The Many Meanings of Open Source

Cristina Gacek and Budi Arief, *University of Newcastle upon Tyne*

The term "open source" frequently refers to a software development process that relies on the contributions of geographically dispersed developers via the Internet. One basic requirement of an open source project is the availability of its source code, without which the software's development or evolution is difficult, if not impossible. But apart from these characteristics, some confusion exists on what an open source project actually is.

Projects that claim to be open source have many varying characteristics. To demonstrate this, we investigated 80 open source projects: several large, well-known projects, such as Linux, Apache, and Mozilla, and some smaller ones. We reviewed published materials about open source, notably *The Cathedral and the Bazaar*,[1] *Rebel Code*,[2] and *Open Sources*,[3] as well as other works.[4–10] We also used several online resources dedicated to open source projects, such as SourceForge (http://sourceforge.net) and Geocrawler (www.geocrawler.com). In addition, we interviewed 12 individuals who either work on open source projects in their free time or are involved with open source as part of their job in large corporations.

From there, we further dissected open source by determining the characteristics that open source projects usually have or should have. We determined a set of characteristics that are almost always present and others that vary among open source projects. By exposing these characteristics, we've created a taxonomy against which you can compare any project's characteristics. Additionally, these characteristics demonstrate that just stating that a project is open source doesn't necessarily precisely define the approach used to support the project.

A multidisciplinary viewpoint can help determine those characteristics that are common to all open source projects and those that vary among projects. Consequently, they provide a starting point for understanding what "open source" means.

A multidisciplinary approach

Software development is a complex process that draws upon knowledge and expertise

0740-7459/04/$20.00 © 2004 IEEE

from many scientific disciplines. So, to understand it better, we need to take into account its interdisciplinary nature. Open source software development is no exception to this rule. In determining the relevant open source characteristics, we considered these disciplines:

- *Computing science* covers the technical aspects of open source projects.
- *Management and organization* deals with managerial issues and how they relate to the projects.
- *The social sciences* address areas related to the communities involved in the projects and their behavior.
- *Psychology* accounts for the characteristics of the individuals involved in the projects.
- *Economics* looks into the economic models that underlie the projects or corporations with respect to their involvement in the projects.
- *Law* focuses on legal issues.

One thing that stands out from our study is that "open source" is not a precise term. Some characteristics exist in all open source projects, but there are even more characteristics that might vary considerably from project to project.

Common characteristics

Although thousands of projects are classified as open source, they all share only two main characteristics: they adhere to the Open Source Definition, and developers are always users.

Adherence to the OSD

The Open Source Initiative composed the OSD (www.opensource.org/docs/definition.html) as a guideline to determine whether a particular software distribution is open source. The OSD outlines three main criteria:

- The ability to distribute the software freely
- The source code's availability
- The right to create derived works through modification

Six more criteria deal with licensing issues and spell out the requisite "no discrimination" stance. (That is, anyone may use this software, for any field of endeavor.) All nine criteria listed under the OSI definition of open source—most prominently source code availability—are the basic requirements for projects to qualify as open source.

For more on the OSD and other attempts to investigate open source characteristics, see the sidebar.

Developers are users

The people who contribute code to an open source project are always users of the code produced. This means that open source developers are a subset of the open source user

Related Work

The Open Source Initiative provides an *Open Source Definition* (www.opensource.org/docs/definition.html), which asserts nine criteria for open source software. (See the section "Adherence to the OSD" in the main article for more on the criteria). The OSD addresses legal issues extensively and encompasses some economic aspects. However, it hardly touches on computing science, and it completely ignores psychology, social sciences, and management issues. Furthermore, there's no guarantee that a given project, by simply adhering to the OSI definition of open source, benefits from the positive effects usually associated with open source (for example, being reviewed by many people).

The open source software characteristics that Huaiqing Wang and Chen Wang proposed address some technical aspects and, in less depth, legal and managerial aspects.[1]

Andrea Capiluppi, Patricia Lago, and Maurizio Morisio analyzed a sample of 400 open source projects.[2] They addressed only technical issues, such as the project's age, application domain, size, modularity and documentation levels, popularity, and vitality.

Sandeep Krishnamurthy[3] gathered statistics from the top 100 projects on SourceForge. He focused on the size of the project's community, the project's age, and the associated number of downloads and official messages, looking for correlations between them.

Joseph Feller and Brian Fitzgerald noted that although OSI certification is useful, it couldn't be considered an essential part of the definition of open source.[4] They applied an analytical framework to understand open source by the type of products, process used, stakeholders, environment, and motivations. Although their research is very interesting, their set of characteristics isn't as rich as the one we propose in this article. Also, they didn't explore the commonalities versus variabilities among open source projects.

References

1. H. Wang and C. Wang, "Open Source Software Adoption: A Status Report," *IEEE Software*, vol. 18, no. 2, Mar./Apr. 2001, pp. 90–95.
2. A. Capiluppi, P. Lago, and M. Morisio, "Characteristics of Open Source Projects," *Proc. 7th European Conf. Software Maintenance and Reengineering* (CSMR 03), IEEE CS Press, 2003, pp. 317–330.
3. S. Krishnamurthy, "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday*, vol. 7, no. 6, June 2002, www.firstmonday.dk/issues/issue7_6/krishnamurthy.
4. J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*, Addison-Wesley, 2002.

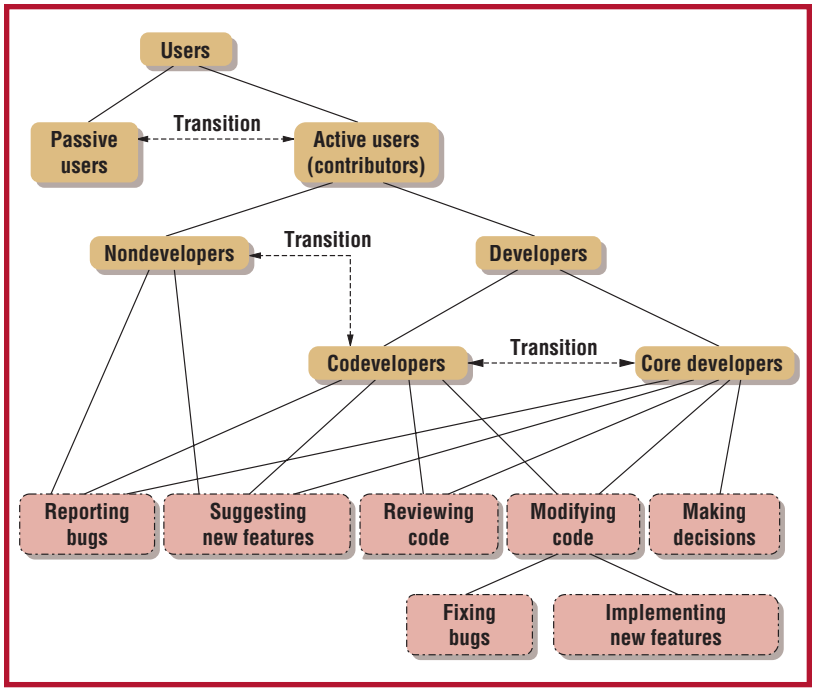

Figure 1. A classification of open source users and developers.

community. That is, all open source developers are users, but not all users are developers (see Figure 1).

Variable characteristics

We found these variable characteristics: *project starting points*, *motivation*, *community*, *software development support*, *licensing*, and *size*.

Project starting points

Open source projects might start from scratch or from existing commercial or research closed-source software systems. All the projects we studied converted closed-source software to open source software at once. Nevertheless, you could envision some closed-source software making a gradual transition to open source, one part (for example, a subsystem) at a time.

Motivation

The biggest question surrounding the open source phenomenon is, why do people do it? Why would people contribute code for free? The answer isn't as straightforward as you might think. Contributors, whether individuals or corporations, contribute to satisfy a perceived need. Individuals usually contribute for personal satisfaction; some have strong philosophical beliefs about the resulting software's openness, while others don't care as much about such issues. Corporations usually get involved to gain market share or undermine their competitors, or they simply use open source software so that they won't have to build an equivalent product from scratch.

Peer recognition also motivates contributions. When the community involved recognizes the contribution of individuals or corporations as appropriate and of good quality, their status increases within the given project. Consequently, others will consider their opinions more carefully with respect to project-related decisions, and their reputation might improve even outside the project boundaries.

Depending on the domain that an open source project addresses, different business models might motivate the involvement of corporations, researchers, individual developers, and end users. So far, we've identified three business models:

- Software for own use
- Packaging and selling of the software
- A platform or foundation for commercial or research software development

Community

Active open source projects usually have a well-defined community with common interests that's involved either in continuously evolving its related products or in using its results. However, many open source projects have no clear community structure or involve just one person (as is the case in many SourceForge projects).

This characteristic involves two issues: *balance of centralization and decentralization* and *meritocratic culture*.

Balance of centralization and decentralization. Some communities have a strict hierarchy differentiating various levels of developers (see Figure 1); others have a much looser structure.

The strict hierarchies bring with them a more centralized power structure. For example, the core developers have more power than ordinary codevelopers in making executive decisions. Some open source projects (for example, Apache) even have more than two levels of developers.

Looser organizational structures have all their developers on the same level. This implies decentralized decision making, which sometimes is based on full consensus.

Meritocratic culture. The basic model underly-

ing open source projects is that knowledge shown through contributions increases the contributor's perceived merit, which in turn leads to power. Exactly how this transition takes place varies from project to project in terms of timing and the obstacles to overcome, and depends on the project's organizational structure.

For example, Figure 1 shows the possible transition from passive to active users when they start contributing to the project. If they can then show their ability (or if they can gain respect from the community), they might be invited into the developer group. There, they would have greater rights over the code (for example, to incorporate their own modifications into the code base). In some projects, codevelopers can be promoted to the core developer group. Transitions can also go the other way. For example, a core developer might wish to resign and become a codeveloper instead (or even leave the project completely), owing to other commitments or a personality clash.

Software development support

Open source software development requires support similar to that for traditional software development. It also requires support for specific needs generated by the (potentially) numerous highly distributed developers.

Modularity. Modular design's benefits are well established in all engineering disciplines; it supports increased understanding during design and concurrent allocation of work during implementation. Because open source development is globally distributed, well-defined interfaces and modularized source code are a prerequisite for effective remote collaboration.[11]

Visibility of software architecture. A computing system's software architecture depicts its structure and comprises its software components, the components' externally visible properties, and their relationships.[12] An open source software system's architecture might be available or not. An unintentionally unavailable software architecture suggests that the structure exists in some people's minds only.

Documentation and testing. Documentation and testing are important aspects of software development. Good documentation allows people to use—and more specifically in open source projects, to understand and modify—the software. Thorough testing gives users (and developers) confidence that the software will function as expected.

These two areas are often overlooked or vary widely during open source development. Open source contributors tend to be more interested in coding than documenting or testing. This is probably because open source tries to replace the formal testing process with the "many eyeballs" approach to eliminating bugs. Also, developers often feel that adding comments in the source code is sufficient documentation. There have been some attempts to address the lack of documentation—for example, the Linux Documentation Project (www.tldp.org) and Mozilla Developer Documentation Web page (www.mozilla.org/docs). However, this is still a rarity for smaller open source projects. In addition, we've yet to find some sort of testing strategies for open source projects. They might exist, but if so, they're implicit and they aren't visible outside the project's developer community.

Accepting submissions. An open source project evolves by receiving submissions from various sources to address the project's various aspects. The most common submissions are bug reports and source code; others include documentation and test cases. Furthermore, open source projects often post the areas for which they want to receive submissions. Consequently, they might receive multiple concurrent submissions addressing the exact same area. So, open source projects have in place processes for accepting various types of submissions, while clearly specifying how to handle multiple concurrent submissions.

The process of accepting submissions comprises three facets. The first is *choosing the work area*. As we just mentioned, open source projects often request contributions to specific areas. Some projects will process both solicited and spontaneous contributions, whereas other open source projects might tend to ignore spontaneous contributions.

The second facet is *decision making*, which relies on four dimensions:

- The quality goals
- The acceptance criteria
- The decision group's cognitive abilities
- The project's social structure

Open source software development requires support similar to that for traditional software development.

Table 1

Varying characteristics of open source licenses

Licenses	Does it impact derived works?	Can it be closed?
GPL (GNU General Public License)	Yes	No
LGPL (GNU Lesser GPL)	No	No
BSD (Berkeley Software Distribution) License	No	Yes
QPL (Q Public License)	No	No
IBM Public License	No	Yes
MPL (Mozilla Public License)	No	Yes

Quality goals vary widely from one project to another; this can happen even in the same application area (for example, one project focusing on performance and another on portability). Acceptance criteria also vary. Example criteria include the best solution out of the first *n* submissions, an aggregation of multiple submissions (even by requesting that someone changes his or her solution to add an aspect seen elsewhere), some memory of previous submissions by the same person, or the first submission received. Additionally, the ability to recognize better solutions depends highly on the decision group's cognitive abilities. This implies that decision making for accepting submissions varies among projects and potentially within projects, unless the same people help make all the decisions.

The social structure might be a defined hierarchy where different groups of people evaluate different submissions (for example, by focus area), or where some people exercise greater power, or both. Or, the structure might be a monolithic group consisting of all developers. The social structure directly affects decision making. If the group is monolithic, it might use consensus or majority vote to accept submissions. If a different social structure exists, consensus or majority voting might also apply, but at times some members' votes will count more than others'.

The third facet is *disseminating the submission information*. A project might passively disseminate this information through newsgroups or comments in the code itself. It might actively disseminate the information through email and mailing lists. Or, it might devote Web space to the information.

Tool and operational support. To facilitate concurrent software development and fast, controlled evolution, most open source projects implement some form of configuration management. They do this by using CVS (the Concurrent Versions System), other tools, or even an ad hoc solution using Web-based support.

Communities related to specific projects communicate almost exclusively by electronic means, which they also use to organize their work. The most common means are dedicated mailing lists, newsgroups, and Web sites. The exact structure and use of these means vary among projects.

Licensing

Several types of licenses conform to the OSD. Some ensure that if any of the software code is used in other software development, all the software will come under the terms of that original license. Another aspect of these licenses concerns whether they restrict distribution of any of the original source code to binary form in future derived software products. Table 1 illustrates how six of the more popular licenses implement these two features.

Size

Size is not a distinctive measure in open source projects. The sizes of both the community and the code base vary widely from project to project.

Using the open source characteristics

We've used these characteristics to describe nine existing open source projects: Linux, Topologilinux, Frozen Bubble, Tux Typing, Mozilla, Bugzilla, Apache, Project @ssistant, and JUnit (all available via SourceForge). We're investigating several other projects with the intent of populating an extensive database of projects enabling us to investigate correlations among these characteristics. We'll also use this database to determine possible correlations between project characteristics and the developed software's reliability.

Clearly, open source projects' characteristics can vary greatly, but that's true of all software projects. Table 2 shows how our taxonomy of characteristics applies to both open source and traditional projects.

Observations

Interesting observations abound regarding open source projects, but few studies of empirical data have confirmed them. Here are some things we observed in our research.

One expert we interviewed claimed that considerably greater activity occurs in the

Table 2

Open source and traditional software project characteristics

Characteristic	Open source		Traditional	
	Common	Variable	Common	Variable
Adherence to the Open Source Definition	✔		N/A	N/A
Developers are users	✔			✔
Starting points		✔		✔
Motivation		✔	✔	
Community				
Balance of centralization and decentralization		✔		✔
Meritocratic culture		✔		✔
Software development support				
Modularity		✔		✔
Visibility of software architecture		✔		✔
Documentation and testing		✔		✔
Accepting submissions				
Choosing the work area		✔		✔
Decision making		✔		✔
Disseminating the submission information		✔	N/A	N/A
Tool and operational support		✔		✔
Licensing		✔		✔
Size		✔		✔

Northern Hemisphere during winter. (He said that during the summer months, people tend to spend more time traveling on vacation and enjoying the outdoors.) We noticed that as the number of developers grows in a project, a more structured hierarchy is implemented.

Because developers are users, a lightweight requirements-engineering approach is possible. Additionally, many people believe that open source projects react more promptly to problems (for example, resolving a security flaw). However, for centralized organizations that use open source software, ensuring that the latest patch is installed everywhere is close to impossible.

Having no contractual deadlines can be a problem for organizations relying on open source projects as a platform for software development. The danger of forking exists in open source projects, but no one knows how often this really happens or what characteristics usually enable this situation. (Forking is the evolution of two or more separate strands of work from the original code base.)

Few projects involve more than 20 developers, and many hundreds of projects involve just one developer. This implies that you can't rely on the number of coders or reviewers to maintain code quality. The developers' maturity and profile also vary greatly.

Several empirical studies have started to address some of these observations. Andrea Capiluppi, Patricia Lago, and Maurizio Morisio found that the number of developers per project is typically low (one or two) and that a project's evolution is usually slow.[9] Efforts tend to be spent on "big" projects such as Linux and Apache, which probably aren't "average" open source projects. Sandeep Krishnamurthy also reports that individuals, rather than communities, develop most open source projects.[8] These findings clearly indicate the need for further empirical studies.

Future work

Many issues remain about understanding and exploiting the open source approach. Questions that we'd like to investigate further include these:

- Does open source foster more dependable software development? If so, how?
- As we mentioned before, one claimed advantage of software products developed as open source is that many reviewers examine the code. So, can more reviews replace formal analysis as a guarantee of dependability?
- What are the mutual influences between software architecture and group structure in open source software development?
- Does the software architecture decay faster in open source software?
- Who takes responsibility if something goes wrong when someone uses an open source product?

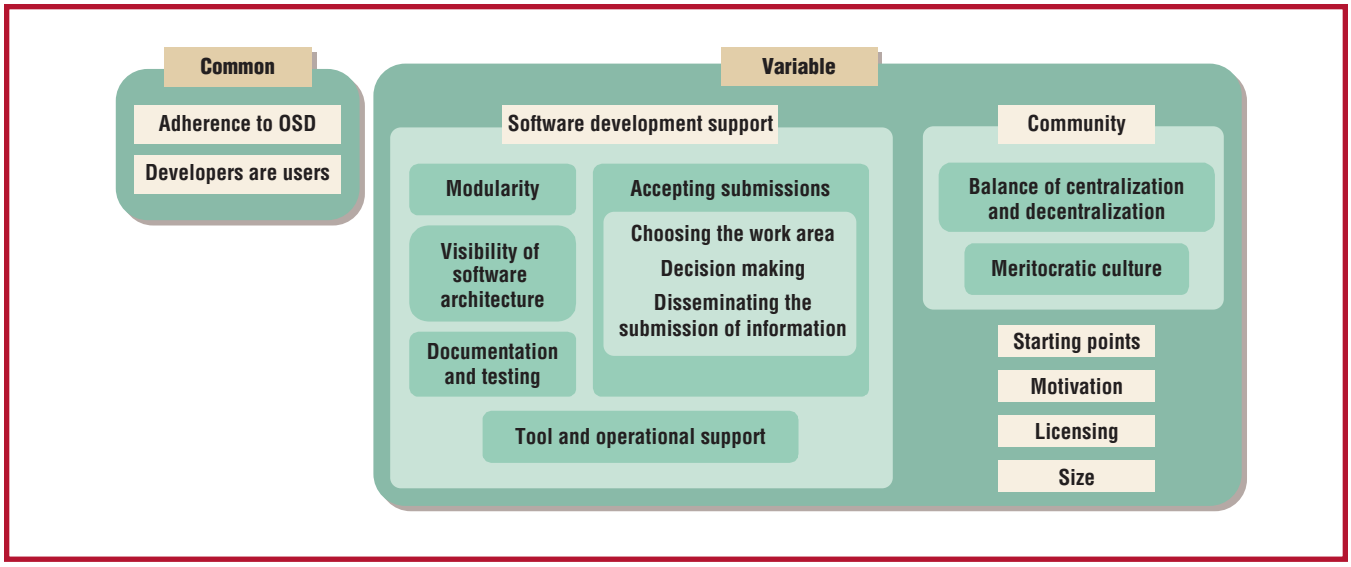

Figure 2. A taxonomy of open source characteristics.

We also plan to look into statistical information regarding open source software. In addition, we'll run controlled experiments to isolate and validate our assumptions and those from the community at large.

Figure 2 summarizes our set of open source characteristics. We understand that no one will ever be able to generate an absolute taxonomy. Because of variations from one open source project to another, additional variable characteristics might exist. However, our list provides a starting point for understanding open source and its many meanings and can be useful for both analyzing and setting up projects.

Acknowledgments

The UK Engineering and Physical Sciences Research Council's Dependable Interdisciplinary Research Collaboration project (www.dirc.org.uk) partly funded this article. We thank Tony Lawrie for his involvement in the research leading to this article, the volunteers who shared their experiences with us, our colleagues from the DIRC project who participated in various fruitful discussions, and our students for their various contributions. These include Denis Besnard, Diana Bosio, Julian Coleman, Mike Ellison, David Greathead, Cliff Jones, Brian Randell, Lorenzo Strigini, and Stuart Wheater. We also thank the anonymous reviewers for their useful feedback.

References

1. E.S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, 1999.
2. G. Moody, *Rebel Code: Linux and the Open Source Revolution*, Perseus Publishing, 2001.
3. C. Dibona, M. Stone, and S. Ockman, *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, 1999.
4. J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*, Addison-Wesley, 2002.
5. J. Feller and B. Fitzgerald, "A Framework Analysis of the Open Source Software Development Paradigm," *Proc. 21st Int'l Conf. Information Systems*, ACM Press, 2000, pp. 58–69.
6. A. Mockus, R.T. Fielding, and J. Herbsleb, "A Case Study of Open Source Software Development: The Apache Server," *Proc. 22nd Int'l Conf. Software Eng.* (ICSE 2000), ACM Press, 2000, pp. 263–272.
7. B.J. Dempsey et al., "A Quantitative Profile of a Community of Open Source Linux Developers," tech. report TR-1999-05, School of Information and Library Science, Univ. North Carolina at Chapel Hill, 1999.
8. S. Krishnamurthy, "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday*, vol. 7, no. 6, June 2002, www.firstmonday.dk/issues/issue7_6/krishnamurthy.
9. A. Capiluppi, P. Lago, and M. Morisio, "Characteristics of Open Source Projects," *Proc. 7th European Conf. Software Maintenance and Reengineering* (CSMR 03), IEEE CS Press, 2003, pp. 317–330.
10. H. Wang and C. Wang, "Open Source Software Adoption: A Status Report," *IEEE Software*, vol. 18, no. 2, Mar./Apr. 2001, pp. 90–95.
11. T. Bollinger et al., "Open-Source Methods: Peering through the Clutter," *IEEE Software*, vol. 16, no. 4, July/Aug. 1999, pp. 8–11.
12. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

About the Authors

Cristina Gacek is a lecturer at the University of Newcastle upon Tyne's School of Computing Science. Her research interests include software architectures and product lines to support the engineering of complex software systems, with a focus on improving their dependability. She received her PhD in computer science from the University of Southern California. She's a member of the IEEE and the ACM. Contact her at the School of Computing Science, Univ. of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, UK; cristina.gacek@ncl.ac.uk.

Budi Arief is a research associate at the School of Computing Science of the University of Newcastle upon Tyne. He's working on the UK Engineering and Physical Sciences Research Council's Dependable Interdisciplinary Research Collaboration project. His research interests include open source and free software, human aspects of computer-based systems, and computer security. He received both his bachelor's of computing science and his PhD in computing science from the University of Newcastle upon Tyne. Contact him at the School of Computing Science, Univ. of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, UK; l.b.arief@ncl.ac.uk.