Handbook of Science and Technology Convergence
DOI 10.1007/978-3-319-04033-2_29-1

Open Source Technology Development

Kevin Crowston*
School of Information Studies, Syracuse University, Syracuse, NY, USA

Abstract

In this chapter, we introduce the practices of free/libre open source software (FLOSS) development as an instance of the convergence of technological affordances with novel social practices to create a novel mode of work. We then consider how FLOSS software might be used for various scientific applications, perhaps leading to a convergence of current distinct disciplines. We conclude by considering how the technologies and practices of FLOSS development might be applied to other settings, thus leading to further convergence of those settings.

Keywords

Free/libre open source software

Introduction

Free/libre open source software (FLOSS) is an umbrella term that covers a diversity of kinds of software and approaches to development. We therefore start this chapter by clarifying its focus. Technically, the term free software or open source software refers to software released under a license that permits the inspection, use, modification, and redistribution of the software's source code. FLOSS can thus be characterized as a privately produced public good (O'Mahony 2003). The freedom to reuse the source code distinguishes FLOSS from other forms of software distribution, such as freeware, where a program is provided for free, but without the source code being available, or licenses that make source code available for inspection but without the right to reuse it (e.g., the Microsoft Shared Source Initiative).

The term FLOSS groups together free software (sometimes referred to as "libre software" to avoid the potential confusion between the intended meaning of free meaning freedom and free meaning at no cost) and open source software. However, there are clear and important differences in the motivation for this approach to software development between the two groups, and the differences are sometimes controversial (Kelty 2008). (See Free Software Foundation (2013) for a detailed definition of free software and Open Source Initiative (n.d.) for a detailed definition of open source software.)

Free software proponents view the ability to view and modify computer source code as a matter of freedom, a basic human right. They thus view proprietary software as immoral, since proprietary licenses cut users off from something they should have. Typical free software licenses (e.g., the Gnu Public Licence or GPL) enforce sharing of source code (requiring it be available to users) and further require that modifications to the source code also be released as open source. This "viral" property means that free software cannot be freely mixed with proprietary software without also freeing that software (though there are technical workarounds).

*Email: crowston@syr.edu

DOI 10.1007/978-3-319-04033-2_29-1

In contrast, developers who adopt the term *open source* view sharing source code not as a moral issue but pragmatically as representing a better approach to development, by enabling broader participation in development thus improving bug detection, code security, and project productivity. Open source licenses (e.g., Apache or BSD licenses) typically allow licensed code to be used in combination with code with proprietary licenses, thus enabling commercial use of open source software. Similar tensions between moral and pragmatic motivations for openness can be observed in other kinds of open movements, as will be discussed later in this chapter.

FLOSS-licensed software may be developed in the same way as proprietary software, e.g., as in the case of MySQL, which was developed by a company and is now owned by Oracle, but available for use under an open source license. The open source license allows reuse of the code in other settings, for novel purposes and even as the basis for competing projects (called forks): in fact, MySQL now has a community-based fork, MariaDB. However, most FLOSS is developed not by a single company but through an open community process. And despite the differences in philosophy, developers in both camps generally adopt similar open development methodologies, making it sensible to speak of FLOSS development practices.

In some projects, a focal organization may lead the project (Fitzgerald 2006). For example, the Mozilla Foundation leads development of the Firefox browser and Thunderbird mail client, the Apache Software Foundation directs development of Apache software (e.g., the Apache httpd web server among many others), and the MariaDB Foundation oversees development of MariaDB. A few of these foundations may have resources to employ programmers to work on projects. However, much of the work done by such foundations addresses issues other than development, e.g., maintaining infrastructure, providing a legal framework for the development, providing support to project leaders, or overseeing the general project development approach.

Many FLOSS projects exist outside of any formal organizational structure. Even where there is some kind of organizational involvement, an important feature of the FLOSS development process is that many (or most) developers contribute to projects as volunteers, without direct remuneration from the project, in what has been described as a community-based development (Lee and Cole 2003). Instead of pay, developers are motivated intrinsically by interest in the project or the chance to work on a project of one's own choosing or extrinsically by the opportunity to gain reputation and develop skills.

Recent years have seen an increase in the participation of firms in FLOSS and so in contribution from employees paid to work on FLOSS projects (Lakhani and Wolf 2005) and whose contributions are made available to the wider community (Henkel 2006). However, even employed developers are not paid directly by the projects to which they contribute, so from the point of view of the project, they are still volunteers that have to be motivated (though motivations may be for the corporations providing developers as well as for individuals volunteering their time).

Because of these organizational factors, the teams developing FLOSS are often organizationally and geographically distributed, forming almost purely virtual teams. The teams have a high isolation index (O'Leary and Cummings 2007) in that many team members work on their own and in most cases for different organizations (or no organization at all). Developers contribute from around the world and meet face-to-face infrequently if at all (Raymond 1998; Wayner 2000).

For most community-based FLOSS teams, distributed work is not an alternative to face-to-face: it is the only a feasible mode of interaction. As a result, these teams depend on processes that span traditional boundaries of place and ownership. To span distance, the work of the team coordinated primarily by means of computer-mediated communications (CMC). Often sets of tools are provided together in a "forge," named after the first such, SourceForge, e.g., discussion groups, source code control, bug trackers, issue trackers, and file distribution. Particularly important are systems to share source code and track revisions. These system provide access to the code to anyone but control who can make changes

DOI 10.1007/978-3-319-04033-2_29-1

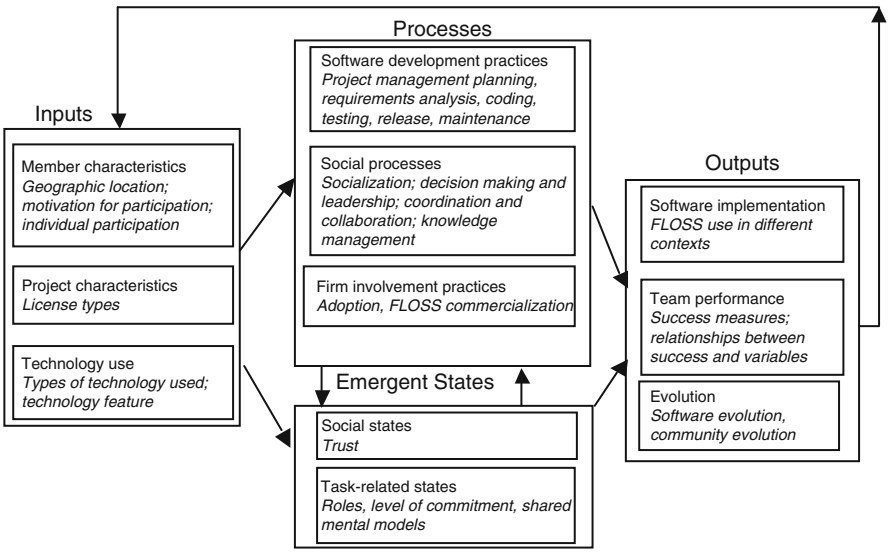

Fig. 1 Constructs and relations studied in FLOSS research (Figure 6 from Crowston et al. (2012)) © ACM. Used with permission. doi:10.1145/2089125.2089127

to the released project codebase (e.g., the code that is distributed to end users either in source format or as a ready-to-run binary distribution).

Another common feature of FLOSS projects that can be traced to reliance on volunteer contributors is the skewed distribution of activity across developers. As many developers contribute in their spare time, the amount of time each has available to work on the project varies. Indeed, it can be difficult to say exactly how many contributors a project has, since the difference between "temporarily inactive" and "dropped out" is not always clear.

Developers also vary in what kinds of activity they contribute. The result is a project with an onion structure. Most project have a core of developers with "commit" rights, meaning that they can contribute directly to the code base stored in the source code control system, as well as a set of codevelopers who may write some code, but whose contributions are reviewed by core members before being accepted. An active project will also have a large set of active users who contribute bug reports or other supporting functions such as documentation, translations, or user support.

Community-developed FLOSS thus represents a different approach to innovation in the software industry. The research literature on software development and on distributed work emphasizes the difficulties of distributed software development, but successful community-based FLOSS development presents an intriguing counterexample. Characterized by a globally distributed developer force, a rapid, reliable software development process and a diversity of tools to support distributed collaborative development, effective FLOSS development teams somehow profit from the advantages and overcome the challenges of distributed work (Alho and Sulonen 1998).

Research has identified a number of factors relating to the success of FLOSS development project. Crowston et al. (2012) surveyed the literature to summarize these. Figure 1 shows the resulting framework, with the major concepts that identified in the FLOSS research papers, organized in an *inputs-mediators-outputs-inputs (IMOI) framework.*

Inputs represent the starting conditions of a team, which for FLOSS include member characteristics and project characteristics such as license choice and technology features. For example, developer motivations to contribute are key to the success of projects that rely on voluntary contributions. While a few projects have more volunteers than they can use, most projects have more work to do than volunteers to do

DOI 10.1007/978-3-319-04033-2_29-1

it, so attracting and retaining volunteer developers is critical for success. Similarly, firm interest is necessary to attract developers paid by their companies.

Mediators represent factors that mediate the impact of inputs on outputs. Mediators can be further divided into two categories: processes and emergent states. *Processes* represent dynamic interactions among team members as they work on their projects, leading to the outputs. For FLOSS development processes include both software development practices and team social processes.

In terms of the *software processes* adopted, FLOSS development largely resembles any other software development project. However, some software processes do seem to be done in a different fashion in FLOSS projects than in conventional projects. For example, researchers have suggested that requirement analysis is done differently, as requirements are often identified based on interaction with current version and feedback from users rather than from a formal requirement gathering process.

In addition to the development practices, the *social and interpersonal processes* that support the distributed team are also critical. For example, an important social process in an open project is the socialization of new members, as they move from new participants to full members of the community. However, in most open source projects studied to date, the onus of socialization falls on the would-be participant, as few projects have any kind of organized socialization processes, likely due to the general reliance on volunteer developers who do not have time for (or much interest in) socializing newcomers, the vast majority of whom will not turn out to be core contributors (recall the skewed distribution of efforts).

Similarly, studies have found an important role for project leadership, described as "providing a vision, attracting developers to the project, and keeping the project together and preventing forking" (Giuri et al. 2008). Leadership is important because there is low organizational control of volunteer members, and the threat of "forking," while uncommon and discouraged, limits the ability of project leaders to discipline members. In most FLOSS projects, leadership is emergent, meaning that rather being appointed, leaders emerge from the group based on contributions, and often shared, as multiple members of the project contribute to leadership.

The second kind of mediator between inputs and outputs are *emergent states*, constructs that "characterize properties of the team that are typically dynamic in nature and vary as a function of team context, inputs, processes and outcomes" (Marks et al. 2001, p. 357), including social states such as trust and task-related states such as roles, levels of commitment, or shared mental models. For example, the finding of an onion structure in most FLOSS projects suggests the general kinds of roles that emerge, though studies suggest that in most projects, there is a general lack of clearly defined roles leading to considerable role ambiguity. In other words, members, especially newcomers, are often uncertain about how best to contribute. Given the lack of explicit direction, another important emergent state is the degree to which project contributors develop a shared mental model of the structure and functionality of the project to guide their contributions.

Finally, *outputs* represent task and non-task consequences of a team functioning (Martins et al. 2004). Many possible measures of success have been proposed. For example, Crowston et al. (2006) identified seven measures of FLOSS project success: system and information quality, user satisfaction, use, individual and organizational impacts, project output, process, and outcomes for project members. They noted that impacts are often hard to discern and to tie to specifics of the project, leading to reliance on measures more closely tied to the work products.

In addition to the direct link from inputs to mediators to outputs, the IMOI model recognizes that there is a feedback loop from the outputs to the inputs of a project. For example, high-quality (e.g., modular) code (a project output) should make the codebase easier to maintain and so facilitate additional contributions (an input). User satisfaction with the project (an output) is important to retaining developers, while the success of a project (an output) may increase its visibility and ability to attract new developers

DOI 10.1007/978-3-319-04033-2_29-1

(an input). Contrariwise, a project that is struggling may find that difficulties in development cause developers and users to leave for more promising projects, thus further complicating development, leading to a downward spiral.

Proponents of FLOSS (in both flavors) claim a number of benefits to users and to society more broadly. The easy accessibility (e.g., zero cost) of software is a benefit to users, though it has been noted that the cost of the software itself is usually a small part of the total cost of adopting a software package. The availability of a system's source code reduces the risk of adoption for users by eliminating the concern that the company providing the software will disappear, taking the code with them and stranding users.

Because of the community's ability to fix bugs and add features, proponents also claim that OSS enables higher code quality. This effect has been summarized as Linux's law: with enough eyes, all bugs are shallow, meaning that bugs can be fixed more quickly than with limited developer base that has time to debug only a limited number of problems, which requires prioritizing bug fixes, even not fixing bugs that affect only a few people. Furthermore, having the code open enables it to be audited, potentially increasing security compared to closed software that is not openly reviewed. However, software being open is no guarantee that it will in fact be audited, as illustrated by the discovery of a bug in OpenSSL, a package that had only a few active developers.

Finally, the ability to examine actual working code can be useful for people learning to program, which can provide a benefit to society. FLOSS is an increasingly important venue for students learning about software development, as it provides a unique environment in which learners can be quickly exposed to real-world innovation, while being empowered and encouraged to participate. For example, Google Summer of Code program (http://code.google.com/soc/) offers student developers stipends to write code for FLOSS projects. The projects also provide mentorship to socialize potential new contributors to the project.

FLOSS in Scientific Research

We turn now to the second topic, namely, the use of FLOSS approaches to develop scientific software and the possible implications of such software for the convergence of scientific fields. Bezroukov (1999) pointed out that FLOSS development resembles academic research in its reliance on intrinsic motivations for contribution, the importance of recognition of contributions and similar funding models. Lots of scientific software is released as open source, consistent with suggestion that programmers have motivations other than payment for their coding. Releasing developed software as open source may also be an expectation of funding agencies. Releasing scientific software in particular as FLOSS is also argued for on the grounds that the tools used to develop scientific results need to be openly available to allow the research process and results to be audited.

Many FLOSS systems are general computing tools, useful across many domains, science included. At the bottom of the software stack are general purpose software such as operating systems (e.g., Linux), networking and programming language compilers or interpreters (e.g., for C++, Perl, Python, Ruby). Also useful for scientific computation is system software for parallel computing (e.g., Hadoop or MPI) and function libraries, both general purpose (e.g., Boost) and for scientific computations more specifically (e.g., SciPy).

Particularly useful for scientific computing are systems for storing data (e.g., databases such as MySQL, PostgreSQL, MongoDB, or Cassandra); systems for performing data analysis, such as the R statistics environment or octave mathematical system; systems for data mining and machine learning (e.g., weka) and for graphics (e.g., gnuplot). Qualitative data can also be processed, e.g., with content

DOI 10.1007/978-3-319-04033-2_29-1

analysis packages (e.g., tamsys) or natural language processing software (e.g., GATE). Finally, workflow packages can automate particular sequences of computation in order to improve the repeatability of computations. Some of these tools have an organization or an open source community behind them, e.g., the R Foundation that supports the development of the R statistical package.

In addition, there are many more specialized software packages for analyses in particular domains (e.g., astronomy, biology, chemistry), either standalone or building on one of the above platforms, e.g., statistical analysis in R. For example, the bioconductor project (http://www.bioconductor.org/) provides "tools for the analysis and comprehension of high-throughput genomic data," built on the R system. The Image Reduction and Analysis Facility (IRAF) is a general purpose software system for the reduction and analysis of astronomical data. Packages exist also for the social sciences, e.g., for carrying out social network analysis (e.g., Gephi).

The open availability of tools may promote cross-disciplinary use. Many of the tools are useful across sciences and are a way in which innovations in one domain can be made available to others. For example, statistical techniques implemented in the R package may originate in one discipline but find uses in others. For example, bioinformatics tools developed for the analysis of DNA sequences are being used to analyze process sequences in management enabling management researchers to plot the genealogy of different variations of process sequences.

The research on FLOSS development highlights a number of issues for the development of open scientific software. A key question is the incentive to make something open source. Many scientific software authors are motivated by the need for academic credit rather than pay. This incentive does motivate making the system available, but not necessarily to integrate it with other software packages (Howison and Herbsleb 2013). Furthermore, making a package usable by others requires additional work beyond what is needed to just use the code for personal use. As well, this credit system breaks down if those who use the code do not cite it properly, but such lack of citation is not uncommon as the norms for citing software are still developing.

Supporting users is a significant problem in all of FLOSS, as limited developer time means developers may not be able to provide help to users. Projects often rely on community to answer questions or develop various levels of documentation. A few products may be popular enough to attract commercial companies who can sell services, but many will still be reliant on volunteer support. A further problem is attracting enough developers to keep the software maintained. It is not uncommon for a tool to be developed with grant funding. But when grant runs out, it is not clear who will continue development.

Conceptual Convergence and OSS

FLOSS development has been quite visibly successful and so has inspired or further invigorated many other open movements. A key feature of FLOSS is that the outcomes are open to further use without restrictions. Accordingly, there are many suggestions for making other kinds of products open.

One such movement is the *open hardware* movement. As hardware has a cost, open hardware usually refers instead to making hardware designs openly available, with similar arguments about benefits of access and ability to customize. With developments in distributed manufacturing such as 3-D printing, the gap between designs and objects may be closing, so open hardware may be increasingly interesting and important.

Open access publishing refers to publishing of academic articles and other materials in a way that makes them available to readers without financial or legal barriers to use. Examples include a variety of Open Access journals as well as a variety of institutional or topical paper repositories (e.g., arXiv in physics and related fields). Open access is sometimes mandated by institutional or funding agency

DOI 10.1007/978-3-319-04033-2_29-1

policies, e.g., the papers published with support from the US NIH should be made available in PubMed, the NIH's repository. Because publishing does have costs (beyond reviewing and editing that are usually donated by community), it is not uncommon for an open access journal to charge authors for publishing.

Open access may benefit society by speeding use of research results and so the progress of science. It also allows novel approaches to using articles, e.g., machine processing large collections to automatically extract facts and relationships. As with free versus open source software, there is a split in the OA community. A part of the OA movement argues that results of scientific research should be available as a matter of right, sometimes more specifically because taxpayers have already paid for the research by supporting researchers. Others suggest that open access is simply a better approach to publishing because it speeds up the uptake of research results and so the progress of science and are therefore more pragmatic about how it is accomplished (e.g., via author deposit of preprints).

There are ongoing attempts to make other kinds of material open. For example, MIT's Open Courseware initiative provides access to open collections of teaching materials, which can be found in other repositories as well.

Open data means making data collected as part of scientific research or from government administration freely available to support future research, again with arguments about speeding future research, reducing its cost or improving the auditability of research. As in other cases, there is an argument that data created with public funding should be available to the public. There are now many data repositories to support data sharing (e.g., Dataverse, Dryad). However, data are much more diverse than articles and can be quite voluminous, making them problematic to store, index, search, or retrieve. There are also concerns that data by itself may be difficult to understand, with possibilities of misinterpretation or even misuse. Finally, if the data concern human subjects, there are issues of privacy that need to be considered.

Finally, the *open science* movement makes the argument that other products of the scientific process should be openly available. The issues here are difficult, as it is more problematic to share intermediate results that have not been vetted and on which the original authors may still be working.

While the previous set of movements concern the outputs of research, other movements have developed open processes for involving contributors. For example, a handful of publication outlets have applied the open development method used for software for papers. In such a system, authors post papers and the review comments and responses are open.

Citizen science – research projects engaging the public as contributors to science in fields like astronomy, ecology, and even genomics – has recently received increased attention, even though the concept is at least a century old. Broadly defined, citizen science is a form of research collaboration that involves volunteers in producing authentic scientific research. For example, citizen science research might engage volunteers as "sensors" to collect scientific data, as in the eBird project that collects bird sittings from amateur bird watchers and makes them available for research, e.g., to identify the effects of climate or habitat change on bird populations. Other projects involve volunteer "processors" to manipulate scientific data or to solve data analysis problems. For example, the GalaxyZoo project has volunteers classify the morphology of galaxies from space telescope pictures and uses the classified data to test hypotheses about galaxy evolution. Volunteers can potentially be involved in any of the processes of a scientific project. For example, the Polymath Project allows interested individuals (in practice, those with a solid background in math) to take part in mathematics research, by contributing to the proof or discussion of some open conjecture.

Involving volunteers in a project enables a new approach to research, cumulating the contributions of many individuals (Bonney et al. 2014) and taking advantage of uniquely human competencies. To produce sound science, citizen science projects must verify the quality of the research activities and data (Wiggins et al. 2011), which distinguishes them from many other forms of collective content production. Some argue that the public should be involved as well in setting goals for research. However,

DOI 10.1007/978-3-319-04033-2_29-1

consideration of motivational implications raises questions about why others would want to be involved if they do not already share those goals.

In addition to its intrinsic merits, FLOSS development has attracted great interest because it provides an accessible example of other phenomena of growing interest. For example, many researchers have turned to community-based FLOSS projects as examples of virtual work, as they are dynamic, self-organizing distributed teams comprising professionals, users, and others working together in a loosely coupled fashion (von Hippel 2001; von Hippel and von Krogh 2003). These features make FLOSS teams extreme examples of self-organizing distributed teams. While the features of community-FLOSS teams place them toward the end of the continuum of virtual work arrangements (Watson-Manheim et al. 2002), the emphasis on distributed work makes them useful as a research setting for isolating the implications of this organizational innovation.

As well, they are not inconsistent with the conditions faced by many organizations when recruiting and motivating professionals or developing distributed teams. As Peter Drucker put it, "increasingly employees are going to be volunteers, because a knowledge worker has mobility and can go pretty much every place, and knows it... Businesses will have to learn to treat knowledge workers as volunteers" (Collins and Drucker 1999). As a result, research on FLOSS development offers lessons for many organizations. Crowston (2011) notes a number of possible lessons for organizations from FLOSS development, e.g., value of making work visible as a basis for coordination and the possibility of alternative approaches to leadership.

Finally, many projects combine both open products and an open process, what are sometimes called *open contribution* projects. A particularly prominent example is Wikipedia. There are similar open communities in which users provide open multimedia contents, e.g., YouTube, MySpace, del.icio.us, Diggit, Twitter, and Facebook. As a related development, given the importance of giving credit as a way to motivate contributions, the field of altmetrics is developing to provide ways to count contributions in different modes.

Conclusion

To summarize, FLOSS technically means simply software released with source code. However, often the process of software development is also open. FLOSS software is widely used, including in science, and FLOSS has parallels of open products and open processes in numerous other domains, particularly in science.

References

Alho K, Sulonen R (1998) Supporting virtual software projects on the Web. In: Workshop on Coordinating Distributed Software Development Projects, 7th international workshop on enabling technologies: infrastructure for collaborative enterprises (WETICE), Palo Alto

Bezroukov N (1999) Open source software development as a special type of academic research (critique of vulgar raymondism). First Monday 4

Bonney R, Shirk J, Phillips T, Wiggins A, Ballard H, Miller-Rushing A, Parrish J (2014) Next Steps for Citizen Science. Science 343(6178):1436–1437

Collins J, Drucker P (1999) A conversation between Jim Collins and Peter Drucker. Drucker Found News 7:4–5

DOI 10.1007/978-3-319-04033-2_29-1

Crowston K (2011) Lessons from volunteering and free/libre open source software development for the future of work. In: Chiasson M, Henfridsson O, Karsten H, DeGross J (eds) Proceedings of the IFIP working group 82 working conference on researching the future in information systems. Springer, Berlin/Heidelberg/Turku, pp. 215–229

Crowston K, Howison J, Annabi H (2006) Information systems success in free and open source software development: theory and measures. Softw Process Improv Pract 11:123–148

Crowston K, Wei K, Howison J, Wiggins A (2012) Free/libre open source software: what we know and what we do not know. ACM Comput Surv 7

Fitzgerald B (2006) The transformation of open source software. MIS Quarterly 30(3):587–598

Free Software Foundation (2013) What is free software? The free software definition. http://www.gnu.org/philosophy/free-sw.html. Accessed 14 Feb 2015

Giuri P, Rullani F, Torrisi S (2008) Explaining leadership in virtual teams: the case of open source software. Inf Econ Policy 20:305–315. doi:10.1016/j.infoecopol.2008.06.002

Henkel J (2006) Selective revealing in open innovation processes: the case of embedded Linux. Res Policy 35:953–969

Howison J, Herbsleb JD (2013) Incentives and integration in scientific software production. In: Proceedings of the ACM conference on computer-supported cooperative work (CSCW), San Antonio, pp. 459–470

Kelty CM (2008) Two bits: the cultural significance of free software. PhD dissertation, Duke University

Lakhani KR, Wolf RG (2005) Why hackers do what they do: understanding motivation and effort in free/open source software projects. In: Feller J, Fitzgerald B, Hissam S, Lakhani KR (eds) Perspectives on free and open source software. MIT Press, Cambridge, MA

Lee GK, Cole RE (2003) From a firm-based to a community-based model of knowledge creation: the case of Linux kernel development. Organ Sci 14:633–649

Marks MA, Mathieu JE, Zaccaro SJ (2001) A temporally based framework and taxonomy of team processes. Acad Manage Rev 26:356–376

Martins LL, Gilson LL, Maynard MT (2004) Virtual teams: what do we know and where do we go from here? J Manag 30:805–835

O'Leary MB, Cummings JN (2007) The spatial, temporal and configurational characteristics of geographic dispersion in teams. MIS Q 31:433–452

O'Mahony S (2003) Guarding the commons: how community managed software projects protect their work. Res Policy 32:1179–1198. doi:10.1016/S0048-7333(03)00048-9

Open Source Initiative (n.d.) The open source definition. http://www.opensource.org/docs/osd. Accessed 14 Feb 2015

Raymond ES (1998) Homesteading the noosphere. First Monday 3

von Hippel EA (2001) Innovation by user communities: learning from open-source software. Sloan Manage Rev 42:82–86

von Hippel EA, von Krogh G (2003) Open source software and the "private-collective" innovation model: issues for organization science. Organ Sci 14:209–213

Watson-Manheim MB, Chudoba KM, Crowston K (2002) Discontinuities and continuities: a new way to understand virtual work. Inf Technol People 15:191–209

Wayner P (2000) Free for all. HarperCollins, New York

Wiggins A, Newman G, Stevenson RD, Crowston K (2011) Mechanisms for data quality and validation in citizen science. presentation at the computing for citizen science workshop at the IEEE eScience Conference, Stockholm, Sweden, 8 December