THE USA JOURNALS
THE AMERICAN JOURNAL OF SOCIAL SCIENCE AND EDUCATION INNOVATIONS (ISSN- 2689-100X)
VOLUME 06 ISSUE07

PUBLISHED DATE: - 28-07-2024
DOI: - https://doi.org/10.37547/tajssei/Volume06Issue07-15 **PAGE NO.: - 137-150**

RESEARCH ARTICLE **Open Access**

THE DYNAMICS OF OPEN SOURCE SOFTWARE DEVELOPMENT AND CODER COMMUNITIES

Vinas Khalid Kadhim
University of Karbala, College of Administration and Economics, Iraq

Abstract
The advent of open source software (OSS) and its emphasis on transparency, cooperation, and innovation has shaken up the software business. This article delves into the mechanics of open-source software development, specifically looking at how coder communities interact to shape software projects. Core developers, contributors, and users are all part of the socio-technical systems we dissect in these communities. We examine the impact of governance structures, collaboration tools, and community standards on the success and sustainability of OSS initiatives through case studies. We also look into the difficulties that OSS communities encounter, including keeping code quality, handling conflicts, and ensuring everyone can participate. Our research shows that good communication, strong leadership, and a common goal are crucial. When building solid and sustainable OSS ecosystems. This research offers insights into practical strategies for cultivating active programmer communities and maintaining the expansion of open-source projects.

Keywords Open source software (OSS), communication, Core developers, contributors.

1. INTRODUCTION

Open source software development is the most visible form of collective action on the internet. Its development codebases, and tools come with many different lexemes, but all share the same principles and values. One of the main aspects of the Open Source Definition (OSD) is about the process or flexibility. Users should be free to adapt and distribute copies to whoever is in need. The reason the developer has a lot of flexibility is because the freedom is kept. The user can download the software for free. This aspect of open source software development is often overlooked.

The word 'software' is a term of art because there is no universal definition in all aspects of the problem that the concept alert configuration is in the way the word is used. Nevertheless, the definition that we have developed into this book is quite far from the real thing, it is easy to enlarge or shrink and that is with new addition or trying what must be encompassed. However, the definition we offer should be sufficiently accurate to convey the outline of a relatively new phenomenon known as Open Source Software, or Open Source, which is responsible for many collaborative activities in software development. In the sense of open source, open-source software can be defined as software whose source codes are freely available, can be modified, and can be reproduced by anyone.

1.1. Definition and Principles of Open Source Software

Open source software is a special instance of OSSD. Although the terms are often used interchangeably, they are not quite identical. Harris and Bwalya describe open source software as "software systems that are freely distributed to anyone who wants to use them". According to

Hammelman, "open source software is software that is, at the least, freely available to any individual who obtains a copy of the software". Okerson and O'Donnell regard open source software as software that "is often free, and many open source software products, including the operating system Linux and the computer language Perl, are provided without charge". The scientific community often chooses free software because any participant can install it on his or her own computer. Software distributions of any kind cannot be used unless installed by the end user.

The Debian Free Software Guidelines (DFSG) are considered to be the definitive principles and definition for open source software. The enforced original description of free software was explained within the General Public License, whose 1st version appeared in 1984. The basic premise of the DFSG was that all software should be distributable when its source is also distributed. The implications of this are that anyone can provide the software to anyone and can expect that anyone will be able to use the software.

2. Historical Evolution of Open Source Movement

The historical evolution of the open source movement is usually dated back to the practical philosophy of the hacker during the late 1950s at MIT. The free sharing and constant improvement of source code became a strong part of the U.S. technical culture when the Homebrew Computer Club was founded in 1974. During the 1970s and 1980s, many companies and organizations provided strong economic and legal incentives for the non-free appropriation of information. Nevertheless, commercial use and communication through networks of merged free and non-free systems and programs became an attractive alternative for groups of programmers who shared a computer at a non-commercial level.

Several pioneering open source projects were started in the 1970s and 1980s by groups of researchers who were building a new kind of toy they called a 'personal computer'. In the 1970s, when proprietary operating systems were developed, programmers involved in the hardware design of computers shared the source code for the operating systems (OS) on their machines with each other. This way, they could all combine the best available operating system features in their OS versions with minimal programming efforts. They could also identify and develop changes in the source code that would improve the operation of the OS in the hardware environment they shared. During the mid-1980s, an open source UNIX descendant was developed with the idea that systems programmers and computer users could receive the combined efforts of the contributors based on the personal interest and ability of people who worked on and with the system. It was then, as some of you probably remember, that BSD and Linux-based systems were used primarily by research-based organizations outside the world of UNIX proprietary commercial computing.

2.1. Early Pioneers and Projects

This chapter is organized to discuss, in subsequent subsections, the history of open source development up to the present day, the characteristics of the social organizations that build open source software, and conclude with a discussion of an issue of terminology relating to the enhancement of open source software.

Many people contributed to the movement that is called "open source." Many were associated with early electronic networks, which is something that people outside of computer science disciplines had only begun to use in the late 1980s. Others ran decisions of computer science academic conferences. These so-called pioneer conferences were accompanied by electronic versions of conference papers and had influential mailing lists, which discussed the conferences and a number of

related issues. These conferences included "The First Workshop on Electronic Texts" held at Princeton University in May 1989, "Release 1.0" held in August 1990, "The Second Workshop on Electronic Texts" held at Princeton in September 1990, "The Third Workshop on Electronic Texts" held in May 1992, and the "First Ethicomp Conference" held in March 1996. Terry Bynum has referred to the people who addressed "core ethical issues" within this network as "coders and philosophers." Terry Winograd refers to "the AI community" and "others serious about exploring this new form of interaction." Bynum and Winograd have noted the emergence of a "virtual networked community." Generally, the people who discussed together about computer-communication systems, networked computer networks, software tools, and related subjects have a good claim to be considered digital media pioneers.

3. Key Concepts in Open Source Development

The open development and diffusion of computer code have become increasingly influential in many industrial sectors. In scholarly and popular literature, terminology and concepts derived from free and open source software, such as free, reveal generalizations and the tacit assumption that all stages of open source activities are equally participatory. Open source literature is also lacking a good inventory of key concepts and tools useful for understanding and differentiating between open source activities and the sectors and communities that support them. While attention to broader social and organizational contexts is increasing in open source studies, resources for understanding the computing activity and the software being developed are lacking. This text is an outline of a lecture course by Laura Forlano and Patrick Haas in which they present a set of key concepts for the study of open source software development (OSSD).

For open source free software to circulate, and thus to function, it must be properly licensed. In this case, the group is often legally called a 'canonical community.' A canonical community sets the legal framework that allows licensed software that travels in certain channels within the community to maintain a critical degree of openness. Should any member of the community tough (either accidentally or maliciously) get proprietary software and fail to properly open their modified source, the software might then cease to be free/open for some reason and the process might wither and die. There are often several possible cans, but in any given community, one is usually the official one and is the one in which you will see references in the copyright headers or licenses in the original documentation and source code.

3.1. Licensing and Legal Frameworks

It is essential that those actively participating, uninterested observers, potential users, and indeed all market participants understand first the legal framework that underpins open source software. This means appreciating clearly the legal rights that are granted to others in relation to the software, the duties and obligations of those who create such code, and the legal risks for users and dependent software projects.

Secondly, it is useful to understand the social reasons why open designers select the specific types of licenses that they may use. And finally, understanding what obligations apply to partners and others, including traditionally unincorporated coder groups within that legal framework.

Most software produced, purchased, or licensed today is 'proprietary'; that is, its source code is protected by copyright and/or trade secret law. Exclusive rights attached to copyright in software means that it is illegal to create modified, 'derived works of the primary code, or openly redistribute the software without first seeking and obtaining a

In response to the increasing levels of investment in software projects, individuals and firms began to reassess the advantages offered by making their code open. One solution has been the concept of open source software, and the development in abbreviated open software licenses or OSLs that grant rights of use to third parties free of charge while retaining protections on the code.

4. Roles and Contributions in Open Source Projects

When being a member of an open source project, an individual plays different roles, depending on his or her level of involvement. Acton (2004) classifies the different groups of active members (based on the amount of time invested in the project and the formal authority) of an open source project into three categories: developers, maintainers, and project leaders. With more than 90% of active participation, developers make up the major part of open source projects. Development corresponds to contributions in the form of testing, bug reports, code, and enhancements. While the passive part of the community is not visible at all, the developer active part is even further divided into the sub-roles of contributors and maintainers.

Acton (2004) also discusses the two-fold goal of every observer or participant in otherwise, a member of an open source project: the goal of pursuing organizational output by the self (in the form of, no matter which activity) and the goal of obtaining social rewards from the members of that organization, the fellow participants. Both goals are pursued by an individual in order to enhance his or her social standing in the larger field of open source developer community. Conventional organizations are typically characterized as consisting of members who form teams due to limited resources and goals that require collective action in order to be achieved. However, there are no tit-for-tat transactions associated with contributions to open source projects. "Sync releases", "set and maintain coding standards", and "improve and maintain links to the project from other websites." "Pvp" feels the need to "ensure that all switches and sockets are in slots '1' or slots '2'.". Each developer waits for contributions and interacts with a variety of other developers in a range of roles. The primary role of the developer, of course, is "general development". Other roles include "make and report bugs and other issues", "provide help and support to other users of the project", "perform tests", "submit patches", and "improve, maintain, and create new documentation". This example shows a project characterized by the fact that any developer can significantly enhance the project.

4.1. Developers, Maintainers, and Contributors

Besides being the generic term for individuals writing computer programs, in the context of a specific open source project, "developers" are the individuals who are actively designing, implementing, and testing new features. Moving from the simple definition of a developer to that of the set of all developers working on an open source project does not allow us to define the development phase of open source projects directly. This is due to the lack of a straightforward answer to the question of who these developers are and what their programming tasks are. In recent years, some scholars have pointed out that a relevant part of open source developers' activity pertains to testing and maintaining. Maintainers, frequently also called coordinators, select and integrate programmers' proposals. The ordinary activity of these individuals is not limited to the selection of patches and the evaluation of quality with the aim of registering a contribution in the software, but also extends to refactoring, developing libraries and APIs, the writing of documentation, designing patches, etc. Theoretical

analyses within the software engineering literature describe coordinators as those who allocate tasks to developers, verify the work done by the developers, and distribute the output.

Moreover, there are "submitters" who come to the project site and post bug reports or feature requests. This categorization scheme thus analyzes activity in terms of the outcome. At the top of the hierarchy are maintainers who maintain the coordinator wisdom; they select and integrate submitted patches, write and/or maintain documentation, add new features, etc. Below them are developers who submit patches and work on the project. Finally, we have users who are the submitters of bug reports and feature requests. This categorization makes sense in an environment where everybody has the right to submit patches. In this environment, being a hardware developer simply refers to submitting patches and then convincing the maintainers of the value of the patches.

5. Communication and Collaboration in Open Source

One of the "software construction" tasks is to draw the "big picture" that emerges from those mostly local "artificial symbiosis" collaborations through which open source software (OSS) gets constructed. An important aspect of this is the communication and collaboration structure and dynamics, or in the words of the field of coordination theory, the (dyadic and triadic) communication and coordination graphs that are typically only partially (and never completely) observed.

Recent work has looked at "communication dynamics", "issue dynamics", as well as (in a development of that) "coordination dynamics" in the development of various OSS projects. The use of version control systems (VCS) is one largely opaque form of communication - important because of the increasing use of distributed VCS (DVCS) for high-scale OSS development and the new insights such large-scale data are starting to provide. Many software developers engage with DVCS (such as Git, Mercurial) during their involvement with free and open source projects. This section elaborates on statistics gathered from five different DVCS-based repositories from source forge. All five projects are at different stages of their lifetime.

For each additional month of the life of open source coding projects, the rate of issues (change requests) being raised by the user community increases by 20% of the number of issues raised in the preceding month, but the closure rate of issues to date with the code base increases over time by only 13.5%. Hence users are exceeding capacity to iron out faults and defects in OSS coding projects. Code speed (rate of revision to date) continues to increase with project age at a constant rate of 4.2% for every additional month of the coding project.

5.1. Version Control Systems and Issue Tracking

Open source software projects generate and employ a large amount of developer code and metadata such as user and bug documentation, web pages, and other auxiliary documentation. Ensuring that all developers can access and use this information is just as important as contributing to projects at the code level. Typically, XML and database technologies (SQL) are employed to maintain this metadata. Automation of tasks such as metadata harvesting and cross-referencing are enabled using scripting languages such as Python, Perl, and Ruby. Yet, a key aspect of any collaborative project involves effective communication and collaboration that leads to profound progress and effective solutions. Version control systems provide a powerful way for developers to coordinate their work, solve problems, and distribute their code.

Many open source projects use CVS because of its

broad user base and extensive documentation. CVS (Concurrent Versions System) is a central repository version control system - the definitive version of the software is maintained in a central database while complete working copies are maintained on the developers' workstations where the code is modified. Tools are provided for comparing and merging changes, managing compatibility, renaming files, merging the collective changes of multiple developers, and reversing changes. Bugs and enhancement requests can be submitted to the Bug Parade; for hands-on help for patch sets, become a member of the Patch Review Group. To manage issues between the developer, quality assurance member, and project manager, some open source projects utilize issue or bug tracking systems such as Bugzilla, Jitterbug, and GNATS. These tools provide project and scheduling information for managers, developers, and even users. A problem report can be assigned to a developer or a specific release. Instructions for reporting a bug can be found in the Feature Request HOW-TO.

6. Community Dynamics in Open Source Projects

This section explores in detail how communities are structured in open source projects. Having understood that the dynamics of the governance of open source projects are best understood by looking closely at either project management or at developer communities, we will focus here mainly on the visible, tangible community layer. Understanding this part of the process dynamics is crucial to understanding conflict and decision-making patterns in open source communities. All important decisions in an open source project are made by a community. Community interests may overlap or conflict. Contributors generating part of a project frequently have a stronger interest in a healthy community than do one-shot submitters of bug reports.

3.1 Community Structure: "Core Teams", "Kernels", and "Networks" There is a long tradition, in sociology and political science, of studying the structure of reference, opinion or advice networks. Mapping the formal or informal networks sheds new light on the boundaries and centers of the community. Decision networks can also be used as a proxy for the existence standard of "core" developers. While association-role division based, for example, on the number of lines of code contributed, usable patches submitted or enough patches submitted are insufficient for identifying the decision-making positions in an OSS project, the respondents in a survey might be wholly unaware of the formal status of developers as "core" contributors. The appropriate specifications for "core" development differ significantly from project to project. The intersection of the members in the specified role networks results in so-called communities' core.

6.1. Governance Models and Decision-Making Processes

The functioning of the described communities might be based on different governance models. Some communities establish a foundation or association that provides them with sufficiently exclusive resources, e.g., for domain or internet address disputes. More recent studies on GPL software protect resources from relocation to proprietary ends. In other cases, informal tribal models of leadership and a meritocratic organization are used. These models put a relatively strong emphasis on the individual developer's skills, efforts, and standing in his or her community. Finally, the three communities use AoA decision-making models to formally channel collective decision-making processes. In the AEGIS collective made in the Free Software Foundation, the FSF article 2 "empowers" the membership of the FSF to make decisions regarding the direction of the Association. This indicates that some AoA

formed communities may have very "inclusive" decision-making processes.

In a study on software patches, O'Mahony identifies "code" as the most important resource. This and other resources relevant for collectively making decisions are shown in table 5.1. The decision-making process in open source software projects is resolved in several ways, consulting a variety of resources. Some of these resources allow all participants of a project to participate in decision-making processes, or are considered "objective" and binding. For example, Linus Timeness was used, as it formally states the release manager's responsibilities and decision-making powers. As Lerner and Tirole put it, the decision whether or not to propose a committer with write-access to the heart of a source tree is a collective decision that is resolved by a simple majority.

7. Economic Models and Sustainability in Open Source

Economic models for sustainability: One of the most pressing questions in any community centered around the future of the collective/open project (like open source) relates to economic sustainability. Open source projects are driven by donations and grants to the project as a whole or facilitated by fiscal sponsors. There are also a limited number of revenue streams possible. The first and most obvious is via the platform through advertisements or via a platform's cryptocurrency (for example, via Steem). For open source projects, access to code is often free, though support may be paid for. One study of the Apache web server showed that, because OSS undercut commercial dominance of the sector, commercial players exited the market, and it ended up creating positive externalities for the company. The need for a 'help' staff was underscored when Apache got used by large corporations.

The notion of economic transactions between support and development communities was the subject of one study examining how automatic tools can be useful in locating potentially commercial positive externalities of an application software. There are a number of different funding mechanisms possible: direct donations, fiscal sponsorship, bake sales, and grants. Each of these can be differentiated on two dimensions. The first is by who is involved. This includes who moves the money, who seeks it, who oversees it, and who keeps track of the money coming and going. The second dimension has to do with contracts. Most funding mechanisms come with a grant proposal process, a contract between the person or organization moving the money and the person or organization seeking it, and some sort of funding agreement when more than one person or organization is involved in getting the funds.

7.1. Funding and Revenue Streams

Despite the role of code as a free general-purpose script, open source developers often hope to be compensated for their labor. provide empirical evidence that this holds true not just for for-profit, but also for non-profit communities. Moreover, given that coding is a generally marketable activity, it appears plausible that a substantial part of open source developers expects to receive remuneration for their services.

7 Funding and Revenue Streams Open source as an economic activity The fact that many, possibly the majority of open source developers, seek compensation or benefits from their contributions indicates a particular form of commercial regulative. It further indicates an orientation towards operational, rather than financial sustainability, given that both commonly involve the translation of resources from the outside world into group benefits. Hence, in the remainder of this paper, the terms funding and revenue streams are used with the meaning provided in this section.

In turn, financial sustainability as understood by Lerner and Tirole "goes beyond balancing the

books: it requires both having enough money to fulfill the organization's mission and developing the strategies and resources needed to sustain income over time." (2002, p. 1495). Loosely following , a distinction must therefore be made between two kinds of financial inflow: Firstly, that continuously accumulated through the organizational setup and at least coordinatively maintained, such as membership fees in a club. Here, the "value proposition", to use the Business Model Canvas term, is a stable product. Secondly, and in a perhaps more natural open source context, there could be flows from value propositions based on the continual labor inputs of core group members, the receipt of which allows these to perform product maintenance. Valuation, here, involves the continuous reassessment of value propositions and the maintenance of core group members in the organization. Given this, a sustainable organization consists of a replicable construction of revenue-generating value propositions and a proliferating coordination of incumbents. Thus, even when no profits are generated nor desired, substantial exchanges between the business firm and its environment are still decisive of its success or failure.

8. Case Studies of Successful Open Source Projects

The Linux Kernel - Started in 1991 by Linus Torvalds, Linux is now one of the most successful open source projects. It is licensed under the GPL and is used by most computer companies. Although the development of the Linux Kernel is mainly in the hands of the core team led by Linus, thousands of people all over the world contribute to the main Linux kernel and to various add-on programs, utilities, and modules. A well-administered FTP server with the sources is run by the Open Systems Lab at Scandinavia University so that vendors and individuals may obtain Linux. More information is available on the web through various forums and FAQs.

Apache Web Server - A web server is a mechanism for viewing multimedia documents. Apache is a web server that originated at the NCSA National Center for Supercomputing Applications at the University of Illinois. Apache is an open-source project and part of the "official" client bandwidth of the National Science Foundation. Apache's market share has garnered considerable interest in the software technology and business press. Apache still runs 54% of the websites, performs 20 million requests per day, and is widely distributed with a contribution from over 1000 independent developers.

Examples of Open Source Commercial Projects - Some examples of Open Source Projects which are being supported in a commercial fashion are: 1) Ghostscript - A project started and run by Aladdin. It is now supported by the cooperation of several commercial developers such as Accel Graphics, Unix System Laboratories, Informix, and others. 2) TeX project at Stanford, now moved to the European side (yes, Knuth created TeX). 3) τ - An Adaptive Revaluation Tool - Now a commercial project by Alvey. (τ was developed under the GPSSS initiative). 4) X Windows Commands - X window manager and clients - Cooperatively by the vendors. 5) Joss Morgan's Distributed System Software - From Xerox in the 80s and now a commercial product by Data General.

Open Source Commercial Projects are doing very well. It should be noted that both Sendmail and Perl are in the Top Ten of the Top GB/Net Software tools.

8.1. Linux Kernel and Apache Web Server

The Linux Kernel and Apache Web Server

Our objective in this section is to start specifying some caveats and issues that arise in large or long-running open source projects by providing case studies of the two most successful open-source

projects to date: the Linux Kernel and the Apache Web Server. While many other projects are also very substantial, these two projects are substantial enough that one can do longitudinal studies of them. They are also fairly unusual in terms of OSS, in that they have substantial corporate sponsorship, but are nonetheless very open in terms of their development process.

2.1. The Linux Kernel

Although Open Source Software is usually thought of as being costless, producing the greatest open source project in history - the Linux Kernel - required substantial resources. The development of Linux took almost 400 person-years and $31 billion up until 2005. Approximately 97% of this development was performed by paid developers.

The development dynamics of open source systems can be surprisingly different from those in the commercial world. Raymond captured some of the thinking underlying this difference in part of a valued public speech - The Cathedral and the Bazaar - that he turned into an essay. This section of the IEEE Computer Magazine article presents a high-level summary of his insights on evolving a successful open-source project, which appeared in the book that was published of The Cathedral and the Bazaar. It's useful to be aware of his background and personal idiosyncrasies. His perspective favors a loose, non-progressive view of the world that doesn't particularly admire engineering, social sciences, or science more generally.

2.2. The Apache Web Server

The Apache server is now the most widely used web server in the world. The software has become a nested combination of products that support both Unix and Windows NT systems. Someone else performed a substantial amount of the programming work noted above. The NCSA team before the Apache group produced the beta version, and much of the development of the products on top of Apache was performed by others. Moreover, there were, at the time we were doing our study, several commercial-grade web servers available, ranging in price from $70 to $1250, and several free alternatives to Apache/Linux, including variations on Windows NT systems as well as development server systems from Microsoft.

As of early 2001, roughly a year into their work between one and two thirds of the roughly 60 official developers were working out of channels that didn't exist prior to Apache, and in the meantime, they had over 600 non-core developers, some of whom released code back into the system. One reliable estimate is that as of 2005, "these volunteers contribute at least as much code under less than 10% of the time (number of clock hours) than the paid workers contribute. Volunteered code therefore accounts for less than 10% of all the developers in Apache but nearly 30% of the total software." Code from those non-core developers was executed on more than half of all the servers in the Internet and about two thirds of the Web servers. This conclusion was reached using collection methods that might produce overestimates since some of the supposedly Apache server-side code might be buried in combinations of servers. Even so, Apache's market performance through 2000 tends to support our possible high estimate of user contributions: Apache went from zero in early 2001 to owning one half of the worldwide server market by 2004. Since then, Apache usage has leveled, making it a very viable choice for server solutions.

9. Challenges and Future Directions in Open Source Development

Open source development: New horizons; new risks?

While open source development continues to evolve and support exciting new ways to conduct

science, manage communities, and generate knowledge, it is not without challenges. In this paper, we discuss a number of challenges for open source, drawing on reflections of several researchers who have worked extensively in open source, including some who shared their insights with more junior collaborators as part of the Human and Social Dynamics of Open Source Software (HSD) and the Climate Model Intercomparison Project (CMIP).

1. Security-minded development and responsibility One recent and growing challenge for open source is the ongoing threat of security flaws and bugs with large, organizational impacts. Research has begun to explore what trust means for open source communities, and how to measure trust in these communities, raising the central importance of trust to distributed communities of volunteer workers. But while the mode of production of open source software is ideally suited to quickly identify and address security flaws, the pervasiveness of open source in the stacks of some of the world's most important caching and application architecture tools invites increased attention from malicious actors. Furthermore, because of ubiquitous web technologies, many modern open source projects are now effectively part of the Internet, vastly expanding the potential attack surfaces. In this light, we queried participants in the Climate Model Intercomparison Project (CMIP) about their perspectives on what issues confront open source development, based on their experiences maintaining and using one of the most important pieces of comprehensive scientific infrastructure these days: CMIP6. Some responses to our queries tackled factors of software quality generally implicated in large software projects.

9.1. Security and Trust Issues

With the widespread use of open source code, new forms of trust and trust interaction emerge. What is particularly interesting is the interaction between the user and non-expert developer, as well as the development crew who need to trust each other in a simple and cost-effective engineering model ("just run the code locally").

Open source development is vibrant with lively debates, shared knowledge, and collective problem solving. There are regular real-world examples of lifelong collaborative coding communities, which are similar to Amish and Kulu v.1 in terms of openness and contextualization of code. These communities draw in a wide variety of users – 89% of whom work in areas of IT, which are widely considered to create moments of trust such as transactions. Users become part-time co-developers, either by reporting errors or making contributions. By doing so, they are placed within the circulation of conversation and coded value, acquiring a personalized sense of communal surety familiar to any credit union member, Amish baker, secure transaction between bank customer and banker, or social security bounty. In this model of collaborative networks, the user-developer with a trust attitude allows for the exchange of code as a time- and space-transformable non-exclusive non-transferable good. Those with a limited trust attitude can judge it for empirical authenticity because they believe "it 'fits' with the other knowledge they have at the time, and would heavily depend on the reputation, culture, and understanding of the person who did the work".

10. The Role of Coder Communities in Open Source

A successful open source project is one that has a large, flexible, and hard-working community. That project may begin with only a single programmer or small group of developers, but to sustain it for any length of time, they usually need a larger team. Building and continuing to build an open source community is part of the development and maintenance process. The community, and the

non-coding user/advocate community, helps the project grow and mature both in size and in influence. The community around an open source project can have a great deal of influence on its "success." For coder communities specifically, rising membership and increased interest are often seen as evidence of a project's growing success, but increased membership does not automatically translate into a higher level of involvement from that community. Not only does a thriving community often make the work of development easier, the community itself can be the main foundation for the long-term success of a project.

Coder communities and a successful project form themselves in part around the development process. Some open source projects, like Linux and Apache, have become destination spots not because they do anything particularly new, but because of how they do development. Hosting an open source project is about providing users with the tools and resources they need to participate in that project. As with the development of the camping trip, users should not have to "work" very hard to participate. If they have to download five version control systems and eight bug reporting systems before they can even get to writing one line of code, chances are they never will. Providing simple ways to get involved and contribute to a project is one way to foster community involvement. Once involved, members of a project may perceive their situation as participating in what is called a "hobby community." These people are giving their time for fun, with full knowledge of what they are doing. This is different from a "work community" and a "community of practice." In these situations, most people are giving their time not only to further a goal or ideal but also for some type of compensation.

10.1. Building and Nurturing a Community

What is considered a bit of luck often is the consequence of hard work. Especially in earlier years, open source software development was often dominated by a small group of committed developers. Over time, they have not only addressed important technical issues but have also made the social processes within the community transparent. This article takes a closer look at community strategies and the mechanisms that have enabled the development of a fruitful coder community in the various open source projects. However, the social processes driving open source development are not only driven by hard work and consistency. Designers of successful projects have also learned how to foster and nourish a thriving project community.

Many open source software projects have lower barriers to entry, ranging from simple interfaces to comprehensive documentation, making it easy for dedicated developers to join an open source project. Moreover, the collective review approach already attracts developers to the project during project development. Successful project founders treat newcomers in an open manner, and other developers also help newcomers to come to terms with the new project. Knowledge is shared within the community, and entry thresholds are communicated in a transparent manner. Complete newcomers, who still have to familiarize themselves with the overall architecture and design, will first be assigned to small, isolated tasks to avoid causing a lot of follow-up work or even errors. Once these newcomers have adapted to the project code and architecture and successfully submitted their first patch, they feel a great commitment to others in the project community. It is also common for a mentorship to be established that guides the developer over the long term, helping him or her enter the project and further progress.

11. CONCLUSION

The open source software projects we surveyed

show both the lasting continuities and the emergent changes we have attempted to capture in this paper. Even as the open source universe continues to grow, further underscoring its strength and viability as an approach to software development, significant quantitative and qualitative changes in the characteristics of the open source world are beginning to take shape. The eventual impact of these emergent post-boom trends on the open source world meets our iron law; at best, we can discern somewhat tenuous hypotheses about emergent long-term changes. As van Wendyl and Hallchrist de-Meuran (2001) colorfully put it:

It is worth examining the ebbs and flows of open source development—the changes and continuities in this survey and the assumptions we have used to shape our comparative analysis. Some assumptions, like the unchanging nature of managerial need or the irrelevance of nationalism in an increasingly globalized world, have stood the test of time. The failure of these and a host of other assumptions, however, counsels humility as we look to the future of open source.

What, then, do our findings from the 2001 comparative survey suggest about the prospects for open source in the twenty-first century? What directions is the movement currently following, and where is it likely to go in the future? In this conclusion of our comparative survey, we pull together several strands of our quantitative and qualitative analysis. In particular, we consider the emerging trends, the directions that contemporary open source appears to be following, and the major open research questions in open source that remain unanswered.

11.1. Emerging Trends in Open Source Development

In this section, we explore some of the most important recent trends in open source development and describe how they relate to the case studies presented elsewhere in the book. We have divided these trends into three categories: institutional environments, organization of OSS development, and the innovation outcomes of OSS.

2.1 Institutional environments

One of the most obvious changes in OSS development in recent years has been the increasing range of institutional settings and structures within which the work is carried out, and which pay for workers to work on projects. Although our histories of OSS projects run from the late 1980s, owe a great deal to lone individuals coding away in their spare time, over the last decade, we have seen the emergence of a number of new forms of project organization and support. These range from stand-alone projects run by a single company for its advantage, through community-based or predominantly community-based projects, to obligatory enterprises based on large amounts of software, some of it OSS or technically open source, packaged up and sold to organizations who wish to use it.

In our cases, we find a variety of different forms of project principle, from foundation-run projects such as the Apache Software Foundation itself to companies such as MySQL or Vignette, which use and distribute LAMP or Lucene. XP and Eclipse are both software projects with something of an obligatory element, in the sense that the communities and companies around the Eclipse simultaneous release are able to achieve significant economies of scale and the widespread confusion attendant upon the release-time delivery of many interrelated software components as a single upgrade pack, packages that would be impossible but for the distributed global community-building activities that result from everyone hitting the same deadline. Some of these products, however, are also being written for fun or simply to see the technology pushed forward, as with GM Rush if not GM. In many cases,

developers have offshoots of the main project within the institutional space in terms of the projects or packages OSS developers work with for clients.

12. REFERENCES

1. Baker, Alexander, Michel Avital, and Eri Kesalainen (2015), Digital Infrastructure: A Content Analysis of the Issues of Open Source, Cloud, and 5G ISP, 21st Americas Conference on Information Systems
2. Bonaccorsi, Andrea, Omar Compagnone, and Luca Spataro (2011), Participation in Open Community: A Model of Motivation for New Participants as Moderators of Conflicts and Exclusion in the OSS Development Communities, Premium Technologies and Infonomics
3. Deessen, Andreas (2013), Designing a Supporting Open Source Platform for Shared Content Creation, Hamburg University of Applied Sciences
4. Deussen, Andreas, and Mathew Henry (2015), The Many Benefits of Being an Open Source Evangelist, University of Leipzig
5. Ghosh, Rishab Aiyer (2003), Understanding Open Source Software Hackers through the Internet, International Journal of Technology, Policy, and Management
6. Iacovou, Charalambos, Cyrill Liu, and Yong Liu (2014), Time to Degree Events in Open Source Software Development: A Comparative Study of Premium/Open Access Journals, Information Systems Management
7. Jungherr, Marcel, Harald Schoen, Oliver Posegga, and Pascal Jürgens (2017), Digital Trace Data in the Study of Public Opinion: An Indicator of Attention Toward Politics Rather Than Political Support, Social Science Computer Review
8. Markus, M. Lynne, Jacqueline Sherris, and Jeff Turner (2005), Social Structures of Email Networks, Journal of Management Information Systems
9. Nan, Ning, She-I Chang, Tianli Wang, Michael Chi Yuan Lee, Jinwei Liu, and Yanna Wu (2017), Analyzing the Relationship Between Productivity and Code Quality in Open Source Software Development, Journal of Management Information Systems
10. Neus, Andreas, Frank Schäfer, Günter Jacobs, and Jan Prasch (2014), Do Open Source Platforms Substitute Firms? An Empirical Analysis of the Relationship between Open Source Platforms and Firms, Journal of Management Information Systems
11. Pan, Shimei, Xuequn Wang, and Jianhua Hou (2015), Capturing Genetic Data on Open Source Software Development, Communications of the Association for Information Systems
12. Picot, Arnold, and Matthias Achter (2001), The Startup-ecosystem as Clans in a Virtual High-Tech Community: Spect, Future Imperatives for Management Information Systems
13. Schackmann, Frank, Andre Carrel, Yusuke Sugano, and Ivo Krka (2014), Guidelines for Reporting Open Source Software Engineering Processes (and Annika Repschläge from 2006 to 2011), Information and Software Technology
14. Weiss, Maik, and Astried Moaba (2015), Studying Privacy Policy Statements for Mobile Apps at Scale Using Heterogeneous Data, Journal of the Association for Information Systems
15. Yeh, Ching-Yi, and Che-Wei Wang (2013), Analyzing Co-experience in a 3.0 Approach with Open Source Ones: A Study on Wikipedia, Drugstore.com Too Antab, International

Journal of Technology and Human Interaction.