively agreeing to license it under the same terms that are applicable to the original work [140].

The MIT/X11 License is another no-copyleft license, which actually predates the BSD (it was written in 1987 for the X Window System source code). The two licenses are very similar with the main difference being that the MIT/X11 does not include the no-endorsement clause.

[10] `http://www.apache.org/licenses/`

6.3.4 Other Software Licenses

Licenses for Documentation Similar to OSS, licenses have also been created for technical documentation and publishing [140].

A typical example is the GNU Free Documentation License (FDL),[11] used for manuals, textbooks or other documents. It grants to everyone the freedom to copy and redistribute the material, with or without modifications, either commercially or non-commercially [83]. Another similar license is the Open Publication License (OPL).[12]

Creative Commons The Creative Commons Corporation[13] is a not-for-profit organization founded in 2001 and currently based at Stanford University Law School. It offers ways for authors to license their work openly [10]. It expands the open source model beyond software, to literature and the arts, and offers a variety of Creative Commons Licenses through which the authors effectively surrender most rights on their work.

These licenses are essentially a contractual undertaking between the creator and Creative Commons. Copyright is granted to the Creative Commons for 14 years and is renewable for one additional 14-year period [140].

Non-OSS Licenses In contrast to all OSS licenses discussed previously, the main characteristic of licenses that do not fall under the Open Source Definition is that there is no distinction between the distribution of original and derivative work [140].

The Sun Community Source License (SCSL)[14] is such an example developed by Sun, that tries to incorporate some of the benefits of OSS in proprietary products. A difference between this and OSS licenses is that Sun imposes a compatibility requirement, whereby any modifications to the licensed work must undergo a compliance certification by the licensor [140]. Furthermore, commercial use of code licensed under the SCSL may require royalty payment.

Another example, the Microsoft Shared Source Initiative[15] was created

[11] http://www.gnu.org/copyleft/fdl.html
[12] http://www.opencontent.org/openpub/
[13] http://creativecommons.org/
[14] http://java.sun.com/j2se/1.5.0/scsl_5.0-license.txt
[15] http://www.microsoft.com/resources/sharedsource/default.mspx

in 2001 when Microsoft provided limited access to some of its source code. It was critiqued for lacking the transparency and simplicity of open source licenses [140].

6.4 License Selection

Various considerations will affect the decision over what license to apply to an OSS project or program (reference [63] contains a concise guide).

First of all, it is generally advised to go with one of the existing and empirically tried licenses, rather than draft a new one [94, 73]. Using a well-known and trusted license will give the users confidence and clarity regarding what uses of the software are allowed. On the contrary an obscure, overly complicated and rarely used license will probably create confusion and ambiguity. Additionally, constructing a license from scratch requires a lot of experience and knowledge of legal matters, so it is not generally advised.

Furthermore, in various cases the choice of license may be limited by pre-existing software used in the project, and its own licensing scheme. For example, if pre-existing BSD-licensed software is used, the project team has the freedom to select any license, provided they respect any requirements regarding notifications and disclaimers. But if GPL-licensed software is used, then the only option would be to use the GPL for the resulting project as well.

Provided there is freedom of choice, the most important factor is the permissiveness of the license, i.e. to what extent it allows derivative work to be licensed under other schemes [140]. Other factors may include the following (see also the overview in reference [203]).

Topic and Audience It is argued that software aimed at developers, system administrators, or more generally technically proficient audiences, as well as projects on topics that target sophisticated peers, are more likely to be licensed under permissive licenses. The reasons include the strong community appeal of such software, as well as the fact that developers involved in these projects are often motivated by career enhancement opportunities, and will therefore be favorable to licenses that allow them to demonstrate their skills to a wide range of audiences, including users of commercial software.

Dependencies on Existing Work Compile time dependencies on existing projects may dictate the type of license a project will use. Projects having compile time (source code) dependencies on GPL software (for example, a driver in the Linux kernel) must adopt the license of the pre-existing software as they comprise derivative work. (Alspaugh et al. [3] provide a metamodel derived proof of this.) This requirement is mitigated if the existing software is a runtime dependency (i.e. the dependency may be loaded dynamically while the main program is running), in which case the derivative work clause may not apply.

Environment and Operating System Projects based on commercial platforms and operating systems are likely to employ more restrictive licenses.

Industrial Involvement If companies have a significant involvement in the project, then they are likely to be reluctant to adopt a strong copyleft license.

Commercialization Goals If the option of including the project's software in some commercial project is considered, then a non-GPL license that will permit this will be required.

Protection from Copying If a project feels that it needs to protect its code from other groups that may copy it and utilize it in their own products, then a license that would prevent this, or at least require that they return to the community their changes, would be preferable.

Attitude The degree of restrictiveness of the license may also depend on whether the developers and project communities believe in the right to redistribute one's work under licenses of their choice or in the goals of the free software movement.

Motivation Various studies [69, 203, 33] analyze theoretically and empirically how the developers' possible intrinsic and extrinsic motivations (e.g. the problem solving challenge, recognition by peers, monetary incentives and future employment) may affect their choice of license for their software.

It is generally found that more permissively licensed projects attract more highly skilled programmers who may want to maintain intellectual rights for their work, or simply for ego gratification, and stimulate more contributions. More restrictive licensed projects, on the other hand, ensure access to everyone's contributions and are favoured by communities with less free-riding [63]. For a broader discussion of motivation see Chapter 9.

6.5 Concerns and Risks

Various concerns have been voiced regarding the adoption of various OSS licensing schemes.

One such concern is that the use of licenses that allow the combination of open source and proprietary software effectively undermines the concepts that the OSS movement advocates [129]. Furthermore, it allows a competitor of a non copyleft licensed project to take the source code and build a proprietary, non open source product [63]. An example of this is Apple basing its Mac OS X on parts of the FreeBSD operating system.

Combining OSS released under different licenses also requires attention to compatibility issues [153, 251]. As a general rule, OSS released under diverse licenses can be combined to yield an outcome under a license at least as restrictive as the original ones. However there are exceptions to this rule, and the particularities of each license need to be carefully taken into account. For example software under the MPL license (see Section 6.3.2) cannot be redistributed under licenses that imposes restrictions not present in MPL. As a result, MPL software is, in principle, incompatible with GPL. However even in this case, a provision exists in the MPL license that allows a program or parts of it to offer a choice of another license as well,[16] thus partially overcoming this restriction.

Unlike proprietary software, projects under OSS licenses are, in various degrees, also unprotected from forking danger [140]. The GPL license hinders the danger of forking a proprietary project by enforcing that the derivative work will remain under the GPL. But under other licenses, and mainly non copyleft ones (such as BSD and Apache), the community is unprotected from developers forking off and continuing with non-OSS development. Often the

[16] `http://www.gnu.org/licenses/license-list.html#GPLIncompatibleLicenses`

community relies on the reputation of the developers for avoiding this, as well as on other measures discussed in Section 4.2.

Commercial software development firms may feel that there is a risk involved in incorporating OSS code in their products, due to the lack of clarity in some definitions [25, 26]. As discussed previously, some OSS licenses only allow the reuse of software if the derived work is also under the same license (most notably GPL). But the definition of derived work may not be clear enough to dictate how the original OSS software could be used.

For example, it has been suggested by some that if a GPL-licensed program has a runtime dependency to a proprietary library (or vice-versa), the result would not need to be licensed under the GPL, as the two programs retain distinct existences. However the FSF does not accept this position, and has argued that in such cases the LGPL should be used instead [26].

Possible strategies around this may include clearly separating, at the architectural level, the pieces of the resulting software product that rely on OSS code from other parts, and license the former as OSS and the latter as proprietary [103, 196]. But in any event, the use of legal advice would be recommended [25], or alternatively "packaging companies" can be employed to serve as intermediaries between the OSS community and the proprietary software house to undertake various technical responsibilities, as well as legal, licensing and intellectual-property rights issues [196]. Alternatively, permission may be explicitly requested from the OSS project owners to include part of the code within a proprietary product [103].

7

Business Models

An organisation's decision to move into the OSS domain must be based not only on technological and social considerations, but also on an evaluation of the business perspective and impact of such a decision.

The OSS business model, and in particular the revenue logic behind developing and distributing OSS, is not one of the most obvious to perceive. However, as various business model analysis frameworks indicate [166], the revenue logic is just one part of the entire picture. Product and business strategy, including the types of services offered, the development of core competences and competitive advantages, the company's market approach, the creation of value chain positions, or the exploitation of specific customer communities, are just as important. A move into OSS can therefore be considered as a strategic maneuver rather than just the formation of a new revenue stream [187].

In this chapter we first focus on the diverse types of strategic advantages that adopting OSS can offer, and the impact these may have at the business level. We then discuss what prerequisites are necessary to make such a move successfully, and some of the concerns that should be taken into account and weighed. We describe specific elements and characteristics of different OSS business models, and the ecosystems of companies, organizations and other players that are formed around them.

7.1 Strategic Advantages and Impact of Moving to OSS

The adoption of OSS practices can offer various strategic advantages, and it can impact a company or organization's business model in various ways.

The move from selling commercial software to distributing OSS may be performed partially, or in steps. Possible strategies include offering source code with a closed license that has an expiration date [187], or converting previous versions of a product to OSS while selling the latest version as closed source [94].

As Joel West found in three case studies [248], the software can be offered as a partly open solution, thus providing value to the customers but making it difficult to be directly exploited by competitors. One way of achieving this is by using a restrictive license. Alternatively only certain parts of the software product can be offered as OSS, while retaining full control of the most critical layers (an example of this is Apple's Mac OS X operating system).

The degree and way in which a software vendor may decide to open source their offering may depend on its placement within the market and with respect to other products. Wijnen-Meijer and Batenburg found through literature reviews, interviews and surveys that a product with a large market share leading an ecosystem of complementary products, or a product with distinct technical capabilities, is less likely to be open-sourced [250].

Table 7.1 summarizes the business and strategic advantages of moving into OSS, which are discussed in more detail below. We group these into three categories: Advantages resulting from the user base and community that is formed around OSS products; advantages resulting from the special market and competition placement offered by OSS; and advantages more directly related to production costs and revenue streams.

7.1.1 User Base and Community

There are cases where large user communities can be quickly built by converting to OSS, often with minimal sales and marketing expenses [243]. Existing markets can be broken into or reshaped [187, 77], and significant market shares can be acquired . An example is the Netscape company, which opened the source code of the Netscape browser to increase its user base with respect to its competitor products.

User base and community	**Market placement and competition**	**Revenue stream and financials**
User base development	Approach restricted markets	Enable new services
Information about market	Increase reputation	Increase demand for complementary services
Innovation dissemination	Attack competitors	
Productivity increase	Preempt development of closed standards	Reduce development costs
Access to customer needs		Lower break-even points
External developers use	Embrace underdog mentality	Introduce new revenue streams
Access to new skills and practices	Escape from vendor lock-in	

Table 7.1 Summary of business and strategic advantages of moving into OSS.

Through OSS development, information is collected dynamically about products, services, customer needs and ultimately the market itself [20, 77]. Joachim Henkel, based on large-scale surveys and interviews, also found that OSS development also creates opportunities for setting industry standards and enabling compatibility with other products or systems [107].

At the same time, it also offers a powerful means of disseminating innovation and research results throughout the community [77, 250]. Making the source code of a product available is expected to lead to greater innovation, provided that a critical mass of developers will find interest in it and will be attracted to it [135].

Development productivity can be significantly increased by leveraging the talent and expertise found within the OSS community, as discussed through the OSS product and process transformation framework proposed in [72].

The close interaction of the OSS development process with the user and customer base often allows the customer needs to be taken into account in the design and customisation processes [20]. This gives a competitive edge with respect to the proprietary software development approach.

Finally a move into OSS allows small companies that employ a minimal number of developers to benefit from a large pool of external developers and their technical skills and expertise, to participate in practices such as peer reviewing that result in better software products, and to be exposed to innovation that they could not otherwise afford to create internally [20, 230, 106].

7.1.2 Market Placement and Competition

OSS can be an effective way of approaching restricted or limited commu-

nities where traditional market strategies do not work [77]. Customisations and adaptations to the particular needs of these niche markets offer additional revenue streams.

Furthermore, the OSS approach can be used to attack competitors by offering similar products at substantially lower cost (or completely free). A typical example of this is the OpenOffice suite[1] [77]. Joel West, in his three case studies we mentioned earlier in this section, found that this practice can also preempt the development of closed proprietary standards by rivals [248]

Often OSS products compete with non-OSS products offering similar functionalities and solutions. The non-OSS products have the advantage of more resources, advertising and public relations, but an "underdog mentality", as well as help from the developer community can help the OSS product to a considerable degree [135].

Finally customers also appreciate the fact that with OSS they are not subject to "vendor lock-in". This was found to positively affect user and customer loyalty both in the literature and through surveys and interviews [250].

7.1.3 Revenue Stream and Financials

When open-sourcing a proprietary software product, there are indirect effects that include an increased demand for other products and services that complement or support the main offering. This can lead to an improvement of a company's profitability, market placement and reputation due to better quality, support, and customization possibilities [250].

The ratio between fixed and total development costs is reduced, lowering break-even points for new ventures and reducing overall risk [20]. The overall development cost is also decreased [250, 22, 134, 12, 235, 191, 60] through the distributed processes that involve external development support (see also Section 5).

7.2 Prerequisites, Deciding Factors and Concerns

The move from closed to OSS distribution requires careful consideration. Diverse software products, companies, or markets require corresponding approaches, and it may be that in some cases opening the software code or

[1] http://www.oracle.com/us/products/applications/open-office/index.html

licensing structure may not be a wise move.

For example, software products focusing on cost leadership, or with a horizontal functional scope, are better candidates for moving to OSS than those offering added value through their sophistication or advanced features [250]. Software with a broad, horizontal scope that is geared toward a mass market is more likely to attract attention and outside developers, and is thus better suited for open-sourcing [43].

The market position of a software product is also an important factor in this decision. For example, open-sourcing a product that has a large market share, that is at the forefront of the state-of-the-art in terms of technical capabilities with respect to its competition, or that has an ecosystem of other products and services depending on it, is unlikely to lead to any gain for its vendor [250].

The decision on whether to open-source a product should also include the following stages and considerations.

Evaluate the Market for the Target Product Consider both commercial and other OSS offerings, or combinations of the two. Determine whether there is market interest for the product to become open-source [9].

Determine Development Community Interest Consider how likely is it for a community of developers to form around the product and provide their skills, expertise and development effort once it is open-sourced. Forums, mailing lists and other communication channels can offer such insight [9].

Decide what Parts of the Product to Open-Source It is possible to only open a part of the product source code, and keep the rest proprietary. Reasons may be trade secrets or algorithms that are better not publicized, part of the source being shared with other products, and dependence on third party technologies with different licensing schemes [103].

Balance Short Term Switching Costs Technical switching costs may involve backward-compatibility of the new OSS versions, or customers being unwilling to embark in the new OSS direction, and instead switch to other products [20]. Additionally, new personnel may need to be hired, and if there

is not enough familiarity with OSS practices it may be required to outsource part of the processes such as installation, configuration, and maintenance [230].

Consider New Processes, Infrastructure and Environment Forming an OSS project and community will change the way a company or organisation does software development. It will require specific technical infrastructure (such as distributed revision control systems and an openy accessible issue-tracking system — see Section 5.6), processes (see Chapter 5), and an appropriate environment supporting issues such as a formation of an open, collaborative community, free information flow, new governance models and managerial skills, labor division, and support for a geographically distributed team [243, 248, 230, 106, 107].

Accessing the OSS community, managing OSS information that is available on the internet, and dealing with licensing issues and customers, are additional organisational routines that will need to be supported [20]. Before embarking in the OSS direction it is important to verify that these elements can be developed and supported.

Ensure the Correct Mentality is Present OSS development requires a particular mentality and culture that may not be present within the company, and will need to be developed. There may be resistance by non-OSS ideologists [107]. The culture of respect for developers' intellectual rights, as well as the need to match their skills and capabilities with the project requirements may be missing, as evidenced through surveys and interviews [230, 248].

Ways to overcome these issues may include participation in other OSS projects or events, selection of licensing that is most likely to attract a development community, or allow company employees to get involved in other OSS projects before making the switch [145, 44].

Tolerance for the inevitable free riding, as well as the understanding that by opening one's code certain advantages will be offered to competitors will also need to be developed.

Sanitize Code Though it may seem a secondary, technical step, making sure that the code is ready for public distribution can be a daunt-

ing task involving rewriting or adding comments and documentation, re-implementation of certain functionalities in better ways, and removal of parts that are only intended for internal viewing [103]. This is important, as the source code is likely to be scrutinized (even by automated tools), and will form part of the company's new image.

Select Appropriate Business Model As described in Section 7.4, various business models can be adopted and the most appropriate one may depend on a wide range of considerations.

Select Appropriate License As discussed in detail in Section 6, there are many licensing approaches, with varying degrees of permissiveness as well as other characteristics.

Decide on Marketing Approach Building awareness for an OSS product can be challenging, as new channels of communication will need to be tapped into or created, and new communities of users will need to be approached.

7.3 The Open Source Software Ecosystem

Business models based on OSS may involve many industry players cooperating at diverse roles to form an ecosystem. We briefly introduce the types of companies, groups or organizations that can be involved in such an ecosystem, and then examine the approaches through which profitable business can be based on these cooperations.

OSS **developers and project communities** form a diverse group of people with a shared interest and passion in both a specific project or product, and in the concept of making it open to the community so that anyone can make improvements or add functionality to it. They are organized in a community that is formed around the project, either as independent individual developers, or within corporate boundaries.

Software distributors focus on the business of system integration, packaging, quality assurance and services [187]. Typical examples are the various Linux operating system distributors (notably RedHat and SuSE). Their role in

OSS is important, as they package the software into different distributions, enhance it with middleware applications, and offer technical support and other value-added services such as training for specific tasks. They gain revenue due to the large number of transactions they are involved in, and they also gain reputation from their participation in the OSS movement.

Software producers and vendors can benefit from OSS by incorporating OSS into their product offerings. They can either use existing source code within products they develop, or they can adopt entire OSS products and include them in their list of offerings, subject to licensing restrictions. In any event, they lower their total production costs. Additionally, they can offer complementary services and technical support to users of third-party OSS products (similar to what distributors do). Finally they can also choose to open the source of (part of) their own products (e.g. Sun and Java, Netscape and Mozilla). Such a move has been found to result in many benefits including increase in revenues and/or reputation. Their choice of license depends on their business development strategy.

Hardware producers and vendors can incorporate OSS, such as drivers and applications, to support their hardware (examples include IBM and HP). They can also use OSS in embedded software platforms, such as set-top boxes, broadband routers, mobile phones, and GPS navigators (the TiVo digital video recorder, and Android-based mobile phones are two notable examples).

Third party service providers provide technical support, assistance, and value-added services, similar to (and sometimes in competition with) projects and distributors [135].

End users of the software products are generally categorized as home or enterprise users, the latter usually being more willing to pay for detailed product documentation and value added services such as technical support [43].

Others business types may be involved in the OSS ecosystem, including companies producing accessories to be marketed along with OSS software

products to the OSS community, or other types of organizations with a belief or stake in the OSS paradigm that may wish to offer support or sponsorships.

7.4 Main Business Models

In the following paragraphs we describe the main OSS-related business models encountered within the OSS ecosystem. Figure 7.1 summarizes the types of players that are most usually involved.

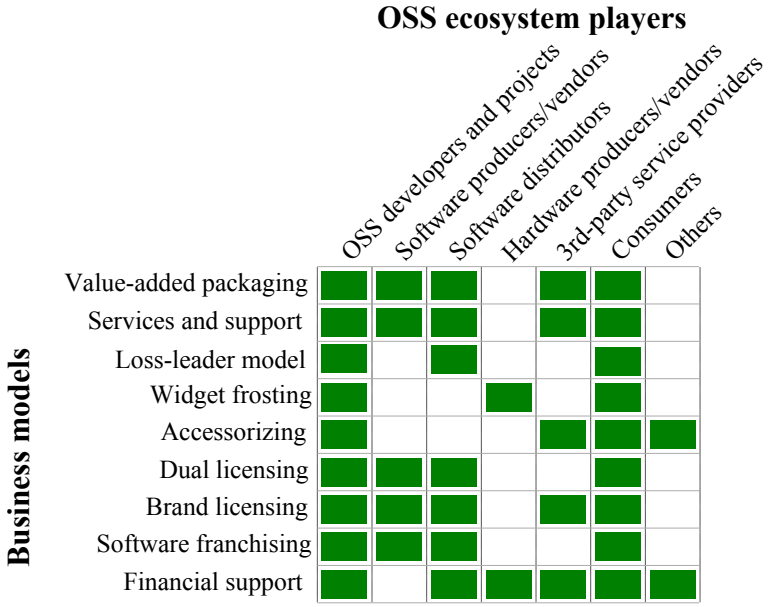


Fig. 7.1 The typical OSS business models, and the OSS ecosystem players that are usually involved in each.

7.4.1 Value-Added Packaging

A variety of value-added products and services can be bundled together with the core OSS product [94]. Typical services may include system installation and integration, technical support, while commercial utilities and applications can also be packaged [44]. A typical example company in this role is RedHat, which facilitates the complex task of installing and configuring the diverse components of the GNU/Linux operating system [72].

Technical support, customization and upgrade services are mostly targeted toward enterprise customers, and may include long-term agreements [135].

Software version support services may include identifying and providing the most recent, stable and safe version of a certain OSS product, as well as offering special premium or advanced versions through some subscription mechanism [134, 135].

Finally, packaging may also be offered in terms of physical distribution and delivery of the OSS product, e.g. in CD-ROMs and printed documentation sent over regular mail [135, 245].

7.4.2 Services and Support

Services and support forms a similar business model to value-added packaging, but is targeted toward more independent services and support-based solutions.

A subscription-based model can offer users the ability to manually check for software updates and new releases, access to discussion forums for technical support, as well as access to paid consultants and contractors to help with specific tasks [243]. Examples include the SuSE (acquired by Novell) Linux Enterprise Desktop, and RedHat with the JBoss application server. This model provides a predictable revenue stream for the providers, and the option to engage such services only when necessary for the customers. Post-sales training and support can also be provided, along with additional documentation and manuals [243, 134, 154].

Finally independent consulting services can be offered regarding the strategic aspects of decisions and investments related to OSS [94, 243]. Sometimes the software providers also undertake this consultant role, benefiting from their reputation to offer these services and increase their revenue [250].

7.4.3 Loss-Leader Model

In the so called “loss-leader” model, some software offering is distributed freely as open source, in order to attract interest and fuel demand for some other, linked, proprietary software. This practice creates a community of developers and users around the product and increases the vendor’s reputation [245].

The proprietary software may be an advanced version of the OSS product (e.g. the open sourced Sendmail versus the proprietary Sendmail Pro), or it may be a set of additional related functionalities and products (e.g. GUIs, toolkits, frameworks and languages offered on top of open-source integrated development environments) [250].

7.4.4 Dual Licensing

As discussed previously, various vendors allow their customers to select what license they want to apply to the use of their software. A free, OSS license such as GPL is usually offered for non-commercial applications, and a proprietary license for commercial ones [243]. Alternatively, extensions of a certain OSS offering may be covered by a non-OSS license, at a cost [250, 243, 94, 135].

Commercial versions of OSS applications may include additional features and capabilities. The advantages of the company and product's presence in the OSS community are used to boost the sales of the commercial versions.

Examples of dual licensed software include Qt[2] and MySQL.[3]

7.4.5 Widget Frosting

Widget frosting is a term for embedding OSS software into hardware products [243, 94], such as a kernel, printer drivers, compilers, operating systems or applications [245]. This offers the hardware vendor all the gains of OSS development (large developer pool, more customer involvement, peer review and other reliability measures and, possibly, also increased customer loyalty) [187]. Software licensing costs are also reduced [250]. Examples of this strategy include the TiVo set top box, which runs a Linux kernel [8], and the One Laptop Per Child Foundation's XO laptop, which is based on the Fedora GNU/Linux distribution [243].

7.4.6 Brand Licensing

In brand licensing, one company charges other companies for the right to use their brand names and trademarks. The brand reputation thus gained in creat-

[2] `http://qt.nokia.com`
[3] `http://www.mysql.com`

ing a successful OSS product is sold to other companies that create derivative products [154]. For example, although Sun (now Oracle) released a GPL-licensed implementation of the Java platform, the Open Java Development Kit, in 2006, it retained control of the Java trademark certifying implementation suites as fully compatible with its specification. It therefore used the Java brand's power to align other companies implementations with the "write once, run anywhere" strategy.

7.4.7 Accessorising

A variety of physical accessories accompany OSS software, ranging from books, manuals and documentation, to more ancillary items such as T-shirts, mugs, and stickers (fuelled by the particular culture surrounding the OSS development community). Companies obtain considerable revenue from the sale of these items [187, 94, 243, 245]. A notable example is the O'Reilly publishing company, which offers hundreds of titles documenting OSS software, and even hosts OSS-themed conferences, including the influential OSCON Open Source Convention.

7.4.8 Financial Support and Coexistence

Although this may not qualify as a business model in the strict sense, OSS projects are often supported by donations from other companies that have adopted their products. Additionally, foundations such as the FSF may support either OSS projects or directly their programmers [77].

Corporations also directly sponsor OSS projects, either in funding or by contributing developers to work on the projects, or by releasing previously closed code and encouraging their employees to work on it. An example is IBM's Eclipse software development environment, which is still supported by IBM developers [244].

Finally, venture capital funds also exhibit considerable interest in OSS projects, especially following success stories such as Red Hat, Netscape and others [103].

8

Adoption and Reuse

The licensing of open source software and the availability of its source code makes it a very good reuse candidate in other software development efforts. Indeed this is a regularly witnessed trend with the documentation of both proprietary and open source software regularly containing acknowledgements for a multitude of reused libraries and components. We discuss the different types of adoption and reuse, and reasons, concerns and criteria for effective and successful reuse.

8.1 Adoption vs Reuse

We distinguish two levels of OSS utilisation within other organizations and software development efforts: adoption and reuse.

8.1.1 Adoption

Adoption is a general term that refers to the strategic decision by a company or organization to utilise OSS software, either by introducing OSS products in its daily tasks, or by reusing (parts of) OSS software within its own products or packages.

In the first case it is not necessary for the organization to open, read,

and modify the code, because the organization only seeks to improve the performance of its internal function by using open source software instead of proprietary offerings (e.g. GNU/Linux vs Windows, OpenOffice vs Microsoft Office, Inkscape vs Adobe Illustrator, or PortgreSQL vs Oracle's relational database).

In the second case, software companies reuse the code and the artifacts of an open source software package in order to build a new product. We refer to this with the more specific term of OSS reuse.

8.1.2 Reuse

Reuse refers to a more specific case of adoption, where a piece of OSS is incorporated in a software vendor's product, either as it is or after modifications are made to it. So OSS reuse is a more organic approach to adoption, whereby the OSS software being adopted is studied, understood, and embedded in the new product. A goal of software reuse is to reduce the amount of new software development [123, 111, 31].

The advantages and disadvantages of software adoption and reuse must be considered before deciding whether to undertake this task [230], as discussed in Section 8.2.

Software reuse has been examined and defined clearly within the software engineering bibliography. In the case of open source software it usually refers to the use of open source code within proprietary or other open source software.

In terms of source code, the purpose of reuse can be either the incorporation of the code in new software, or its use as a reference [228]. The size and granularity of the reused software can vary from lines of code to functions, modules, or entire systems and applications.

We can categorize OSS reuse into three types, according to the degree to which the originating software is used, as-is, studied or modified.

Black-Box Reuse In this case the original software is used as-is, or with very minor modifications [2, 230]. This may or may not include source code, and it is similar to the reuse of other proprietary software. The advantages of OSS in this case include the zero purchasing and royalty cost, the lower risk of vendor lock-in, and the ability to migrate to other, more intrusive reuse

models, should such a need arise. An example of such reuse are the various applications distributed with Linux-based netbooks.

Grey-Box Reuse Modifications of the code only take place at interface level [216, 2], and may not involve but a small portion of the source code. However a study and in-depth understanding of the code may be required, which may involve a considerable investment, even for limited modifications [230]. Continuing the previous example, some netbooks ship with a Linux version enhanced with modules that support their particular hardware features.

White-Box Reuse The inner workings of the reused software are studied, and adaptations, customizations and bug fixes take place [230]. Notable examples of this reuse type are Juniper's use of the FreeBSD kernel [151], and Google's use of the Linux kernel [87].

8.2 Criteria for Reuse

Before deciding to reuse OSS software within a proprietary software product a careful consideration and study of the circumstances should be carried out, with focus on the characteristics of the organization that is considering to reuse the OSS code, as well as the OSS code itself. Criteria that should be considered emerge both from theoretical and empirical studies, and include the following.

Organizational Versatility involves experience and know-how with OSS software reuse [2, 25], existence of skilled IT personnel [91, 50], the level of management support, formalization and strategic planning [2, 91, 216], an ideological predisposition toward OSS [91, 2] and availability of technical support [230].

Adequate Tool Support includes tools such as operating systems, databases, application servers, compilers, build managers and integrated development environments, and version control management systems compatible with the OSS software to be adopted (see also [216, 25, 2, 111]). Most

tools used for developing OSS are themselves OSS, so the main difficulty here is organizational resistance toward their adoption.

Appropriate Development Standards Appropriate software engineering and development standards and practices should be followed to ensure safe and effective integration of the reused code into an application. These may include visibility and accessibility of the code repository, use of version control systems, peer reviews, issue tracking, regression testing, use of various code quality metrics, training into new OSS practices, and the use of OSS development tools (see also [216, 25, 2]). The aim of adopting these standards is to ensure that modifications of the reused source code can be smoothly integrated with the original source. Failure to back-port these changes creates a fork between the reused code and the original source code version, which increases the maintenance cost of merging the two brnaches over time.

Status of the OSS Code to be Reused The status of the code must be carefully assessed in terms of aspects such as

- reliability, maturity and robustness [242, 31, 2, 230],
- maintainability [31, 20],
- compatibility with the existing technology and skills [50],
- scalability [242] (to the extent that this is important),
- portability [111],
- functionality, with respect to the new product needs [111],
- security, availability and robustness [111],
- flexibility of interfaces and ability to upgrade [111, 31, 20],
- ease of installation and upgrade [196, 2, 111],
- interoperability and ability to run on other operating systems or older hardware [91], and
- legality with respect to licensing and potential for patent infringement [111].

8.3 Adoption Drivers

We distinguish and analyse four key drivers behind the adoption and reuse of OSS: strategic advantages, enhancement of development process, product

quality, and external factors.

These are discussed below, and also itemized in Table 8.1.

	Adoption drivers	Concerns
Strategic advantages	Reduced development cost No proprietary lock-in Reduced time to market External support Business opportunities	Switching costs Locating the right OSS Licensing and legal responsibilities OSS lock-in possible
Development process	Dissatisfaction with other products Good OSS development practices Community support	Non-standard processes, knowledge barriers Requires high expertise Code coupling and interconnection Orphaned code No time to market pressure
Product quality	Functionality maintainability reliability efficiency portability	Quality variability poor documentation Code ownership and accountability Code quality assessment
External factors	Successful projects Need for transparency	Cost of experienced personnel Incentives for developers

Table 8.1 Motivation and concerns regarding the reuse of OSS within proprietary software products.

8.3.1 Strategic Advantages

The strategic advantages behind software adoption and reuse emerge from various empirical studies including interviews, field studies and surveys, as well as theoretical approaches and literature reviews, and include the following.

Reduced Development Cost and No Licensing Fees OSS is a cheaper solution in terms of direct acquisition, upgrades and maintenance costs, and is a substitute for the development of new software [230, 2, 164, 50, 153, 242, 196, 28].

Avoidance of Proprietary Lock-In As the source code is available, there is no risk of being locked into proprietary standards or specific software or

hardware platforms, and dependence on particular software vendors is minimized [242, 20, 230, 28].

Reduced Time to Market Reusing software components significantly reduces the overall development time [169, 98, 28]. This has also been empirically verified in [2].

Availability of External Support Support can come in diverse forms and at various levels, through the OSS project community or through support contracts for advanced versions of OSS products [230] (e.g. enterprise Linux distributions).

New Business Opportunities Many companies have embraced OSS and built business models around offering value-added services based on such software [169].

8.3.2 Development Process Enhancement

The adoption of OSS affects and enhances the entire development process in various ways.

By incorporating OSS source code, as well as the valuable coding and development practices of well-engineered OSS, including bug fixing [2], peer reviews [196, 216] and collaboration procedures [164], a mediocre development process can be greatly improved (Reference [216] advances this view with examples from prominent OSS projects.)

Additionally, by reusing code of choice, developers can work with code they appreciate and feel comfortable with [98]. In fact many developers that reuse OSS end up becoming contributors to the original OSS project from which the reused code originated.

Furthermore, the communities formed around OSS projects can be leveraged in the process of adopting and/or reusing the project's outcome [25]. The knowledge and experience of the community is transfered to the commercial setting within which the new product is developed. Additionally, the OSS community can be attracted to the new development effort, to participate in discussions or even in the code development itself with its large developer base [164], thus maintaining the product up-to-date.

Finally the community may even be targeted as potential clients of the resulting commercial product.

8.3.3 Product Quality

As OSS projects often go through an extensive period of maturation before being a target for adoption [247], the code originating from them can be of particularly high quality, thus giving additional value to the proprietary product. Specific characteristics of interest include the following.

Advanced Functionality Often, the functionality provided by reused components is more complete than what could be achieved by a single company's development effort. Having access to the source code allows additional requirements to be fulfilled [91].

Code Security The open availability of source code allows us to examine it and ensure that there are no dangerous bugs, viruses, or other security holes, and modify it if necessary [164, 230]. It is found that security breaches in OSS code are usually fixed very quickly [196].

The peer review process offers additional reassurance, as do the recognized good practices usually followed in OSS development [2].

Customization The ability to customize OSS offers additional flexibility and allows changes, experimentations and freedom of choice [164]. Code that meets specific requirements can be reused and adapted to achieve additional customer satisfaction [242, 2, 169]. The resulting software can then be redistributed within the OSS community for additional cycles of testing and feedback, thus improving it at a minimal cost and resulting in better and more trustworthy software [242, 230, 98].

Modularity and Granularity In OSS development modularity and granularity concerns are given a high priority. As a result it is also possible to reuse code at different levels of granularity ranging from lines of code to classes, methods, libraries or entire system, thus increasing the opportunities for reuse [2, 216].

8.3.4 External Factors

Additional external factors that may favor the reuse of OSS in proprietary products include the showcase of successful publicly recognized exemplars of software development within one's product [91], and a need for transparency, publicity of processes, and security.

8.4 Concerns

Various concerns are involved with the decision to adopt and/or reuse OSS within an organisation. We group these into the same four categories. (See also Table 8.1 and the survey in [222]).

8.4.1 Strategic

Adoption of OSS software might incur short-term switching costs, often requiring a substantial investment. For example using a Linux-based operating system for the first time may involve a steep learning curve [196] and a considerable application porting effort.

Searching, analysing and integrating an existing piece of OSS code may actually turn out to be an equivalent task as writing it from scratch [98], so the total cost of reusing code must be carefully calculated [230]. The amount of experience with OSS within the organization, including both technical and organizational skills will play an important role [20, 50].

Additionally, the software itself may incur costs, e.g. enterprise Linux distributions that include additional services and technical support are not free, nor are external consultants that may need to be hired [230, 241].

A category of additional hidden costs must also be anticipated, including initial procurement and maintenance [111], customization and integration [153], personnel retraining [20, 230], data migration to the new system environment and file conversion [230, 20] as well as consultants cost [169]. A careful consideration of all these costs projected over a reasonable period of time must be done to decide whether OSS reuse is actually in the best interest of the organization.

The complexity and sheer number of different OSS licenses also present complications and perceived risk factors, as discussed in Chapter 6. Restrictions imposed in a license may affect the ability of the proprietary software

to be extended, internally reused, or resold [241].

The organization adopting the OSS is legally responsible for ensuring the license requirements are met, but may not have the technical expertise for this [105]. It is therefore exposed to the risk of the owner of the original OSS claiming for damages or, more commonly, demands for burdensome compliance measures, such as the publication of proprietary code incorporated within a GPL-licensed product.

Another legal risk is the inadverted infringement of third-party patents or intellectual rights, which may be included in the OSS code that is reused [196]. The OSS licenses generally don't provide indemnity from such claims. Dual-licensed OSS may help overcome some of the above issues, but it will require purchase of a license for use in proprietary software [230].

Finally, although an advantage of OSS is that it offers protection from proprietary vendor lock-in, it is still possible to get locked-in with an OSS software choice [230]. For example, an organization basing its product on an enterprise Linux distribution and the relevant services, support, and updates, may have difficulty switching to a different solution.

8.4.2 Development Process

In terms of the actual development processes followed by the proprietary company or organization, OSS adoption or reuse may also be subject to various issues.

The adoption and reuse procedures are not standardized [169]. Rigorous processes should be followed, including conscious searches in online OSS repositories, specific training of staff members or even hiring of new staff with experience in OSS, and the outsourcing part of the adoption process. These are often neglected.

It is also argued that as OSS projects pay less importance to strategic planning and organization of the overall development effort [216] some of the important project life-cycle tasks may be downgraded or skipped. The availability of support for the reuse and integration tasks may often not be correctly organized and available [169].

Another issue with the reuse of OSS is that modifying the source code of important software such as operating systems or applications servers requires particularly mature and experienced programmers [230]. If problems come

up with the integration (e.g. the coupling and interconnection of different pieces of reused code may have undesired and unexpected side-effects, if not done carefully [216]) the response time and the organization's capacity of dealing with them may be questionable. This is supported by the work of Meng Huang et al. who designed an assessment framework and reported on a case study based on it [111].

Establishing a long term relationship with the OSS community behind the code that is adopted is definitely an important factor. Even so, as found by an exploratory empirical study in [164], managers feel that in case issues arise there is no safe mechanism or entity to undertake the task of resolving them. It is possible, for example, that a piece of OSS that is reused ends up being abandoned or "orphaned" by the original project (sometimes a result of forking [169]). In this case, the code will not be properly maintained and will not evolve, or if it does it may not be in a direction compatible with the rest of the project [216].

Finally, a critical difference between OSS and proprietary software development, as also discussed in Chapter 5, is that with OSS the time to market is not an important issue. As a result, a proprietary software release may need to be indefinitely delayed until an important upgrade or fix is completed within the original OSS project whose code is reused.

Overall, there are still knowledge barriers to the safe reuse of OSS code. These affect issues such as searching for the appropriate OSS application whose code to reuse [169], assessing the maturity of an OSS package [230], transfer of knowledge and skills for bug fixes and modifications, the lack of ability to adapt such code for legacy systems and others. Still, Ajila and Wu, through questionnaires, surveys and a literature review, found that these issues are not particularly more severe in the reuse of OSS compared to software reuse in general [2].

8.4.3 Product Quality

Regarding the quality and condition of the reused product itself, there are concerns about OSS documentation [164], which may not meet the standards of the rest of the proprietary code.

Open source software quality is considered to be often of a high standard, but there can always be exceptions. It may not be easy to assess which is

the case [216], or to verify that there are no hidden security flaws in large code bases, thus increasing the risk of security problems [25], and this has significantly deterred companies from reusing software, or only do so after extensive code review processes or if a close relationship is established with the originating OSS project.

Additionally, as mentioned in Section 8.4.2, orphaned code will not be correctly maintained, and thus will be of poorer quality [216].

Finally, the ownership of the code being reused may sometimes be difficult to determine, especially if it is licensed under a viral model [216], making it difficult to hold someone responsible or accountable for it. Furthermore, even though newer SCM systems do permit authorship identification at the source code line level, most OSS licenses include limitation of developer liability clauses.

8.4.4 External Issues

Other external issues that may adversely affect the decision to reuse OSS include the fact that in some cases there is not enough funding to get the necessary support for adapting the OSS to the needs of the proprietary product [50], as well as the need to develop incentives for the developers to engage in such a process [98]. In some cases the reuse of code is considered by developers to be less rewarding than writing in new code.

8.5 Software Reuse Process

The process of reusing a piece of OSS within a proprietary software product consists of the following four stages: decision, selection, integration and assessment.

The factors that should be considered in the decision to reuse OSS have been analysed in the above sections. Considerable additional insight is offered in [159], which outlines a particular case of OSS development and reuse within HP, highlighting some of the relevant concerns and how they were dealt with and overcome.

In the selection stage the appropriate software to be reused is identified through the following steps.

(1) Collection of high-level requirements and identification of candi-

date projects [111, 123]. This happens through an analysis of the functional and non-functional requirements for the specific software, and a comparison with various existing OSS projects. Other types of requirements may include cost, hardware constraints, project status, maturity and popularity, programming language and operating system (see also Chapter 3 for indicators of successful projects). Then deeper surveys involving the OSS project repositories and community will yield additional details about the project quality, size, number of developers, submission and bug fix rate [153].

(2) The high level architecture and modularity of the candidate projects is studied, to identify in which cases the particular parts of interest can be effectively extracted for reuse [111].

(3) More detailed and low-level specifications are then laid out, together with specific selection criteria [111, 123, 153]. Additionally, licensing concerns and interactions should be carefully considered [153], and the interdependencies between software licensed under different license types should be taken into account (see also [121]).

(4) A final selection of the OSS project, and in particular of what parts of it will be reused eventually takes place [31, 228, 153].

In the integration and implementation stage, the following four steps are involved.

(1) The initial criteria and requirements are adjusted, if required, based on the characteristics of the OSS to be reused.

(2) The type and scope of reuse are determined, including the degree to which the OSS code will be modified (e.g. a selection between black, grey or white-box reuse). This decision should take into account the risks involved and the match between the requirements and the OSS specifications [123].

(3) The next step consists of implementation of changes, improvements and integration of the OSS code within the product [111, 31]. According to the type of reuse there may be a need to study and analyse the source code, and perform code modifications of

different extent. In the case where legacy systems are involved, experienced consultants may be required to perform specific code adaptations [169].

(4) Finally the resulting software product will be subject to proper maintenance and updates, including modifications in system functionality, keeping the relevant documentation up to date, debugging, restructuring, and conversion [153]. An important part of this phase involves the contribution of any changes made to the reused OSS back to its originating project. This integration effort, apart from being the correct ethical choice, will also reduce the effort of future maintenance.

At the end of this process the product status should be assessed, including rigorous testing of the resulting artifact, evaluation of the entire process, its cost and potential need for extensions, and validation of the license status [153, 123].

9

Motivation

One of the aspects of OSS that attracts a lot of attention is the fact that it is a high quality public good developed for free by qualified volunteer developers, individuals or organizations, and evolving at a rapid pace. This is puzzling, because based on economic theory one would expect such privately provided public goods to be stagnant and of inferior quality [16].

In an attempt to explain the above phenomenon, the motivation for contributing to OSS projects has been the subject of many studies from diverse scientific domains. In this chapter we try to summarize their findings and present some of the relevant literature.

According to findings by various studies, contributing to OSS is a "private-collective" activity, meaning that even though the contributed code becomes public, elements of it remain private property of its creator [235].

Table 9.1 summarizes the categories of motivational aspects that affect the involvement of individuals and businesses or other organizations in OSS efforts, and itemizes some of the more relevant bibliography for each aspect. These are further analyzed and discussed in the following sections.

Categorization		Motivating factors	Relevant bibliography			
			Theoretical	Empirical studies	Surveys	Other/generic
Motivation factors for Individuals	Intrinsic (hedonistic)	Enjoyment, amusement	[88]	[204], [172], [138], [109]	[138]	[226], [227], [16], [213]
		Fulfillment, satisfaction	[88], [21]	[102], [89]	[102]	
		Sense of cientific discovery and creativity	[254]	[254], [204], [138]	[138]	[46]
		Challenge		[138]	[138]	[203]
	Extrinsic	Reputation and status	[88]	[109], [89]		[213]
		Signalling incentives (career, job opportunities)	[145], [146], [147], [22]	[109], [22], [89], [141], [172]		[213]
		Financial incentives and rewards	[145], [146], [147], [22]	[22], [89], [138]	[138]	
	Political/ideological	Anti-commercialism	[22], [68]	[22], [89]		[221]
		Hacker culture	[68]			[221], [5], [186]
		Advancing the free s/w movement				
	Social	Sense of belonging (community identification)	[88], [22]	[102], [109], [89], [22]	[102]	[221]
		Altruism	[22], [21]	[102]	[102]	
		Contribution to public good	[88]	[138]	[138], [235]	[12], [213]
		Generalized reciprocity	[11]	[101]	[101]	[185], [257], [16]
	Technological environment and working style	Learning and skills development	[22], [254], [68]	[22], [89], [254], [137]		
		Community contribution and feedback	[22], [21]	[22]		[185]
		Bleeding-edge technology	[22]	[22]		[213]
		Realisation of personal ideas	[68], [22]	[89], [22]		[185], [75], [213]
		User-driven innovation	[146], [147]	[101], [138]	[101], [138]	[232]
		Integration of individual's fixes in maintained code				
Motivation factors for Businesses	Process/product	Adoption of code that fits company's business model				
		High quality code	[68]	[250], [106]		[134], [12], [191]
		Contribution, feedback & support from OS community	[22]	[22], [250], [106]		[134], [191]
		Standardization	[22], [68], [146], [147]	[22], [250], [106]		
		Fragmentation and modularity techniques		[250], [106]		[134], [12]
	Openness	User-driven innovation	[22], [145], [146], [147]	[22], [250], [106]	[235]	[12]
		Human capital improvement	[22], [145], [147]	[22], [138], [102], [250], [106]	[138], [102]	[20]
		Adoption of OS model	[22], [145], [147]	[22], [250]	[235]	
		Competition overview	[145], [147]	[250]		
		Reputation		[250]		
		Commercial visibility		[106]		[20]
		Employee satisfaction	[145]	[107]		[12]

Table 9.1 A categorization of the factors motivating individuals and organizations to contribute to OSS. The relevant bibliography is also included and grouped in works including theoretical studies, empirical findings, surveys and reviews of the scientific area, or other general studies.

9.1 Motivational Aspects for Individuals

In attempting to identify what motivates developers to contribute to OSS projects, most studies focus on two types of motivating factors: intrinsic (or hedonistic, including factors such as enjoyment, challenge, and satisfaction) and extrinsic (mostly involving economic and signalling incentives). Additional factors uncovered are of a social, political or ideological nature (participation into a community effort, or belief in the OSS movement and what it represents) as well as what the technological environment that OSS projects are surrounded by may offer an individual.

A general hypothesis that contributions to OSS projects are merely driven by generosity and altruism is challenged by economists [145]. As found in a survey-based empirical study of motivational profiles, although altruism does play a significant role, it is not enough to explain the phenomenon; other motives we mentioned come into play [48]. We discuss these below.

9.1.1 Intrinsic

By intrinsic motivations, which have been widely studied in psychology (see [138] and the references therein), we refer to those relating to satisfaction of an immediate need, or pleasure from carrying out a specific activity [22]. These rewards can involve intellectual gratification, a sense of aesthetic pleasure [21], or satisfaction of a basic need for competence, control or autonomy [236].

Following are some intrinsic motivational factors for contributing to OSS that have been identified in the literature.

Enjoyment and Amusement Linus Torvalds characteristically claimed that the main motivation behind programming for the Linux project is fun [226]. The enjoyment factor has been found to be important by other researchers [88, 227, 204, 213]. Various empirical studies also back this statement, referring to an "innate desire to code" [102] and concluding that hedonistic motivations are among the strongest ones [109, 138, 16, 172].

Satisfaction and Fulfillment OSS programming has been described as an artistic process providing satisfaction and fulfillment associated with problem

solving [21].

Sense of Scientific Discovery, Creativity and Challenge Similarly, various studies identify as strong incentives the sense of scientific discovery and the inherent creativity factor, as well as the challenge of participating in a project together with many other highly skilled and motivated programmers [21, 204, 138, 203, 46, 254].

9.1.2 Extrinsic

By extrinsic motivations we refer to those that are satisfied indirectly, and usually through financial or monetary compensations [22]. An example of this indirect fulfillment would be a programmer contributing to an OSS project so as to gain visibility or better reputation within the community, in the hope that this will lead to better job opportunities.

Following are examples of extrinsic motivational factors according to the literature.

Reputation and Status As described in the preceding example, programmers have various reasons to project and increase their reputation and status within the professional community [88, 109, 89, 213]. Participation in successful, and often high-profile OSS projects are excellent media to achieve this, as they provide high visibility into the programmer's work, style, performance and achievements.

Signaling Incentives Participation in an OSS project may be driven by career concerns. Developers may use this approach to signal their availability and skills for potential job opportunities [145, 146, 147, 22, 176]. Another category of signalling incentives relates to the developer's ego gratification that results from peer recognition of their work, in a way similar to the intellectual gratification that motivates members of the scientific community [14, 46, 11, 21, 120].

Financial Incentives and Monetary Rewards There are various ways in which participation in OSS projects can lead to financial gain. As open source

technologies are widely used and taught in universities, graduate programmers are likely to continue using them, due to their lower costs (the so called Alumni Effect [146, 147]). Additionally, participation in an OSS project benefits from bug fixes, customizations and extensions contributed by other members. This can offset the cost of one's own contributions to the project. And monetary rewards can also be directly reaped from this activity [22, 89, 138]. Reference [136] provides an in-depth description of the way in which financial incentives (among others) influence and motivate OSS developers by incorporating both intrinsic and extrinsic elements in an integrative theory about OSS developer motivation.

9.1.3 Political and Ideological

Participation in OSS projects can also be the result of one's political, ideological or cultural beliefs. For example OSS entails an expression of anti-commercialism [22, 89, 68, 221, 62], while there is also a clear cultural element shared within OSS communities (the "hacker culture") [68, 221, 5, 186, 62]. At the same time, it can also be a statement belief in the free software movement, and an active attempt to advance it.

9.1.4 Social

If we consider OSS as a social movement, we can adopt another perspective for studying the incentives to participation in OSS projects [239]. The following types of motivations can be identified.

Altruism Contribution to an OSS project has a clear altruistic element. Various studies provide evidence of this, including empirical work contrasting the outcomes of a phone survey of various firms regarding their motivations for participating in OSS projects with an earlier study of individual respondents [22], a study that identified the prominent motivating factors based on classical theory as well as empirical studies [102], and a paper discussing key economic problems of OSS based on theory and a proposed simulation model [21].

Sense of Belonging and Contributing to a Public Good Community identification [88, 102, 109, 89, 221, 22] is a widely accepted motivating factor.

As concluded in a Linux-based study [5], it involves elements such as participation in a collective effort, social interactions and group influences. Other factors include a consciousness of connecting with other members, shared rituals, a sense of duty, trust and loyalty toward the community, and contributing to a public good [186, 205, 221, 5, 213].

Generalized Reciprocity Also referred to as "gift giving" [185], it involves relationships forged through mutual reciprocal actions, similar to relationships within families [257] or academic societies [11].

9.1.5 Technological Environment and Working Style

OSS projects are typically characterized by the formation of highly motivated and skilled communities of developers and a technologically challenging and exciting environment. This is found to be an important motivating factor for participating and contributing to such efforts, for various reasons.

Learning and Skills Development Achieved through interaction with other members, examination of other developers' code, and the open and transparent OSS development processes [22, 89, 254, 68, 137].

Community Contribution and Feedback The ability to reciprocally interact on a technical level with other project members [21, 22, 185].

Working with Bleeding-Edge Technology Exposure to state-of-the art technologies and inventions that would otherwise be hard to come across in such an open way, if at all [22, 213].

Realisation of Personal Ideas The OSS project environment allows the implementation and realisation of bigger goals and aspirations than would otherwise be possible [185, 89, 75, 68, 22, 213].

User-Driven Innovation Direct interaction with users, and immediate integration of their needs into the various development phases is an opportunity offered by OSS projects [232, 101, 138, 147] that may be hard to get in a

larger software development house where engineers are often shielded from customers through multiple layers of support personnel.

Integration of Individual Fixes By submitting one's fixes and improvements to an OSS project, these become integral parts of the maintained source code and will therefore be available in future releases, to both the contributing individual and others [213].

9.2 Motivational Aspects for Businesses

Several of the motivations for individuals may also apply in the case of businesses or other organizations that contribute to OSS software projects. We examine separately motivations pertaining to the software development processes and the resulting products, and motivations relevant to the open nature of OSS projects.

9.2.1 Process and Product

The development carried out within proprietary software firms can be enhanced and enriched through participation in OSS projects in several ways.

Contribution From the OSS Community The feedback and support received from the OSS community can be highly beneficial to the software production process, which is typically more closed and isolated within proprietary vendor firms [22, 250, 106, 134, 191, 45]. Specific benefits include peer review, reduction of effort duplication, utilization of existent modules, and access to talented developers.

Modularity The product's design and effort allocation can be improved by taking advantage of the extensive modularity approaches used in OSS projects. This is evidenced in numerous literature reviews, survey studies, software provision models, case studies and related analyses [250, 134, 12, 106].

Code that Fits the Company's Model A business may contribute to an OSS project in order to adopt some code that fits their business model. The

fit may concern factors such as code functionality, quality, cost, licensing, and competition. In start-up or research intensive companies, additional motivations such as lowered entry barriers and fast product development seem to play a significant role [96]. Examples of this include Apple[1] basing their Mac OS X operating system on OSS projects such as FreeBSD, GCC[2] and WebKit;[3] TiVo[4] using Linux for their DVR; and Juniper[5] basing their router software on FreeBSD.

9.2.2 Openness

Additional motivation for the business and its employees stems from the open nature of OSS projects. Specific motivational factors may include.

Commercial visibility and reputation is gained by participating in OSS projects [20, 106, 250, 45]. For instance, Sun (now part of Oracle) gained tremendous goodwill from the OSS community and developers in general by open-sourcing the Java platform and the Solaris operating system.

Competition knowledge can be gathered through the project community or by observing other companies that participate in the same projects [145, 147, 250].

Adoption of the OSS model and ideas may significantly improve the processes and working environment within a business [22, 145, 147, 250].

Human capital improvement Participation in OSS projects can have a beneficial effect on the employees [22, 145, 147, 138, 102, 250, 20, 106, 12, 107]. The same motivations that were examined for the case of individual developers will, to a certain extent, also apply here. Employees that take initiatives will experience recognition by the other members of the project community [107], restrictive managerial attitudes will be softened and programmers will be working in a more intellectually stimulating environment [145].

[1] http://www.apple.com/
[2] http://gcc.gnu.org/
[3] http://webkit.org/
[4] http://www.tivo.com/
[5] http://www.juniper.net/

User-driven innovation The active involvement of users throughout the various phases of the project evolution is a particularly important advantage for software vendor firms [22, 145, 147, 250, 12, 106, 235].

10

Impact and Outlook

The impact that OSS has had both on the software business, but also on society as a whole, at a local and global scale, is undeniable. For example science and education, developing countries, and the youth of our societies have all been positively affected by OSS directly or indirectly.

It is interesting to examine how OSS and its main characteristics are aligned with many of the current global challenges faced in different fields and disciplines, and how it can be applied to help provide potential solutions.

It is also worth delving into the outlook for OSS, both in terms of what its future might hold, and in terms of identifying the research directions that some of the main concerns currently voiced around it might open.

10.1 Impact on the Software Industry

Various aspects of the impact of OSS on the software business and market have already been discussed in other chapters of this survey. Overall, it could be argued that OSS has "broadened" the software industry by significantly reducing entry barriers for both individuals and companies, introducing a healthier form of competition, reducing the possibility of market monopolizing and forming globally distributed software production communities. The

free and open licensing of OSS has played a very big role in fuelling these changes [96].

The emergence of a multitude of open, widely distributed and exciting OSS projects has allowed new (as well as experienced) developers and IT practitioners to enter the world of innovative software production, develop their skills, exchange ideas, showcase their capabilities and become parts of a vibrating software industry [219]. This is to a large extent made possible through the transparency and accessibility of almost all the information about the resulting software artifacts, including source code, past versions, recorded issues and communications, documentation, development roadmap and tools.

At the same time OSS has provided the opportunity for small firms to enter the market with lower costs, barriers and risks, by utilizing or building on applications, operating systems and utilities that are founded on OSS and made available through a relatively low investment [74, 245]. Such firms will in turn offer employment to IT professionals, possibly including some who originated from the development communities that supported these same OSS products.

New software development models, technologies and infrastructures for collaboration, design and modularisation patterns, and concurrent development and debugging processes have emerged from OSS, and are changing the commercial software development scene [244, 72, 216, 245]. This rapid spread of ideas from the OSS world has reached the proprietary software domain as well [219]. Software vendors carefully follow these developments, and many opt to either embrace similar methodologies, or keep very close ties with the OSS domain, for practical or strategic reasons.

With the appearance of many dominant OSS products in most areas of software, including applications, operating systems, infrastructural and middleware software, which are available at a very small cost, modifiable and adaptable, a new skill for IT professionals is the ability to monitor and search the OSS repositories to identify, combine and work with these products [219].

Global competition within the software industry has also been significantly affected by OSS. One example is the introduction of a new type of firm in the market that focuses on the packaging, distribution and support of OSS products (see also Section 7.3). OSS also offers an effective anti-monopolistic defence, and allows the software industry to move away from a model based on proprietary software lock-in [74].

Due to its licensing characteristics and low cost, many hardware vendors also increasingly prefer to ship their products with embedded OSS. For example various consumer electronics and telecommunications firms increasingly use the Linux operating system or other OSS in their products [237, 74, 96].

The OSS approach of full transparency and distribution of digital archives can also help in preserving software artifacts in the future. By covering the basic requirements of digital preservation systems [194], [1] OSS has the potential of preserving the current state of the art in software engineering for future study [112]. Indeed, at the moment researchers can dig into publicly available software archives dating as back as mid-1980, whereas binaries from the same era are barely executable on current computers.

Finally, we have reviewed in considerable detail in Chapter 7 and summarized in Table 7.1 the entire ecosystem of different types of companies that formed around OSS, and the different business opportunities and models that are emerging.

10.2 Impact on Society

In order to examine the impact of OSS on society, we focus on three important, dynamically evolving and changing sectors of global society: the youth, science and education, and the developing countries.

10.2.1 The Youth and Teaching

In today's modern society, free and easy access to information is of paramount importance in the development of skills and knowledge, in particular for the young. OSS has undeniably helped underpin the spread of the internet with technologies such as web servers, browsers, and messaging technologies [219]. These technologies have become a part of the youth's everyday lives, and allow them to search for information, connect with others and join communities. By engaging in such collaborative activities, cultural divides are torn down, prejudices are abandoned, and a real global identity can be achieved [219].

The low cost and free availability of many OSS educational applications

[1] Linus Torvalds humorously quoted on the subject:"Only wimps use tape backup: *real* men just upload their important stuff on ftp, and let the rest of the world mirror it."

also allows them to be used in schools and in teaching without requiring large funds or resources. These provide access to many online repositories of learning material that are openly accessible to all (one example being MIT's Open CourseWare Initiative)[2].

For young people with special interest in software, or aiming to enter the software industry, OSS provides great opportunities for developers or entrepreneurs, regardless of one's geographic location. A track record of one's contributions to projects, and participation in OSS communities can become a substitute for a CV. The knowledge, information and skills that flow inside OSS project communities and can be accumulated through participation are immense, rendering them "a complete new kind of learning platform" [141].

The culture of OSS projects, and the acceptance of new contributors to their communities further facilitate learning. The access to OSS source code makes this software particularly appropriate for teaching in technically oriented classes. Even if students are not skilled enough to make contributions to a project, they can learn from it by examining and studying it (reading other's code is a strongly recommended practice for developing programming skills [208, 213, 211]). Additionally, organizational and project management models that are associated with running OSS projects can be studied through observation.

Finally, OSS is the distribution mechanism and driving force behind many innovative and successful educational initiatives. These include the Scratch[3] and EToys[4] programming environments for children, the Processing[5] platform for creative artists, and the One Laptop Per Child initiative [132].

10.2.2 Science, Engineering and Research

The philosophy of sharing and open cooperation is not only a key element of OSS projects and initiatives. Science, research, and engineering are areas where widespread collaboration on large and complex projects and tasks are inherent and necessary. The open source model of innovation can be applied in these areas as well, in fields ranging beyond software development, such

[2] `http://ocw.mit.edu/`
[3] `http://scratch.mit.edu`
[4] `http://www.squeakland.org`
[5] `http://www.processing.org`

as social sciences, life sciences, and biomedicine [240].

In fact, it is argued that the OSS approach can be compared to the way research is conducted in the scientific and academic communities [129]. In both communities strong norms apply regarding the respect for and importance of knowledge and recognition of contribution, and the need for public validation of scientific or engineering results (as in peer reviewing). Similar intellectual property regimes apply to both worlds, and members of both communities are primarily rewarded through the dissemination of their work, the resulting status and prestige gains, the learning experience, and ultimately the fun of it, rather than being motivated mainly through monetary incentives. Finally, in both domains virtual collaboration within large distributed teams is a fundamental element.

Another interesting observation is that work in both OSS development and science and research are financed through similar processes. In many cases in science individual researchers will become interested in a particular subject and conduct work that is not directly financed through their main projects [237]. Similarly, as has been discussed, a large percentage of OSS developers are at the same time employed by firms working on different projects.

As discussed in a literature survey of OSS characteristics that promote research [237], these similarities open up opportunities for dialog between members of the two communities. Although the fundamental goals and questions may vary, the shared creative process and rules are similar. Observing OSS development can therefore make more apparent the corresponding strengths or weaknesses of the scientific process, and help improve it.

Another important impact of OSS to the research and academic community is the availability of very large repositories of data [161]. This data can be of different sorts including source code, mailing lists, bug reports, technical communications, user feedback, and version control repositories. All this information is very valuable to researchers, as it allows them to study technical, organizational, or behavioural matters (e.g. millions of lines of source code, specification documents and technical exchanges, topics of discussions, people's influence on the development process, decision making and coordination, conversational protocol etc.) [237]. These repositories can be searched and data retrieved based on recent advances in data mining technologies can be analysed both qualitatively and quantitatively to reveal new information useful in fields as varied as engineering, sociology and economics, to name

but a few [252].

Finally, the use of OSS in research and academia allows progress to be made in spite of scarcity of funds, and quicker results to be produced by using OSS tools. A couple of recent examples of OSS projects that are applied to research include the MOSES[6] OSS toolkit for statistical machine translation, the IntAct[7] OSS database and software suite for modeling, storing and analyzing molecular interaction data, and the OpenStack[8] OSS Cloud computing system and the R-Project[9] statistical computing platform.

10.2.3 The Developing Countries

Developing countries have a great percentage of the world's brain power, yet only enjoy a very small share of the world's technological innovation [129]. OSS is considered to be a solution for bridging the digital divide between the most advanced countries and developing countries that still face massive economic, social and infrastructural challenges. Being generally free and easily accessible, OSS is attractive for all types of users, ranging from home users and schools to businesses and governments, especially in challenged environments where funds and resources are particularly scarce. As an added bonus many OSS offerings require fewer computing resources than their proprietary alternatives, and can therefore run on older or cheaper hardware.

In such economies and conditions, locking institutions into proprietary software that charges, or may in the future start charging license fees is not feasible. With OSS it is possible to train professionals to use, modify and maintain the software they need to perform their professional, educational or everyday tasks, due to its openness and availability of the source code [237].

The distributed development model of OSS also allows people in developing countries to participate in and learn from such projects, without the need to relocate to other parts of the world, as would be typically required in order to work in a large software firm (an effect labelled the "brain drain", whereby educated people are forced to abandon their home countries in search of employment and career opportunities elsewhere in the world) [219].

---

[6] http://www.statmt.org/moses/
[7] http://www.ebi.ac.uk/intact/main.xhtml
[8] http://www.openstack.org/
[9] http://www.r-project.org

For developing countries to evolve and benefit from current technological advances, a community of trained local professionals must be formed and supported. Outsourcing opportunities offer considerable employment currently, but this is not enough [219]. The OSS approach is well suited for empowering the software development and research communities of developing countries.

A particularly interesting case of the application of OSS to serve the needs of developing countries is the OLPC (One Laptop Per Child) initiative [132], which based the development of the XO, a very low cost portable computer on the choice of the Fedora Linux operating system and the Sugar graphical user interface. The XO was developed with the key concepts of learning, openness and collaboration in mind. The use of the Linux/Sugar OSS solutions allowed its production cost to remain minimal, and upheld the open source ideology. Indeed this project's technology proved to be a potential threat to the PC industry in emerging markets.

10.3 Tackling Global Challenges

The OSS philosophy adheres to principles that reach beyond software development and the IT domain, and revolve around openness and collaboration at a global level. These principles are particularly relevant and applicable to many of today's global challenges and problems, and the OSS approach can be part of the process of tackling these challenges and making progress that will potentially lead to an improvement at a global scale.

The effects of globalization are widely observed, and its impact on education and universities is seen in numerous collaborative projects and exchange programs. Concepts such as e-learning and collaborative learning have attracted a lot of attention within science and education [219].

Methods for distributed organization and division of labour at a global scale have also been developed [129], and OSS projects have a lot to offer in this domain. Indeed, the results of research performed on OSS has been of interest to fields as varied as social sciences, economics, anthropology and computer science [237]. The OSS development approach thus bridges interdisciplinary barriers in both the research, engineering and organizational context.

With global scale implementations and infrastructures in fields such as

telecommunications and networking, medicine and biology, e-government, and transportation, OSS projects can provide an organization method that is based on open tools, active participation of a large community of developers and users, transparency of processes, innovative governance structures, better services and a fresh mentality [165]. In particular, processes including coordinating the flow of information, tracking and resolution of issues, allocation of tasks and responsibilities, and handling the difficulties inherent in the management of widely distributed efforts can be copied from those used in large OSS projects.

The OSS model is also of particular importance as it provides an example for how a public and openly available good, namely the information contained within OSS projects and their artifacts can give rise to profitable commercial investments [129]. We have examined in this survey the various business models and ecosystems that surround such OSS efforts in Chapter 7.

The entire software market has been influenced at a global level as a result of OSS (see our discussion in Chapter 10.1), affecting issues of monopoly, competition, and market placement.

OSS has also been found to harness a quality-enhancing demand-side learning approach [27], whereby through close interaction with user communities, and more frequent development and release cycles that allow constant testing, incentives are provided for user groups to report problems or request new features. This approach, which relies on opening the source code and providing the software with no licensing costs, has been found to be more beneficial than other, conventional cost-reducing approaches in the market. Given the global financial challenges we are faced with, the OSS experience can clearly be leveraged in other domains as well.

10.4 Concerns, Research and Outlook

Although OSS has the advantages and positive characteristics discussed in this survey, it is by no means a panacea for all problems faced in the software production process. First of all, not all software projects are amenable to open-sourcing. Projects that contain intellectual property of very high value, or projects with very case-specific code that is not reusable, are unlikely candidates. The same is true for products based on an arcane technology or addressing an application domain that is unlikely to attract OSS enthousiasts.

In the cases where open-sourcing is a potential approach, there is still plenty of room for improvement.

10.4.1 Concerns

Various concerns have been voiced around the OSS methodology and philosophy, some of which have alread been already discussed in this survey.

Usability Issues It is considered that not enough attention is paid to usability in OSS projects [104]. One potential reason is that OSS developers focus more on the functional characteristics of their code, rather than the user interfaces and usability, and that they are often not educated or trained to deal with usability and human-computer-interaction matters [29].

Indeed the most successful OSS projects include software used by people with experience in IT, such as operating systems, libraries, compilers and shell applications, which require less training.

Often the OSS project development teams have no access to usability experts to consult or usability labs to run experiments on, resulting in minimalistic or unpolished user interfaces [171, 219]. Additionally, usability design should take place from the beginning, and it is a task that is difficult to distribute [171].

The situation seems to have improved in recent years, however, especially as the result of high-profile user-oriented projects such as word processors (e.g. OpenOffice) and web browsers (e.g. Firefox). Moreover, there is also an emerging trend in OSS projects, especially those working on desktop applications, to employ usability experts in user interface design. The current versions of both leading OSS desktop environments (GNOME and KDE) feature human-computer interaction guidelines while reusable interface elements have been designed by professional graphics artists, leading to more consistent user experience.

Potential solutions include involving usability experts, educating developers around usability issues, academic involvement (where considerable research in usability and human-computer-interaction takes place), and more active participation of users. The links between users and developers should also be reinforced with better communication tools that will allow the description and tracking of usability problems [171].

Licensing Complexity As we have discussed in Section 6.5, the large number of OSS licencing options, and the risks of combining them, pose challenges to developers of software applications.

The need for a good understanding of the legal implications of incorporating OSS code within proprietary applications is increasingly important, and software practitioners are becoming aware of it and looking for methods to address this [105, 85]. Furthermore, OSS supporters are increasingly enforcing the software license's requirements.[10]

The Problem of Commons There is a clear gap between the public and open nature of OSS software and the private and competitive practices of proprietary software firms. The same distinction is found between academic research in various fields, and the private enterprises that secure property rights on the resulting ideas and innovations [129].

The right balance needs to be found between the openness of OSS and the protectionism of private firms. This requires reconsidering the practices of intellectual property rights management to find the optimum balance between software developers' incentives obtained by gaining exclusive rights to their work and society's benefit through the proliferation and wide sharing of innovations [107].

An additional topic of debate concerns whether software patents should be allowed, and to what extent. Advocates of patenting software argue that it is a necessary means of enforcing a property claim and protecting an invention, so that its owner can extract an economic return from it, and it thus promotes innovation and development [129].

On the other hand, organizations such as the Free Software Foundation, have taken a strong stance and have been campaigning against software patenting, stating that it hinders and undermines the free software movement [51, 26]. Some of the arguments include the fact that, contrary to copyrights, patents cover ideas and their use, and not the details of specific implementations [83], and thus constitute an absolute monopoly against using a certain idea, even if developers could prove that they independently invented it. Additionally, the duration of patents, which is of the order of 20 years, is very long for the software field. They further argue that the quality and scope of

[10] See e.g. `http://www.gpl-violations.org/`.

patents is sometimes inadequately evaluated, leading to trivial patents covering obvious inventions [13]. Finally, they contend that the financial cost of obtaining a patent, investigating prior art (even a small program can cover hundreds of patentable ideas), or defending one's self or organization against a patent dispute is prohibitively large [83, 51, 101]. In an attempt to fight software patents, the 2007 revision of FSF's GPL (GPLv3) includes language that forces distributors of GPL code to license their patents practiced by the code to the software's users, thus hindering attempts of patent holders' to collect royalties from GPLed software users [217].

Motivational Issues The motivational factors for developers contributing effort to OSS projects have been examined in Chapter 9.1. Sometimes these factors are not strong enough, and developers can drift out of the project communities and stop contributing [204, 240].

Interestingly the project's licensing decisions, as well as the participation of developers in the key project decisions are strongly correlated with the project's success, and the link seems to involve additional motivation for the developers to continue contributing to the project [203].

Forking Danger The danger of forking has been discussed in several places in this survey. The risk is for a part of code (that could be embedded in another application) to remain stagnant and unsupported. This risk is not as pronounced in proprietary software, where market pressures force the firm's management to closely monitor and direct the decisions of developers [240].

10.4.2 Research and Outlook

The field of OSS has attracted a lot of attention, and research addresses practically all of its aspects, including technological, social, managerial and economic.

A taxonomy of OSS research and frameworks for performing empirical studies has been proposed [182, 124]. The research however is still at an evolving stage, with some aspects receiving more attention than others. Research methods used so far include case studies, surveys and quantitative studies, but combination of these would yield more substantial results [223].

Although OSS project data is easy to acquire and analyse, there is not yet a lot of insight on characteristics such as quality, innovation, and evolution [223]. Additionally, the complexities of mining this data are considerable and the dangers of misinterpretation are present [168].

We try to identify some of the more active areas of research and evolution based on current literature, and discuss the direction in which they may lead the future of OSS.

Incentives for Contributing to OSS Projects A deeper understanding of what motivates developers to participate and contribute to OSS projects would be useful [240, 5]. There are various open questions that could be investigated.

What role do the project's characteristics and various rewards play in motivating developers? How much does the project's quality and (current or expected) success affect this? How does this choice weigh with respect to contributing to a proprietary software project [60]?

What are the dynamics of the average developer's role within an OSS project community? How does their attitude change over time, and how does this affect the project's sustainability?

What is the role of software firms in this equilibrium, what are their experiences with participation in OSS initiatives so far [107]? As the participation of firms becomes more important, could there be a new mix of incentives, combining the participation in the OSS project with the impact of this on the employee's career within the firm [240, 190]?

Is there some correlation between the motivation to participate to an OSS project and the project's license type? Do the economic effects of the license choice affect the developers' choice of project to participate in [203]?

Licensing As we discussed in Section 6.5 the multitude of OSS licenses and their combination as OSS is reused in other products poses various challenges and risks.

The way in which this interdependence and combination of licenses evolves needs to be analysed, and the weaknesses and limitations of the current licenses will need to be addressed and possibly new licensing types developed to simplify the current situation [180].

The effects of the licensing choice on the reuse of OSS code in proprietary projects also needs to be analysed, and the specific factors related to the license that affect the adoption and reuse of the OSS code need to be identified and studied [98]. Automatic tools that are already considered for consistency checking between licenses of cloned OSS software [86] may need to evolve.

Reuse The reuse of OSS software into other projects has become an increasingly common activity. Still there are questions about the efficacy of and requirements for successfully reusing OSS software, as well as concerns about how practical and advantageous it really is.

Research on the issue of OSS software reuse could identify different aspects of this practice such as what types of projects are more likely candidates for reuse and what individual or organizational characteristics affect this; whether the frequency of a specific component's reuse could be predicted based on its properties, such as functionality and quality; what are the main motivations for reusing OSS software; and finally, what is the cost to an OSS project of building reusable components, and how this cost is distributed [98].

Community The properties of OSS project communities and their counterpart commercial software development workforces should be studied both independently and comparatively. The goal would be to identify how innovation is carried out in both contexts, how they can mutually benefit from each other through joint efforts or the exchange of results, and what the implications of this may be toward extending the theory of private-collective innovation [98].

As more and more developers from private software firms actively participate in OSS projects, it would be important to investigate how this affects the two, and whether a hybrid form of community may be emerging. If so, would this be incompatible with the OSS principles, and would it hinder the OSS processes? Or could it be a positive development that may lead to more strategic partnerships, strengthened at the community level?

As the effectiveness of software production depends on the quality and characteristics of the underlying communities of developers, such information would be very valuable both for the OSS projects and the firms that form

relations with them [53].

Business Aspects At the business level, OSS and proprietary software products are generally in competition. However we have discussed in Section 7.4 there are also cooperation opportunities between the two [240].

The result of these hybrid efforts (labelled as second generation OSS, or OSSg2) offer important added value propositions to their customers. However this is still an evolving model, undergoing continuous adjustments, and new models are expected to emerge blurring the lines between the two worlds even further [244].

The motivation for moving from proprietary to OSS regimes may also be the subject of future research [250]. The ability to evaluate the performance and outcomes of such joint efforts, including methodologies and tools, will be very valuable [191].

The type of licensing of the OSS software will clearly determine to a large extent whether such cooperations may be feasible, and how open their outcome must be. The implications of this on the competitive placement of the resulting product is also an area where research could produce valuable results [142]. Automated tools for assessing copyright attributions and code ownership by companies engaged in OSS projects would also be of interest to the industry [191].

Finally it is argued that involvement in OSS projects has many similar characteristics with outsourcing, for example the choice of what part of the product to develop and what activities to perform internally, and what to outsource (or open-source). It would thus be interesting to compare these two practices and study their similarities and differences [250].

Overall, the OSS principles and practices represent a trend toward democratizing innovation and creativity, by empowering users to contribute and evolve through the openness of projects outcomes and communities alike [233, 234]. The full breadth of social and economic impact that can be achieved through their application in all the fields we have examined remains to be seen. However, there is an important opportunity to employ, adapt, or explore the applicability of the OSS principles in each field's research agenda [237].

Open source software is part of a paradigm shift in the way we build com-

plex artefacts, divide our labor, organize sophisticated endeavours, and handle supply-chain relationships. Although it is unlikely that OSS will become the only, or even the main, game in town, there is plenty of evidence indicating that practitioners and researchers can benefit a lot from its study and use.

Acknowledgements

We would like to thank Vaggelis Giannikas, Vassilis Karakoidas, and Dimitris Mitropoulos for comments and suggestions on earlier drafts of this work. We are especially thankful for the feedback of Panos Louridas, who performed this survey's internal review. Furthermore, we are grateful to Charles Corbett, Uday Karmakar, Thanos Papadimitriou, and Konstantinos Psounis for their role in setting this paper in motion, and to Zac Rolnic for nurturing it to completion. We would also like to thank the anonymous reviewers for their insightful comments and suggestions.

A

Representative Applications

Following is a collection of some notable OSS applications, categorized according to their type. The selected applications include some of the most popular ones (in terms of downloads from the SourceForge.net site), some of our personal favourites, as well as some that we selected as representative examples from diverse categories. The selection is by no means objective.

A.1 Systems Applications

A.1.1 Operating Systems

GNU/Linux is a family of free, popular Unix-like computer operating systems using the Linux kernel, running on a variety of computer hardware, ranging from mobile phones, tablet computers and video game consoles, to mainframes and supercomputers. `http://www.linux.org/`

FreeBSD is a free Unix-like complete operating system descended from Unix cleaned from the AT&T code via the Berkeley Software Distribution (BSD). Generally regarded as reliable and robust. Focuses on performance and the x86 platform `http://www.freebsd.org/`

NetBSD is a freely available open source version of the Unix-derivative Berkeley Software Distribution (BSD). Still actively developed, the NetBSD project focuses on high quality design, stability, portability and performance. `http://www.netbsd.org/`

OpenBSD is a 1995 fork of NetBSD, focusing on security, portability, standardization, code correctness, and quality documentation. `http://www.openbsd.org/`

Xen is a virtual machine monitor that allows several guest operating systems to run on the same computer. `http://www.xen.org/`

A.1.2 Desktop Environments

Gnome is a desktop environment for computers running Linux and Unix-like operating systems. The Gnome project was initiated by the Mexican programmers Miguel de Icaza and Federico Mena. It is part of the GNU project and was released in 1999. `http://www.gnome.org/`

X11 , also known as the X Window System, provides a graphical user interface (GUI) for networked computers. The X.Org project provides an open source implementation of the X Window System. It is used as the base for running Gnome and KDE. `http://www.x.org/wiki/`

KDE is a graphical desktop environment and integrated set of cross-platform applications designed to run on Linux, FreeBSD, Windows, Solaris and Mac OS X systems. It was initially developed by a single person, 24 year old computer science student Matthias Ettrich at the University of Tübingen, as an OSS project and first released in 1998. `http://www.kde.org/`

A.1.3 Databases

MySQL is a relational database management system. It was originally distributed as open source software only under a standard copyleft GPL-like license. In 2001 the original developers founded the company MySQLAB that owns the copyright to the software. A dual licensing scheme, similar to Sendmail, was then adopted, allowing either free GPL-like licensing for OSS applications, or a non-free proprietary license to integrate it with proprietary products [51]. Currently owned by Oracle. `http://www.mysql.com/`

PostgreSQL is an object-relational database management system developed at UC Berkeley from 1986 to 1994. In 1995 a group of developers was formed around it as an open source project, giving it its new name. It is released under the PostgreSQL license, which is similar to the BSD license. `http://www.postgresql.org/`

HSQLDB is an embedded SQL relational database engine written in Java, including tools such as a command line SQL and GUI query interface. `http://hsqldb.org/`

SQLite is a software library that implements an embedded, self-contained, serverless, transactional SQL database engine. `http://www.sqlite.org/`

A.1.4 Web and Application Servers

Apache HTTP Server is a secure, efficient and extensible open-source HTTP server for modern operating systems including Unix and Windows NT. Its development started in 1998, as a fork off the httpd, a web server created at the National Center for Supercomputing Applications (NCSA). An online group of developers formed to support and enhance it. Within one year it was the most popular server on the internet. In March 1999 the group formed the Apache Software Foundation[1] with the goal of supporting the server (at an organizational, legal and financial level) and promoting the development of community-driven software. `http://httpd.apache.org/`

Jakarta Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies, developed by the Apache Software Foundation. `http://tomcat.apache.org/`

JBoss is a free software, open-source Java EE-based application server. `http://www.jboss.org/`

AWStats is a free powerful and featureful tool that generates advanced web, streaming, ftp or mail server statistics, in a graphical form. `http://awstats.sourceforge.net/`

A.1.5 System Administration Tools

Wireshark is a free and open-source packet analyzer used for network troubleshooting, analysis, software and communications protocol development, and education. `http://www.wireshark.org/`

Nagios is a powerful IT monitoring management system that allows organizations to identify and resolve IT infrastructure problems before they affect critical business processes. `http://www.nagios.org/`

phpMyAdmin is a tool written in PHP intended to handle the administration of MySQL over the Web. Currently it can create and drop databases, create/drop/alter tables, delete/edit/add fields, execute any SQL statement and manage keys on fields. `http://www.phpmyadmin.net/`

A.1.6 Email

Fetchmail was a mail utility for Unix-like systems, released in the early 1990s. The project was initiated by Eric Raymond, and used as a model in his famous essay "The Cathedral and the Bazaar" [185] to discuss his ideas about open source. `http://www.fetchmail.info/`

[1] http://www.apache.org/

Sendmail is a general purpose internetwork email routing facility (mail transfer agent) that supports many kinds of mail transfer and delivery methods, including the Simple Mail Transfer Protocol (SMTP) used for email transport over the internet. `http://www.sendmail.org/`

Postfix is a fast, easy-to-administer, and secure open-source mail transfer agent that routes and delivers electronic mail. `http://www.postfix.org/`

SpamAssassin is a computer program used for e-mail spam filtering that uses a variety of local and network tests to identify spam signatures. `http://spamassassin.apache.org/`

A.1.7 Networking Infrastructure

BIND is a widely used DNS software that provides a robust and stable platform on top of which organizations can build distributed computing systems with the knowledge that those systems are fully compliant with published DNS standards. `https://www.isc.org/software/bind`

Zenoss is an enterprise network and systems management application `http://www.zenoss.com/`

A.1.8 Security

Clonezilla is a partition or disk clone tool `http://www.clonezilla.org/`

putty is an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. `http://www.putty.org/`

TrueCrypt is a free open-source disk encryption tool for Windows, Mac OS X, and Linux. `http://truecrypt.org/`

WinSCP is an open source FTP/SFTP client for Windows, used for secure file transfer between computers. `http://winscp.net/eng/index.php`

A.2 Dekstop

Mosaic was a famous web browser developed at the NCSA in 1993, which played an important role in the first period of the internet. It was released as open source and free of charge for academic or internal business use. For commercial distribution the license terms had to be separately negotiated with the NCSA [51]. `http://www.ncsa.illinois.edu/`

Firefox is a free and open source web browser descended from the Mozilla Application Suite and managed by the Mozilla Corporation. `http://www.mozilla.org/`

Thunderbird is a free, open source, cross-platform e-mail and news client developed by the Mozilla Foundation. `http://www.mozilla.org/`

OpenOffice.org is an open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more. It was originally developed by the StarDivision, originally as StarOffice. StarDivision was acquired by Sun Microsystems in 1999. It was released in 2000 as open source under the LGPL/SISSL license and promoted as an alternative to Microsoft's Office suite of applications. `http://www.openoffice.org/`

Evolution is a program that provides integrated mail, addressbook and calendaring functionality to users of the Gnome desktop. `http://projects.gnome.org/evolution/`

Pidgin (Gaim) is an easy to use and free chat client that can connect to AIM, MSN, Yahoo, and more chat networks all at once. `http://www.pidgin.im/`

7Zip is a file compression and archival tool supporting various formats including 7z, ZIP, CAB, and RAR. `http://www.7-zip.org/`

KeePass Password Safe is a free, open source, light-weight and easy-to-use password manager for Windows `http://keepass.info/`

A.3 Entertainment

Mumble is a low-latency, high-quality voice communication tool for gamers. `http://mumble.sourceforge.net`

MediaInfo is a tool for getting technical information and tags for multimedia files. `http://mediainfo.sourceforge.net`

Media Player Classic is a free audio and video media player for Windows. `http://mpc-hc.sourceforge.net/`

Bittorrent is a popular open-source peer-to-peer file sharing client. `http://www.bittorrent.com`

VLC media player is an open source media player that can handle DVDs, (S)VCDs, Audio CDs, web streams, TV cards etc. `http://www.videolan.org/vlc/`

Audacity is a free, open source software for recording and editing sounds, available for Mac OS X, Windows, GNU/Linux, and other operating systems. `http://audacity.sourceforge.net`

A.4 Graphics

Inkscape is an Open Source vector graphics editor, with capabilities similar to Illustrator, CorelDraw, or Xara X, using the W3C standard Scalable Vector Graphics (SVG) file format. http://www.inkscape.org/

Ghostscript is an interpreter for the PostScript language and for PDF documents. http://www.ghostscript.com/

Gnuplot is a portable command-line driven graphing utility for Linux, Windows, OS X, VMS, and many other platforms. It is distirbuted under its own open source license (not GPL), according to which the source code is copyrighted but freely distributed. http://www.gnuplot.info/

GMT is an open source collection of tools for manipulating geographic and Cartesian data sets (including filtering, trend fitting, gridding, projecting, etc.) and producing Encapsulated PostScript File (EPS). http://gmt.soest.hawaii.edu/

GraphViz offers various tools for automatically rendering graphs specified in a declarative/textual fashion. http://www.graphviz.org/

GIMP , an acronym for GNU Image Manipulation Program, is a freely distributed program for such tasks as photo retouching, image composition and image authoring. Originally released in 1996, it is now ported to many operating systems. http://www.gimp.org/

iReport is a popular visual reporting tool for JasperReports (Java reporting library) and JasperServer (reporting server) that can manage charts, images, and subreports. http://www.jasperforge.org/projects/ireport

FreeMind is a free mind mapping application written in Java. It allows the user to edit a hierarchical set of ideas around a central concept. http://freemind.sourceforge.net

A.5 Education

Moodle is a Course Management System (CMS): a free web application that educators can use to create effective online learning sites. http://moodle.org/

Tux Paint is a painting program for kids between 3 and 12 years old http://tuxpaint.org/

EToys is a free educational tool for teaching children powerful ideas in compelling ways through a media-rich authoring environment and visual programming system. http://www.squeakland.org/

Scratch is an application aimed primarily at children that allows them to explore and experiment with the concepts of computer programming by using a simple graphical interface. http://scratch.mit.edu/

A.6 Scientific and Engineering

R-Project R is an extensible language and environment for statistical computing and graphics. It supports a wide variety of statistical and graphical techniques. http://www.r-project.org/

GNU Octave is a high-level language, primarily intended for numerical computations. It includes a command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments. http://www.gnu.org/software/octave

A.7 Publishing

TeX is a typesetting system designed and mostly written by Donald Knuth with the goal to allow anybody to produce high-quality books with the exact same results on all computers. http://www.tug.org/

Docbook is a semantic markup language for technical documentation, originally intended for writing technical documents related to computer hardware and software. It can be used for any other sort of documentation. http://www.docbook.org/

TCPDF is an Open Source PHP class for generating PDF documents. http://www.tecnick.com/public/code/cp_dpage.php?aiocp_dp=tcpdf

TeXnicCenter is an integrated environment for creating LaTeX documents on the Windows platform. http://www.texniccenter.org/

A.8 Software Development

A.8.1 Languages, Interpreters, Compilers

GCC, the GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, and Ada, as well as libraries for these languages (libstdc++, libgcj,...) and back ends for tens of processor architectures. GCC was originally written as the compiler for the GNU operating system. http://gcc.gnu.org/

Java Technology by Orcale (Sun) is a programming language originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. http://www.oracle.com/technetwork/java

Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. www.scala-lang.org/

Erlang is a programming language designed at the Ericsson Computer Science Laboratory. http://www.erlang.org/

Haskell is an advanced, purely functional programming language allowing rapid development of robust, concise, correct software. http://www.haskell.org/

Perl is a feature-rich programming language written by Larry Wall and released in 1986 as free software. It is distributed under the GPL or the Artistic License. A particular advantage of Perl are the thousands of add-on libraries available through the CPAN library. http://www.perl.org/

Python is a high-level object oriented programming language that places emphasis on code readability, and includes a large and comprehensive standard library. http://www.python.org/

PHP is a widely-used general-purpose scripting language released in 1995. It is especially suited for Web development and can be easily embedded into HTML. http://www.php.net/

Ruby is a dynamic, open source programming language with a focus on simplicity and productivity, and an elegant syntax that is natural to read and easy to write. http://www.ruby-lang.org/en/

Tcl/Tk , short for Tool Command Language, is an interpreted language with a very portable interpreter. Tcl is embeddable and extensible, and has been widely used since its creation in 1988 by John Ousterhout. It is particularly versatile for the creation of GUIs. http://www.tcl.tk/

Lua is a powerful, fast, lightweight, embeddable scripting language. http://www.lua.org/

ScummVM is a cross-platform interpreter for several point-and-click adventure engines. http://www.scummvm.org/

MinGW , a contraction of "Minimalist GNU for Windows", is a port of the GNU Compiler Collection (GCC), and GNU Binutils, for use in the development of native Microsoft Windows applications. http://www.mingw.org/

A.8.2 Libraries

Libpng was written as a companion to the PNG specification, as a way to reduce the amount of time and effort it takes to support the PNG file format in application programs. http://www.libpng.org

GD is an open source code library for the dynamic creation of images by programmers. GD creates PNG, JPEG and GIF images, among other formats. http://www.libgd.org

Boost C++ Libraries provide many useful free portable peer-reviewed C++ libraries. http://www.boost.org

A.8.3 Editors

Vim is an advanced text editor that seeks to provide the power of the de-facto Unix editor 'Vi', with a more complete feature set. http://www.vim.org/

GNU Emacs is an extensible, customizable text editor, part of the GNU project http://www.gnu.org/software/emacs/

Notepad++ is a source code editor and MS Windows Notepad replacement. http://notepad-plus.sourceforge.net

A.8.4 Version Control Systems

CVS is a version control system, used to keep track of all work and changes in a set of files, and to allow several developers to collaborate. http://www.nongnu.org/cvs/

Apache Subversion is a revision control system founded and sponsored in 2000 by CollabNet Inc. as an improvement to CVS. http://subversion.apache.org/

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. http://git-scm.com/

Mercurial is a free, distributed source control management tool. It efficiently handles projects of any size and offers an easy and intuitive interface. http://mercurial.selenic.com/

A.8.5 IDEs and Build Tools

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system, written mainly in Java. It was created by IBM in 2001 and released as open source. The Eclipse Foundation[2] was created in 2004 to support the Eclipse community. http://www.eclipse.org/

[2] http://www.eclipse.org/

NetBeans is an open-source project dedicated to providing software development products (the NetBeans IDE and the NetBeans Platform) for developers, users and the businesses. http://netbeans.org/

Apache Ant is a tool for automating the software build processes, similar to make. http://ant.apache.org/

A.8.6 Frameworks

ZK Simply Ajax and Mobile is an open-source Ajax Web application framework, written in Java, that enables creation of rich graphical user interfaces for Web applications with no JavaScript and little programming knowledge. http://www.zkoss.org

Mono is a software platform designed to easily create cross platform applications. Sponsored by Novell, Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA standards for C# and the Common Language Runtime. http://www.mono-project.com/Main_Page

Qt is a cross-platform application development framework widely used for the development of GUI programs and for developing non-GUI programs such as console tools and servers. Originally developed by Trolltech, which was acquired by Nokia in 2008. http://qt.nokia.com/

Ruby on Rails is an open source web application framework for the Ruby programming language, intended to be used with an Agile development methodology. http://www.rubyonrails.org/

A.9 Content Management Systems

Drupal is a popular free and open source CMS written in PHP, used for websites ranging from personal blogs to larger corporate and political sites including whitehouse.gov and data.gov.uk. http://drupal.org/

WordPress is a popular open source CMS, often used as a blog publishing application powered by PHP and MySQL. http://wordpress.org/

Joomla is an award-winning CMS, which enables users to build Web sites and powerful online applications, based on ease-of-use and extensibility http://www.joomla.org/

Arianne RPG is a multiplayer online engine to develop turn based and real time games providing a simple way of creating the game server rules and clients. http://arianne.sf.net

Media Wiki is a free software wiki package written in PHP, originally for use on Wikipedia, but now used by several other projects. http://www.mediawiki.org

A.10 Business Applications

Compiere is an open source ERP and CRM business solution for the small and medium-sized Enterprises in distribution, retail, service and manufacturing. It's architecture is such that it avoids duplication of information and the need for synchronization. `http://www.compiere.com/`

OpenERP is a complete and feature rich ERP and CRM system. OpenERP has a 3 layer structure: database, server and thin client that contains minimal business logic. The database is PostgreSQL, and the server is written in Python. `http://www.openerp.com`

PostBooks ERP is a free open source ERP, accounting and CRM package for small to midsized businesses. Built with open source Qt framework it runs on Linux, Mac, and Windows. Its business logic resides in a PostgreSQL database. `http://postbooks.sourceforge.net`

Openbravo ERP is a comprehensive and professional web-based open source ERP solution. The model for Openbravo was originally based on the Compiere ERP program. Using Openbravo, ERP organizations can automate and register most common business processes. `http://www.openbravo.com`

webERP is an open-source web-based ERP system. `http://www.weberp.org/HomePage`

OrangeHRM is an Open Source Human Resource Management System that covers Personnel Information Management, Employee Self Service, Leave, Time & Attendance, Benefits, and Recruitment. `http://orangehrm.sourceforge.net`

JStock is a free stock market software for 23 countries `http://jstock.sourceforge.net/`

References

[1] S. Abiteboul, I. Dar, R. Pop, G. Vasile, and D. Vodislav, "EDOS distribution system: A P2P architecture for open-source content dissemination," in *Open Source Development, Adoption and Innovation*, pp. 209–215, Springer Verlag, 2007. IFIP International Federation for Information Processing Volume 234.

[2] S. Ajila and D. Wu, "Empirical study of the effects of open source adoption on software development economics," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1517–1529, 2007.

[3] T. A. Alspaugh, W. Scacchi, and H. U. Asuncion, "Software licenses in context: The challenge of heterogeneously-licensed systems," *Journal of the Association for Information Systems*, vol. 11, pp. 731–754, Nov. 2010.

[4] U. Asklund and L. Bendix, "Study of configuration management for open source software," *IEE Proceedings—Software*, vol. 149, pp. 40–46, Feb. 2002.

[5] R. P. Bagozzi and U. M. Dholakia, "Open source software user communities: A study of participation in Linux user groups," *Management Science*, vol. 52, pp. 1099–1115, July 2006.

[6] C. Y. Baldwin and K. B. Clark, "The architecture of participation: Does code architecture mitigate free riding in the open source development model?," *Management Science*, vol. 52, pp. 1116–1127, July 2006.

[7] M. Bar and K. F. Fogel, *Open Source Development with CVS*. Scottsdale, AZ: The Coriolis Group, 2001.

[8] J. Barton, "From server room to living room," *Queue*, vol. 1, no. 5, pp. 20–32, 2003.

[9] B. Behlendorf, "Open source as a business strategy," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[10] Y. Benkler, "Intellectual property: Commons-based strategies and the problems of patents," *Science*, vol. 305, pp. 1110–1111, Aug. 2004.

[11] M. Bergquist and J. Ljungberg, "The power of gifts: Organising social relationships in open source communities," *Information Systems Journal*, vol. 11, no. 4, pp. 305–320, 2001.

[12] J. Bessen, "Open source software: Free provision of complex public goods," July 2005. Available at SSRN: `http://ssrn.com/abstract=588763`.

[13] J. Bessen and M. Meurer, *Patent Failure: How Judges, Bureaucrats, and Lawyers Put Innovators at Risk*. NJ, USA: Princeton University Press, 2008.

[14] N. Bezroukov, "Open source software development as a special type of academic research (critique of vulgar Raymondism)," *First Monday*, vol. 4, Oct. 1999.

[15] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista," *Communications of the ACM*, vol. 52, pp. 85–93, August 2009.

[16] J. Bitzer, W. Schrettl, and P. J. Schröder, "Intrinsic motivation in open source software development," *Journal of Comparative Economics*, vol. 35, pp. 160–169, May 2007.

[17] J. Bitzer and P. J. H. Schröder, *The Economics of Open Source Software Development*. Emerald Group Publishing, 2006.

[18] J. Bitzer and P. J. Schröder, "Bug-fixing and code-writing: The private provision of open source software," *Information Economics and Policy*, vol. 17, pp. 389–406, July 2005.

[19] J. Bitzer and P. J. Schröder, "The impact of entry and competition by open source software on innovation activity," in *The Economics of Open Source Software Development*, (J. Bitzer and P. J. Schröder, eds.), ch. 11, pp. 219–245, Emerald Group Publishing, 2006.

[20] A. Bonaccorsi, S. Giannangeli, and C. Rossi, "Entry strategies under competing standards: Hybrid business models in the open source software industry," *Management Science*, vol. 52, pp. 1085–1098, July 2006.

[21] A. Bonaccorsi and C. Rossi, "Why open source software can succeed," *Research Policy*, vol. 32, no. 7, pp. 1243–1258, 2003.

[22] A. Bonaccorsi and C. Rossi, "Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business," *Knowledge, Technology and Policy*, vol. 18, pp. 40–64, Dec. 2006.

[23] I. T. Bowman, R. C. Holt, and N. V. Brewster, "Linux as a case study: Its extracted software architecture," in *ICSE '99: Proceedings of the 21st International Conference on Software Engineering*, (New York), pp. 555–563, ACM, 1999.

[24] D. Bretthauer, "Open source software: A history," *Information Technology and Libraries*, vol. 21, pp. 3–11, March 2002.

[25] A. W. Brown and G. Booch, "Reusing open source software and practices: The impact of open source on commercial vendors," in *Software Reuse: Methods, Techniques, and Tools*, (C. Gacek, ed.), pp. 381–428, Springer Berlin / Heidelberg, 2002.

[26] B. W. Carver, "Share and share alike: Understanding and enforcing open source and free software licenses," *Berkely Technology Law Journal*, vol. 20, no. 1, pp. 443–481, 2005.

[27] R. Casadesus-Masanell and P. Ghemawat, "Dynamic mixed duopoly: A model motivated by Linux vs. Windows," *Management Science*, vol. 52, pp. 1072–1084, July 2006.

[28] M. Cassell, "Why governments innovate: adoption and implementation of open source software by four european cities," *International Public Management Journal*, vol. 11, no. 2, pp. 193–213, 2008.

[29] G. Çetin and M. Gokturk, "A measurement based framework for assessment of usability-centricness of open source software projects," in *SITIS '08: IEEE International Conference on Signal Image Technology and Internet Based Systems*, pp. 585–592, Dec. 2008.

[30] P. E. Ceruzzi, *A history of modern computing*, ch. Workstations, UNIX and the Net. MIT University Press, 2003.

[31] W. Chen, J. Li, J. Ma, R. Conradi, J. Ji, and C. Liu, "A survey of software development with open source components in Chinese software industry," in *Software Process Dynamics and Agility*, pp. 208–220, Springer Verlag, 2007.

[32] N. Choi, I. Chengalur-Smith, and A. Whitmore, "Managing first impressions of new open source software projects," *IEEE Software*, vol. 27, pp. 73–77, 2010.

[33] J. Colazo and Y. Fang, "Impact of license choice on open source software development activity," *J. Am. Soc. Inf. Sci. Technol.*, vol. 60, pp. 997–1011, May 2009.

[34] M. Conklin, J. Howison, and K. Crowston, "Collaboration using OSSmole: a repository of FLOSS data and analyses," in *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, (New York, NY, USA), pp. 1–5, ACM, 2005.

[35] T. Cornford, M. Shaikh, and C. Ciborra, "Hierarchy, laboratory and collective: Unveiling linux as innovation, machination and constitution," *Journal of the Association for Information Systems*, vol. 11, pp. i–v, Nov. 2010.

[36] K. Crowston and B. Scozzi, "Open source software projects as virtual organizations: Competency rallying for software development," *IEE Proceedings—Software*, vol. 149, pp. 3–17, Feb. 2002.

[37] K. Crowston, H. Annabi, and J. Howison, "Defining open source software project success," in *ICIS '03: Proceedings of the 24th International Conference on Information Systems*, 2003.

[38] K. Crowston, H. Annabi, J. Howison, and C. Masango, "Towards a portfolio of floss project success measures," in *WOSSE '04: Proceedings of the 4th Workshop on Open Source Software Engineering*, (Edinburgh, Scotland), pp. 29–33, May 2004.

[39] K. Crowston and J. Howison, "Hierarchy and centralization in free and open source software team communications," *Knowledge, Technology & Policy*, vol. 18, pp. 65–85, Dec. 2006.

[40] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: Theory and measures," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 123–148, 2006. Special Issue on Free/Open Source Software Processes.

[41] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison, "Self-organization of teams for free/libre open source software development," *Information and Software Technology*, vol. 49, no. 6, pp. 564–575, 2007. Qualitative Software Engineering Research.

[42] K. Crowston and M. Wade, "Introduction to jais special issue on empirical research on free/libre open source software," *Journal of the Association for Information Systems*, vol. 11, pp. i–v, Nov 2010.

[43] M. A. Cusumano, *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*. New York: The Free Press, 2004.

[44] M. A. Cusumano, "Reflections on free and open software," *Communications of the ACM*, vol. 47, pp. 25–27, Oct. 2004.

[45] L. Dahlander, "Penguin in a new suit: a tale of how de novo entrants emerged to harness free and open source software communities," *Industrial and Corporate Change*, vol. 16, no. 5, pp. 913–943, 2007.

[46] J.-M. Dalle and P. M. David, "The allocation of software development resources in open source production mode," SIEPR Discussion Paper No. 02-27, Stanford Institute for Economic Policy Research, Stanford University, March 2003.

[47] S. Daniel, "Structure, cohesion, and open source software success," in *Proceedings of the 1st International Conference on Open Source Systems*, pp. 317–319, July 2005.

[48] P. A. David and J. S. Shapiro, "Community-based production of open-source software: What do we know about the developers who participate?," *Information Economics and Policy*, vol. 20, no. 4, pp. 364 – 398, 2008. Empirical Issues in Open Source Software.

[49] A. N. Dedeke, "Is Linux better than Windows software?," *IEEE Software*, vol. 26, pp. 104, 103, 2009.

[50] J. Dedrick and J. West, "Why firms adopt open source platforms: A grounded theory of innovation and standards adoption," in *MIS Quarterly Special Issue Workshop*, pp. 236–257, Dec. 2003.

[51] F. P. Deek and J. A. M. McHugh, *Open Source: Technology and Policy*. Cambridge: Cambridge University Press, 2008.

[52] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, "Who is an open source software developer?," *Communications of the ACM*, vol. 45, pp. 67–72, Feb. 2002.

[53] M. den Besten, J.-M. Dalle, and F. Galia, "The allocation of collaborative efforts in open-source software," *Information Economics and Policy*, vol. 20, pp. 316–322, Dec. 2008. Empirical Issues in Open Source Software.

[54] C. DiBona, S. Ockman, and M. Stone, eds., *Open Sources: Voices from the Open Source Revolution*. O'Reilly, 1999.

[55] Digital Equipment Computer Users Society, "Decus program library: PDP-11 catalog," Aug. 1978. Online `http://www.bitsavers.org/pdf/dec/decus/programCatalogs/DECUS_Catalog_PDP-11_Aug78.pdf`.

[56] T. T. Dinh-Trong and J. M. Bieman, "The FreeBSD project: A replication case study of open source development," *IEEE Transactions on Software Engineering*, vol. 31, pp. 481–494, June 2005.

[57] J. J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail," *Communications of the ACM*, vol. 30, no. 5, pp. 403–407, 1987.

[58] N. Duchneaut, "Socialization in an open source software community: A socio-technical analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, pp. 323–368, 2005.

[59] P. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional, first ed., 2007.

[60] N. Economides and E. Katsamakas, "Two-sided competition of proprietary vs. open source technology platforms and the implications for the software industry," *Management Science*, vol. 52, no. 7, pp. 1057–1071, 2006.

[61] M. S. Elliott and W. Scacchi, "Free software development: Cooperation and conflict in a virtual organizational culture," in *Free/Open Source Software Development*, (S. Koch, ed.), pp. 152–172, Hershey, PA: Idea Group Publishing, 2004.

[62] M. Elliott and W. Scacchi, "Mobilization of software developers: the free software movement," *Information Technology & People*, vol. 21, no. 1, pp. 4–33, 2008.

[63] A. Engelfriet, "Choosing an open source license," *IEEE Software*, vol. 27, pp. 48–49, Jan./Feb. 2010.

[64] N. Ensmenger, "Open source's lessons for historians," *IEEE Annals of the History of Computing*, vol. 26, pp. 104, 102–103, 2004.

[65] J. Erenkrantz, "Release management within open source projects," in *WOSSE '03: Proceedings of the 3rd Workshop on Open Source Software Engineering*, pp. 51–55, May 2003.

[66] J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*. Reading, MA: Addison-Wesley, 2001.

[67] J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds., *Perspectives on Free and Open Source Software*. Cambridge, MA: The MIT Press, 2005.

[68] J. Feller and B. Fitzgerald, "A framework analysis of the open source software development paradigm," in *ICIS '00: Proceedings of the 21st International Conference on Information Systems*, (Atlanta, GA), pp. 58–69, Association for Information Systems, 2000.

[69] C. Fershtman and N. Gandal, "Open source software: Motivation and restrictive licensing," *International Economics and Economic Policy*, vol. 4, no. 2, pp. 209–225, 2007.

[70] R. T. Fielding, "Shared leadership in the Apache project," *Communications of the ACM*, vol. 42, no. 4, pp. 42–43, 1999.

[71] B. Fitzgerald, "The transformation of open source software," *IEEE Transactions on Software Engineering*, vol. 30, pp. 587–598, September 2004.

[72] B. Fitzgerald, "The transformation of open source software," *MIS Quarterly*, vol. 30, pp. 587–598, Sep. 2006.

[73] K. Fogel, *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, Inc., 2005.

[74] S. Forge, "The rain forest and the rock garden: The economic impacts of open source software," *info*, vol. 8, no. 3, pp. 12–31, 2006.

[75] N. Franke and E. von Hippel, "Satisfying heterogenous user needs via innovation toolkits: The case of Apache security software," *Research Policy*, vol. 32, pp. 1199–1215, July 2003.

[76] Free Software Foundation, "Categories of free and nonfree software," Aug. 1996. Online `http://www.gnu.org/philosophy/categories.html`.

[77] A. Fuggetta, "Open source software—an evaluation," *Journal of Systems and Software*, vol. 66, no. 1, pp. 77–90, 2003.

[78] C. Gacek and B. Arief, "The many meanings of open source," *IEEE Software*, vol. 21, no. 1, pp. 34–40, 2004.

[79] C. Gacek, T. Lawrie, and B. Arief, "The many meanings of open source," Tech. Rep., University of Newcastle upon Tyne, 2001.

[80] M. J. Gallivan, "Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies," *Information Systems Journal*, vol. 11, no. 4, pp. 277–304, 2001.

[81] E. Gamma and K. Beck, *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Boston, MA: Addison-Wesley, 2004.

[82] B. Gates, "An open letter to hobbyists," *Homebrew Computer Club Newsletter*, vol. 2, p. 2, Jan. 1976.

[83] J. Gay, ed., *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Boston: GNU Press, Free Software Foundation, 2002.

[84] D. M. German, "Software engineering practices in the GNOME project," in *Perspectives on Free and Open Source Software*, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), pp. 211–226, Cambridge, MA: The MIT Press, 2005.

[85] D. M. German and A. E. Hassan, "License integration patterns: Addressing license mismatches in component-based development," in *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pp. 188–198, IEEE Computer Society, May 2009.

[86] D. M. German, M. D. Penta, Y.-G. Gueheneuc, and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications," in *MSR '09: Proceedings of the 6th International Workshop on Mining Software Repositories*, pp. 81–90, IEEE Computer Society Press, May 2009.

[87] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, (New York), pp. 29–43, ACM, 2003.

[88] R. A. Ghosh, "FM interview with Linus Torvalds: What motivates free software developers?," *First Monday*, vol. 3, March 1998.

[89] R. A. Ghosh, "Understanding free software developers: Findings from the FLOSS study," in *Perspectives on Free and Open Source Software*, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), pp. 23–46, Cambridge, MA: The MIT Press, 2005.

[90] P. Giuri, F. Rullani, and S. Torrisi, "Explaining leadership in virtual teams: The case of open source software," *Information Economics and Policy*, vol. 20, pp. 305–315, Dec. 2008. Empirical Issues in Open Source Software.

[91] E. Glynn, B. Fitzgerald, and C. Exton, "Commercial adoption of open source software: An empirical study," in *The International Symposium on Empirical Software Engineering*, Nov. 2005.

[92] M. Godfrey and Q. Tu, "Growth, evolution, and structural change in open source software," in *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, (New York), pp. 103–106, ACM, 2001.

[93] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study," in *Conference on Software Maintenance*, (Piscataway, NJ,USA), pp. 131–142, IEEE, 2000.

[94] R. Goldman and R. P. Gabriel, *Innovation Happens Elsewhere: Open Source as a Business Strategy*. San Francisco: Morgan Kaufmann, Elsevier, Apr. 2005.

[95] R. Grewal, G. Lilien, and G. Mallapragada, "Location, location, location: How network embeddedness affects project success in open source systems," *Management Science*, vol. 52, no. 7, pp. 1043–1056, 2006.

[96] M. Gruber and J. Henkel, "New ventures based on open innovation – an empirical analysis of start-up firms in embedded Linux," *International Journal of Technology Management*, vol. 33, no. 4, pp. 356–372, 2006.

[97] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, pp. 897–910, Oct. 2005.

[98] S. Haefliger, G. von Krogh, and S. Spaeth, "Code reuse in open source software," *Management Science*, vol. 54, pp. 180–153, Jan. 2008.

[99] J. Hahn, J. Moon, and C. Zhang, "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties," *Information Systems Research*, vol. 19, pp. 369–391, Sep. 2008.

[100] J. Hamerly, T. Paquin, and S. Walton, "Freeing the source. the story of Mozilla," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[101] D. Harhoff, J. Henkel, and E. von Hippel, "Profiting from voluntary information spillovers: How users benefit by freely revealing their innovations," *Research Policy*, vol. 32, pp. 1753–1769, Dec. 2003.

[102] A. Hars and S. Ou, "Working for free? Motivations for participating in open source projects," in *Proceedings of the 34th Hawaii International Conference on System Sciences*, (Hawaii), June 2001.

[103] F. Hecker, "Setting up shop: The business of open-source software," *IEEE Software*, vol. 16, pp. 45–51, Jan./Feb. 1999.

[104] H. Hedberg, N. Iivari, M. Rajanen, and L. Harjumaa, "Assuring quality and usability in open soruce software development," *Emerging Trends in FLOSS Research and Development, International Workshop on*, vol. 0, p. 2, 2007.

[105] D. Hedgebeth, "Gaining competitive advantage in a knowledge-based economy through the utilization of open source software," *VINE*, vol. 37, no. 3, pp. 284–294, 2007.

[106] J. Henkel, "Selective revealing in open innovation processes: The case of embedded Linux," *Research Policy*, vol. 35, no. 7, pp. 953–969, 2006.

[107] J. Henkel, "Champions of revealing—the role of open source developers in commercial firms," *Industrial and Corporate Change*, vol. 18, no. 3, pp. 435–471, 2009. Published by Oxford University Press on behalf of Associazione ICC.

[108] J. Herbsleb, A. Mockus, T. Finholt, and R. Grinter, "Distance, dependencies, and delay in a global collaboration," in *CSCW '00: Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, (New York), pp. 319–328, ACM, 2000.

[109] G. Hertel, S. Niedner, and S. Hermann, "Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel," *Research Policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[110] E. v. Hippel, "Economics of product development by users: The impact of "sticky" local information," *Management Science*, vol. 44, no. 5, pp. 629–644, 1998.

[111] M. Huang, L. Yang, and Y. Yang, "A development process for building OSS-based applications," in *Unifying the Software Process Spectrum (2005), Lecture Notes in Computer Science 3840*, (B. B. M. Li and L. Osterweil, eds.), pp. 122–135, Springer-Verlag Berlin Heidelberg, 2005.

[112] A. Hunt and D. Thomas, "Software archaeology," *Software, IEEE*, vol. 19, pp. 20–22, Mar/Apr 2002.

[113] A. Israeli and D. G. Feitelson, "The Linux kernel as a case study in software evolution," *Journal of Systems and Software*, vol. 83, no. 3, pp. 485–501, 2010.

[114] C. Jensen and W. Scacchi, "Role migration and advancement processes in ossd projects: A comparative case study," *International Conference on Software Engineering*, vol. 0, pp. 364–374, 2007.

[115] M. John, *What the Dormhouse Said: How the 60s Counterculture Shaped the Personal Computer*. New York: Viking Adult, 2005.

[116] J. P. Johnson, "Open source software: Private provision of a public good," *Journal of Economics & Management Strategy*, vol. 11, pp. 637–662, Dec. 2002.

[117] N. Jørgensen, "Putting it all in the trunk: Incremental software development in the FreeBSD open source project," *Information Systems Journal*, vol. 11, pp. 321–336, Oct. 2001.

[118] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance and Evolution*, vol. 19, pp. 77–131, Mar 2007.

[119] S. H. Kan, *Metrics and Models in Software Quality Engineering (2nd Edition)*. Addison-Wesley, 2003.

[120] W. Ke and P. Zhang, "The effects of extrinsic motivations and satisfaction in open source software development," *Journal of the Association for Information Systems*, vol. 11, pp. 784–808, Dec. 2010.

[121] M. Kechagia, D. Spinellis, and S. Androutsellis-Theotokis, "Open source licensing across package dependencies," in *PCI '10: 14th Panhellenic Conference on Informatics*, pp. 27–32, IEEE Computer Society, Sep. 2010.

[122] B. W. Kernighan and P. J. Plauger, *Software Tools*. Reading, MA: Addison-Wesley, 1976.

[123] D. Y. Kim, J. B. Kim, and S. Y. Rhew, "Effective reuse procedure for open source software," in *Software Engineering Research and Practice*, pp. 163–167, CSREA Press, 2006.

[124] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *Software Engineering, IEEE Transactions on*, vol. 28, pp. 721–734, Aug 2002.

[125] D. Knuth, *The TeXbook*. Reading, MA: Addison-Wesley, 1984.

[126] S. Koch, "Profiling an open source project ecology and its programmers," *Electronic Markets*, vol. 14, pp. 77–88(12), June 2004.

[127] S. Koch, "Effort modeling and programmer participation in open source software projects," *Information Economics and Policy*, vol. 20, no. 4, pp. 345 – 355, 2008. Empirical Issues in Open Source Software.

[128] S. Koch and G. Schneider, "Effort, co-operation and co-ordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.

[129] B. Kogut and A. Metiu, "Open-source software development and distributed innovation," *Oxford Review of Economic Policy*, vol. 17, no. 2, pp. 248–264, 2001.

[130] P. Kollock and M. Smith, "Managing the virtual commons: Cooperation and conflict in computer communities," in *Computer-Mediated Communication: Linguistic, Social, and Cross-Cultural Perspectives*, (S. Herring, ed.), pp. 109–128, John Benjamins Publishing, 1996.

[131] D. Kozlov, J. Koskinen, M. Sakkinen, and J. Markkula, "Assessing maintainability change over multiple software releases," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 1, pp. 31–58, 2008.

[132] K. L. Kraemer, J. Dedrick, and P. Sharma, "One laptop per child: Vision vs. reality," *Communications of the ACM*, vol. 52, no. 6, pp. 66–73, 2009.

[133] S. Krishnamurthy, "Cave or community?: An empirical examination of 100 mature open source projects," *First Monday*, vol. 7, June 2002.

[134] S. Krishnamurthy, "A managerial overview of open source software," *Business Horizons*, vol. 46, pp. 47–56, Sep./Oct. 2003.

[135] S. Krishnamurthy, "An analysis of open source business models," in *Perspectives on Free and Open Source Software*, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), pp. 279–296, Cambridge, MA: The MIT Press, 2005.

[136] S. Krishnamurthy, "On the intrinsic and extrinsic motivation of free/libre/open source (FLOSS) developers," *Knowledge, Technology & Policy*, vol. 18, pp. 17–39, Dec. 2006.

[137] K. R. Lakhani and E. von Hippel, "How open source software works: 'Free' user-to-user assistance," *Research Policy*, vol. 32, pp. 923–943, June 2003.

[138] K. R. Lakhani and R. G. Wolf, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," in *Perspectives on Free and Open Source Software*, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), pp. 3–22, Cambridge, MA: The MIT Press, 2005.

[139] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer Verlag, 2006.

[140] A. M. S. Laurent, *Understanding Open Source and Free Software Licensing*. Cambridge, Massachusetts: O'Reilly, 2004.

[141] L. Lawrence, *Free Culture: How Big Media Uses Technology and the Law to Lock Down Culture and Control Creativity*. New York: Penguin Group Inc., 2004.

[142] D. Lee and H. Mendelson, "Divide and conquer: Competing with free technology under network effects," *Production and Operations Management*, vol. 17, no. 1, pp. 12–28, 2008.

[143] S.-Y. T. Lee, H.-W. Kim, and S. Gupta, "Measuring open source software success," *Omega*, vol. 37, no. 2, pp. 426 – 438, 2009.

[144] S. Leffler, M. McKusick, M. Karels, and J. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*. Reading, MA: Addison-Wesley, 1989.

[145] J. Lerner and J. Tirole, "The open source movement: Key research questions," *European Economic Review*, vol. 45, pp. 819–826, May 2001.

[146] J. Lerner and J. Tirole, "Some simple economics of open source," *The Journal of Industrial Economics*, vol. 50, pp. 197–234, June 2002.

[147] J. Lerner and J. Tirole, "Economic perspectives on open source," in *Intellectual Property and Entrepreneurship*, pp. 33–69, Emerald Group Publishing Limited, 2004.

[148] J. Lerner and J. Tirole, "The scope of open source licensing," *The Journal of Law, Economics and Organization*, vol. 21, no. 1, pp. 20–56, 2005.

[149] L. Lessig, "Open code and open societies," in *Perspectives on Free and Open Source Software*, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), pp. 349–360, Cambridge, MA: The MIT Press, 2005.

[150] J. Lions, *Lions' Commentary on UNIX 6th Edition with Source Code*. San Jose, CA: Peer-to-Peer Communications, Inc., 1996.

[151] M. W. Losh, "An overview of FreeBSD/mips," in *AsiaBSDCon 2009*, March 2009. Online `http://2009.asiabsdcon.org/papers/abc2009-P4B-paper.pdf`. Current September 2010.

[152] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Science*, vol. 52, pp. 1015–1030, July 2006.

[153] T. R. Madanmohan and R. De', "Open source reuse in commercial firms," *IEEE Software*, vol. 21, pp. 62–69, Nov./Dec. 2004.

[154] M. L. Markus, B. Manville, and C. E. Agres, "What makes a virtual organization work: Lessons from the open-source world," *MIT Sloan Management Review*, vol. 42, pp. 13–26, Fall 2000.

[155] J. Martinez-Romo, G. Robles, J. M. Gonzalez-Barahona, and M. Ortuño-Perez, *Open Source Development, Communities and Quality*, ch. Using Social Network Analysis Techniques to Study Collaboration between a FLOSS Community and a Company, pp. 143–158. Vol. 275 of *IFIP International Federation for Information Processing*, Springer Boston, July 2008.

[156] J. Mateos-Garcia and W. E. Steinmueller, "The institutions of open source software: Examining the Debian community," *Information Economics and Policy*, vol. 20, pp. 333–344, Dec. 2008. Empirical Issues in Open Source Software.

[157] S. McConnell, "Open-source methodology: Ready for prime time?," *IEEE Software*, vol. 16, pp. 6–11, July/Aug. 1999.

[158] M. K. McKusick, "Twenty years of Berkeley Unix: From AT&T-owned to freely redistributable," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[159] C. Melian and M. Mahring, "Lost and gained in translation: Adoption of open source software development at Hewlett-Packard," in *Open Source Development, Communities and Quality*, pp. 93–104, Boston: Springer, 2008.

[160] A. Meneely and L. Williams, "Secure open source collaboration: An empirical study of Linus' law," in *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, (New York), pp. 453–462, ACM, 2009.

[161] A. Mockus, "Amassing and indexing a large sample of version control systems: Towards the census of public source code history," in *MSR '09: Proceedings of the 6th IEEE Intl. Working Conference on Mining Software Repositories*, (M. W. Godfrey and J. Whitehead, eds.), pp. 11–20, 2009.

[162] A. Mockus, R. Fielding, and J. Herbsleb, "A case study of open source software development: The Apache server," in *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, (New York), pp. 263–272, ACM, 2000.

[163] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[164] L. Morgan and P. Finnegan, "How perceptions of open source software influence adoption: An exploratory study," in *ECIS '07: The 15th European Conference on Information Systems*, (W. R. Osterle H, Schelp J, ed.), pp. 973–984, 2007.

[165] B. Mukerji, V. Kumar, and U. Kumar, "The challenges of adopting open source software in promoting e-government," in *ICEG '06: International Conference on E-Governance*, pp. 22–31, 2006.

[166] N. Munga, T. Fogwill, and Q. Williams, "The adoption of open source software in business models: A Red Hat and IBM case study," in *SAICSIT '09: Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, (New York), pp. 112–121, ACM, 2009.

[167] E. D. Mynatt, A. Adler, M. Ito, and V. L. O'Day, "Design for network communities," in *CHI '97: The 1997 Conference on Human Factors in Computing Systems*, pp. 210–217, March 1997.

[168] N. Nagappan, "Potential of open source systems as project repositories for empirical studies working group results," in *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, pp. 103–107, Springer Verlag, 2007. Lecture Notes in Computer Science 4336.

[169] D. Nagy, A. M. Yassin, and A. Bhattacherjee, "Organizational adoption of open source software: Barriers and remedies," *Communications of the ACM*, vol. 53, no. 3, pp. 148–151, 2010.

[170] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution*, (Orlando, FL), May 2002.

[171] D. M. Nichols and M. B. Twidale, "The usability of open source software," *First Monday*, vol. 8, Jan. 2003.

[172] O. Nov, "What motivates Wikipedians?," *Communications of the ACM*, vol. 50, pp. 60–64, Nov. 2007.

[173] W. Oh and S. Jeon, "Membership herding and network stability in the open source community: The ising perspective," *Management science*, vol. 53, pp. 1086–1101, July 2007.

[174] S. O'Mahony and F. Ferraro, "The emergence of governance in an open source community," *Academy of Management Journal*, vol. 50, no. 5, pp. 1079–1106, 2007.

[175] T. O'Reilly, "Lessons from open-source software development," *Communications of the ACM*, vol. 42, pp. 32–73, Apr. 1999.

[176] W. Orman, "Giving it away for free? the nature of job-market signaling by open-source software developers," *The BE Journal of Economic Analysis & Policy*, vol. 8, no. 1, 2008.

[177] B. O'Sullivan, "Making sense of revision-control systems," *Communications of the ACM*, vol. 52, pp. 56–62, Sep. 2009.

[178] D. L. Parnas, "Software aging," in *Proceedings of the 16th international conference on Software engineering*, (Los Alamitos, CA, USA), pp. 279–287, IEEE Computer Society Press, 1994.

[179] J. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE Transactions on Software Engineering*, vol. 30, pp. 246–256, April 2004.

[180] M. D. Penta, D. M. German, Y.-G. Gueheneuc, and G. Antoniol, "An exploratory study of the evolution of software licensing," in *ICSE '10: Proceedings of the 32nd International Conference on Software Engineering*, ACM Press, May 2010.

[181] B. Perens, "The open source definition," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[182] D. E. Perry, A. A. Porter, and L. G. Votta, "Empirical studies of software engineering: a roadmap," in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, (New York, NY, USA), pp. 345–355, ACM, 2000.

[183] R. Purushothaman and D. Perry, "Toward understanding the rhetoric of small source code changes," *IEEE Transactions on Software Engineering*, vol. 31, pp. 511–526, June 2005.

[184] J. S. Quarterman and J. C. Hoskins, "Notable computer networks," *Communications of the ACM*, vol. 29, pp. 932–971, Oct. 1986.

[185] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA: O' Reilly and Associates, 2001.

[186] E. S. Raymond, "A brief history of hackerdom," Online `http://catb.org/~esr/writings/cathedral-bazaar/hacker-history/`, 2000.

[187] E. S. Raymond, "The magic cauldron," Online `http://www.sfu.ca/oldlidc/LMSSC/documents/other%20related%20documents/magic-cauldron.pdf`, 2000.

[188] D. M. Ritchie and K. Thompson, "The unix time-sharing system," *Commun. ACM*, vol. 17, pp. 365–375, July 1974.

[189] J. E. Robbins, "Adopting open source software engineering (OSSE) practices by adopting OSSE tools," in *Perspectives on Free and Open Source Software*, (J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, eds.), pp. 245–264, Cambridge, MA: The MIT Press, 2005.

[190] J. A. Roberts, I.-H. Hann, and S. A. Slaughter, "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects," *Management Science*, vol. 52, pp. 984–999, July 2006.

[191] G. Robles, S. Dueñas, and J. Gonzalez-Barahona, "Corporate involvement of libre software: Study of presence in Debian code over time," in *Open Source Development, Adoption and Innovation*, pp. 121–132, Springer Verlag, 2007. IFIP International Federation for Information Processing Volume 234.

[192] G. Robles, *Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results*. PhD thesis, Universidad Rey Juan Carlos, Madrid, 2005.

[193] G. Robles, J. M. Gonzalez-Barahona, and J. J. Merelo, "Beyond source code: The importance of other artifacts in software development (a case study)," *Journal of Systems and Software*, vol. 79, no. 9, pp. 1233 – 1248, 2006. Selected papers from the fourth Source Code Analysis and Manipulation (SCAM 2004) Workshop.

[194] D. S. H. Rosenthal, T. Robertson, T. Lipkis, V. Reich, and S. Morabito, "Requirements for digital preservation systems: A bottom-up approach," *D-Lib Magazine*, vol. 11, Nov. 2005.

[195] M. Rounds, "IBM saw 'limited' software industry," *Software*, vol. 9, pp. 37–40, March 1989.

[196] M. Ruffin and C. Ebert, "Using open source software in product development: A primer," *IEEE Software*, vol. 21, no. 1, pp. 82–86, 2004.

[197] B. M. Sadowski, G. Sadowski-Rasters, and G. Duysters, "Transition of governance in a mature open software source community: Evidence from the Debian case," *Information Economics and Policy*, vol. 20, pp. 323–332, Dec. 2008. Empirical Issues in Open Source Software.

[198] P. H. Salus, *A Quarter Century of UNIX*. Boston, MA: Addison-Wesley, 1994.

[199] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, "The SQO-OSS quality model: Measurement based open source software evaluation," in *OSS '08: 4th International Conference on Open Source Systems: Open Source Development, Communities and Quality*, (E. Damiani and G. Succi, eds.), (Boston), pp. 237–248, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, Springer, Sep. 2008.

[200] I. Samoladas, I. Stamelos, L. Angelis, and A. Oikonomou, "Open source software development should strive for even greater code maintainability," *Communications of the ACM*, vol. 47, no. 10, pp. 83–87, 2004.

[201] W. Scacchi, "Understanding requirements for developing open source software systems," *IEE Proceedings—Software*, vol. 149, no. 1, pp. 24–39, 2002.

[202] W. Scacchi, "Free and open source development practices in the game community," *IEEE Software*, vol. 21, pp. 59–66, 2004.

[203] R. Sen, C. Subramaniam, and M. L. Nelson, "Determinants of the choice of open source software license," *Journal of Management Information Systems*, vol. 25, no. 3, pp. 207–239, 2009.

[204] S. K. Shah, "Motivation, governance, and the viability of hybrid forms in open source software development," *Management Science*, vol. 52, pp. 1000–1014, July 2006.

[205] S. Sharma, V. Sugumaran, and B. Rajagopalan, "A framework for creating hybrid-OSS communities," *Information Systems Journal*, vol. 12, no. 1, pp. 7–25, 2002.

[206] P. V. Singh, "The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, pp. 6:1–6:27, September 2010.

[207] D. Spiller and T. Wichmann, "FLOSS final report part 3. (FLOSS) free/libre open source software: Survey and study. Basics of open source software markets and business models," Tech. Rep. IST-2000-4.1.1, Berlecon Research, July 2002.

[208] D. Spinellis, *Code Reading: The Open Source Perspective*. Addison Wesley Professional, 2003.

[209] D. Spinellis, "Version control systems," *IEEE Software*, vol. 22, pp. 108–109, Sep./Oct. 2005.

[210] D. Spinellis, *Code Quality: The Open Source Perspective*. Boston, MA: Addison-Wesley, 2006.

[211] D. Spinellis, "Future CS course already here," *Communications of the ACM*, vol. 49, no. 8, p. 13, 2006.

[212] D. Spinellis, "Global software development in the FreeBSD project," in *International Workshop on Global Software Development for the Practitioner*, (P. Kruchten, Y. Hsieh, E. MacGregor, D. Moitra, and W. Strigel, eds.), pp. 73–79, ACM Press, May 2006.

[213] D. Spinellis, "Open source and professional advancement," *IEEE Software*, vol. 23, pp. 70–71, Sep./Oct. 2006.

[214] D. Spinellis, "A tale of four kernels," in *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, (W. Schäfer, M. B. Dwyer, and V. Gruhn, eds.), (New York), pp. 381–390, Association for Computing Machinery, May 2008.

[215] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P. J. Adams, I. Samoladas, and I. Stamelos, "Evaluating the quality of open source software," in *SQM '08: 2nd International Workshop on Software Quality and Maintainability*, pp. 5–28, The Reengi-

neering Forum, Apr. 2008. Electronic Notes in Theoretical Computer Science Volume 233 (March 2009).

[216] D. Spinellis and C. Szyperski, "How is open source affecting software development?," *IEEE Software*, vol. 21, pp. 28–33, Jan./Feb. 2004. Guest Editors' Introduction: Developing with Open Source Software.

[217] R. Stallman, "Why upgrade to GPLv3," 2007. Online `http://www.gnu.org/licenses/rms-why-gplv3.html`.

[218] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open source software development," *Information Systems Journal*, vol. 12, no. 1, pp. 43–60, 2002.

[219] K. Staring, O. Titlestad, and J. Gailis, "Educational transformation through open source approaches," in *IRIS '05: Proceedings of the 28th Information Systems Research Seminar in Scandinavia*, Apr. 2005.

[220] J. Stark, "Peer reviews as a quality management technique in open-source software development projects," in *Proceedings of the 7th International Conference on Software Quality*, (London, UK, UK), pp. 340–350, Springer-Verlag, 2002.

[221] K. J. Stewart and S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *MIS Quarterly*, vol. 30, pp. 291–314, June 2006.

[222] K.-J. Stol and M. Ali Babar, "Challenges in using open source software in product development: A review of the literature," in *International Conference on Software Engineering*, pp. 17–22, 2010.

[223] K.-J. Stol, M. A. Babar, B. Russo, and B. Fitzgerald, "The use of empirical methods in open source software research: Facts, trends and future directions," in *FLOSS '09: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, (Washington, DC), pp. 19–24, IEEE Computer Society, 2009.

[224] A. S. Tanenbaum, *Operating Systems: Design and Implementation*. Englewood Cliffs, NJ: Prentice Hall, 1987.

[225] M. Tiemann, "Future of Cygnus solutions: An entrepreneur's account," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[226] L. Torvalds, "The Linux edge," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[227] L. Torvalds and D. Diamond, *Just for Fun: The Story of an Accidental Revolutionary*. Harper Collins, May 2001.

[228] M. Umarji, S. Sim, and C. Lopes, "Archetypal internet-scale source code searching," in *Open Source Development, Communities and Quality: IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software*, pp. 257–263, IFIP: International Federation for Information Processing, Springer, Sep. 2008.

[229] S. Valverde, G. Theraulaz, J. Gautrais, V. Fourcassie, and R. Sole, "Self-organization patterns in wasp and open source communities," *Intelligent Systems, IEEE*, vol. 21, pp. 36–40, March-April 2006.

[230] K. Ven, J. Verelst, and H. Mannaert, "Should you adopt open source software?," *IEEE Software*, vol. 25, no. 3, pp. 54–59, 2008.

[231] P. Vixie, "Software engineering," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[232] E. von Hippel, "Innovation by user communities: Learning from open source software," *MIT Sloan Management Review*, vol. 42, pp. 82–86, Summer 2001.

[233] E. von Hippel, "Democratizing innovation: The evolving phenomenon of user innovation," *Journal für Betriebswirtschaft*, vol. 55, pp. 63–78, March 2005.

[234] E. von Hippel, "Horizontal innovation networks -by and for users," *Industrial and Corporate Change*, pp. 1–23, May 2007.

[235] E. von Hippel and G. von Krogh, "Open source software and the 'private-collective' innovation model: Issues for organization science," *Organization Science*, vol. 14, pp. 209–223, March/Apr. 2003.

[236] E. von Hippel and G. von Krogh, "Free revealing and the private-collective model for innovation incentives," *R & D Management*, vol. 36, pp. 295–306, June 2006.

[237] G. von Krogh and S. Spaeth, "The open source software phenomenon: Characteristics that promote research," *The Journal of Strategic Information Systems*, vol. 16, no. 3, pp. 236–253, 2007.

[238] G. von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: A case study," *Research Policy*, vol. 32, pp. 1217–1241, July 2003.

[239] G. von Krogh and E. von Hippel, "Special issue on open source software development," *Research Policy*, vol. 32, pp. 1149–1157, July 2003.

[240] G. von Krogh and E. von Hippel, "The promise of research on open source software," *Management Science*, vol. 52, pp. 975–983, July 2006.

[241] H. Wang and C. Wang, "Open source software adoption: A status report," *IEEE Software*, vol. 18, pp. 90–95, March/Apr. 2001.

[242] T. Waring and P. Maddocks, "Open source software implementation in the UK public sector: Evidence from the field and implications for the future," *International Journal of Information Management*, vol. 25, no. 5, pp. 411–428, 2005.

[243] A. I. Wasserman, "Building a business on open source software," in *Proceedings of Conference on Technological Entrepreneurship*, Edward Elgar, 2009. To appear.

[244] R. T. Watson, M.-C. Boudreau, P. T. York, M. E. Greiner, and D. W. Jr., "The business of open source," *Communications of the ACM*, vol. 51, pp. 41–46, Apr. 2008.

[245] S. Weber, *The Success of Open Source*. Harvard University Press, Oct. 2005.

[246] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, (Washington, DC, USA), IEEE Computer Society, 2007.

[247] J. West and J. Dedrick, "Scope and timing of deployment: moderators of organizational adoption of the linux server platform," *International Journal of IT Standards and Standardization Research*, vol. 4, pp. 1–37, July 2006.

[248] J. West, "How open is open enough? Melding proprietary and open source platform strategies," *Research Policy*, vol. 32, pp. 1259–1285, July 2003.

[249] D. A. Wheeler, "The free-libre/open source software (FLOSS) license slide," Online `http://www.dwheeler.com/essays/floss-license-slide.pdf`, Sep. 2007.

[250] M. Wijnen-Meijer and R. Batenburg, "To open source or not to open source: That's the strategic question," in *ECIS '07: Proceedings of the 15th European Conference on Information Systems*, pp. 1019–1030, 2007.

[251] M.-W. Wu and Y.-D. Lin, "Open source software development: An overview," *IEEE Computer*, vol. 34, pp. 33–38, Jan. 2001.

[252] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, "Data mining for software engineering," *Computer*, vol. 42, pp. 55–62, 2009.

[253] J. Xu, Y. Gao, S. Christley, and G. Madey, "A topological analysis of the open souce software development community," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, Jan. 2005.

[254] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *ICSE '03: Proceedings of 25th International Conference on Software Engineering*, (Portland, OR), pp. 419–429, May 2003.

[255] R. Young, "Giving it away: How Red Hat software stumbled across a new economic model and helped improve an industry," in *Open Sources: Voices from the Open Source Revolution*, (C. DiBona, S. Ockman, and M. Stone, eds.), O'Reilly, 1999.

[256] L. Yu, S. R. Schach, K. Chen, G. Z. Heller, and J. Offutt, "Maintainability of the kernels of open-source operating systems: A comparison of linux with freebsd, netbsd, and openbsd," *Journal of Systems and Software*, vol. 79, no. 6, pp. 807 – 815, 2006.

[257] D. Zeitlyn, "Gift economies in the development of open source software: Anthropological reflections," *Research Policy*, vol. 32, pp. 1287–1291, July 2003.

[258] W. Zhang and J. Storck, "Peripheral members in online communities," in *AMCIS '01: Proceedings of the 7th America's Conference on Information Systems*, (Boston, Massachusetts), 2001.

Id: article.tex,v 1.230 2011/07/04 05:17:39 dds Exp