Cover Feature

Open Source Software: Lessons from and for Software Engineering

Brian Fitzgerald
Lero Research Centre, University of Limerick, Ireland

Despite initial suggestions to the contrary, open source software projects exhibit many of the fundamental tenets of software engineering. Likewise, the existence of category-killer apps suggests that conventional software engineering can draw some lessons from OSS.

Open source software can elicit strongly contrasting reactions. Advocates claim that OSS is high-quality software produced on a rapid time scale and for free or at very low cost by extremely talented developers. At the same time, critics characterize OSS as variable-quality software that has little or no documentation, is unpredictable as to stability or reliability, and rests on an uncertain legal foundation—the result of a chaotic development process that is completely alien to software engineering's fundamental tenets and conventional wisdom.

Research suggests a more balanced view. On one hand, OSS is not the "silver bullet" championed by its most vocal partisans. On the other hand, it does not radically diverge from traditional software engineering practice as its severest detractors claim, and, as evidenced by some notable successes, OSS offers many tangible benefits.

OSS AS A SILVER BULLET

Twenty-five years ago, IBM software engineer Fred Brooks famously contended that "there is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity."[1] However, many claim that OSS is indeed such a silver bullet.

Defenders argue that OSS, beyond its obvious cost advantages, is of very high quality. Contributors to OSS projects are in the top 5 percent of developers worldwide in terms of ability, and are self-selected and thus highly motivated. Furthermore, the testing pool is global, and peer review is truly independent.

Another key advantage cited is the rapid development time of projects. The OSS community has taken odds with Brooks' law—namely, that "adding manpower to a late software product makes it later,"[2] a conclusion based on his experience managing development of the IBM OS/360—by endorsing Linus's law: "given enough eyeballs, every bug is shallow."[3]

There are many examples of OSS products of exceptional quality and reliability across a range of application domains—indeed, "category killers" such as the Linux kernel and Apache webserver perform so well that there is no market for an alternative.

NOT SO FAST

Critics concede that a staggering mélange of OSS products is readily available for free download, but they claim it is virtually impossible to predict the usability, stability, and reliability of these products. The uneven quality is not helped by a lack of documentation and the reliance on support and upgrades from a voluntary community who must be convinced to accept changes to suit specific circumstances. These flaws are exacerbated by a complex licensing situation in which even the lawyers cannot definitively resolve IP rights issues.[4]

Furthermore, OSS arises from a development process that seems to flout traditional best practices. For example, typically there is no real formal design process, no risk assessment or measurable goals, often no

direct monetary incentives for developers or organizations, informal coordination and control, and much redundancy as tasks are duplicated in parallel initiatives. All of this is anathema to conventional software engineering.

Other analyses of OSS say that 30 years of prior software engineering research cannot be discounted so easily. The claims in relation to the quality of OSS products and of community feedback are particularly questionable when exposed to scrutiny.

Quality

A study by Ioannis Stamelos and colleagues assessed quality issues in the SuSE Linux 6.0 release.[5] Using the Logiscope code analysis tool, they examined more than 600,000 lines of code across 100 modules and found that only 50 percent were acceptable. Of the remainder, 31 percent required comments, 9 percent required further inspection, 4 percent required further testing, and 6 percent needed to be completely rewritten. These results are quite average in the software industry: only half of all modules meet generally accepted standards.

In a similar vein, Srdjan Rusovan, Mark Lawford, and David Parnas studied the implementation of the Address Resolution Protocol in the Linux TCP/IP implementation and identified numerous software quality problems.[6]

Community feedback

The claim of high-quality feedback from the OSS community is also questionable. A study of OSS development by Niels Jørgensen revealed that while simpler code gets more feedback, it is generally not all that useful.[7] A sort of inverse Pareto principle is likely at work, in that 99 percent of OSS developers spot 80 percent of the bugs, but only about 1 percent of the developers can identify the more difficult 20 percent. Furthermore, the Jørgensen study showed that there was very little feedback on design issues, a significant deficiency.

Also, the fact that OSS is the choice of the technologically literate could be problematic. On his OS/2 Headquarters website, Tom Nadeau argued that proprietary software vendors always gear their software to "the most ignorant customers," while OSS developers cater to the "smartest customers" and can thus cut back on niceties such as a user-friendly interface.[8] This phenomenon appears to be somewhat borne out by the comments of one Linux user who, after installing the OS, posted a message referring to the "thrilling adventure" of the installation.

LESSONS FROM SOFTWARE ENGINEERING

In light of these critiques of OSS, it is worth considering the lessons and principles that OSS has drawn from software engineering. It is readily apparent that sound software engineering principles such as a modular architecture and sophisticated configuration management are very much at the heart of successful OSS projects.

Linux offers one demonstration of the importance of *modularity* in OSS. Linux benefited greatly from the elimination of defects and fleshing out of requirements in Unix.[9] Indeed, the manner in which different individuals take responsibility for various self-contained modules within Linux is acknowledged as a major factor in its successful evolution.

The Sendmail utility offers additional evidence of the role of modularity in OSS. Sendmail was first developed in the late 1970s at the University of California, Berkeley, by Eric Allman, who made the source code available to all interested parties. However, when problems in integrating these efforts emerged as the utility began to evolve through others' contributions, Allman rewrote Sendmail to follow a more modular structure. This ensured that the program would be a suitable candidate for massive parallel development, a characteristic of OSS, as developers could largely work independently on different aspects. Sendmail is now the dominant internetwork e-mail router, handling an estimated 80 percent of all Internet e-mail.[10]

The modular approach applies to project structure as well as the code base: large OSS projects tend to be aggregations of smaller projects.[11] This allows developing, fixing, and releasing components more independently.

Configuration management is likewise a vitally important factor in OSS, and several sophisticated tools exist for this purpose. In addition, the software engineering principles of *independent peer review and*

testing are highly evolved within OSS.

In short, the code in OSS products is often very structured and modular, and developers carefully vet and incorporate contributions in a disciplined fashion in accordance with good configuration management and independent peer review and testing. OSS development does not depart significantly from many sensible and proven software engineering principles, and it is simplistic to characterize OSS as a "bazaar" with an undisiciplined development process.

LESSONS FOR SOFTWARE ENGINEERING

Despite overblown hype at times, there are undoubted and notable OSS successes. OSS can contribute much to software engineering knowledge, including open innovation, global software development, inner source, and time-based release management.

Open innovation

Open innovation has become a holy grail in organizational endeavors, including software development. Recognizing that no single organization will have a monopoly on creative people, open innovation seeks to leverage ideas from a wider, ideally global, talent pool.

Certain characteristics are important stimulants to innovation, including

- *autonomy*, which forms the basis for self-organizing and increases the possibility that individuals will motivate themselves to form new knowledge;
- *creative chaos*, whereby individuals do not have to follow organizational rules but are challenged to investigate alternatives and rethink assumptions;
- *information redundancy*, whereby individuals have information that goes beyond their immediate needs for a particular task; and
- *requisite variety*, whereby individuals have the diverse skills needed to match the complexity and variability of the environment they face.[12]

All these characteristics are readily found in OSS communities. Developers tend to self-select and are largely autonomous in relation to the tasks they undertake. Given that most OSS developers work outside organizational boundaries, creative chaos can exist. The openness of the code at mature points in the development process facilitates information redundancy. And the cosmopolitan nature of OSS developer communities ensures requisite variety.

Much of OSS obeys a power law.[13] An interesting property of power-law distributions is that they do not have a peak at the average—hence they scale. This is evident in typical OSS projects. For example, while some might suggest that Firefox has too many developers,[14] several hundred thousand people use test versions of the browser, and about 20 percent take the time to contribute bug reports. This pool of users is an extremely useful resource.

OSS has also been a source of inspiration in terms of innovative business models. One model is to offer a free open source version of a proprietary product that entices customers to purchase the enterprise version with some additional functionality.[15] Also, innovations and new features emerge from the OSS community's creative mindset.

As Eric Raymond memorably observed, most OSS developers have "a personal scratch to itch."[3] It is thus no accident that many successful OSS products are general purpose. Given Jørgensen's finding that feedback on design issues in OSS development is rare,[7] it appears that OSS is best suited to horizontal domains in which there is widespread agreement on the design architecture and the general composition of the software requirements is fairly well known and unproblematic. This is probably essential with a large base of contributors from a wide variety of industrial and academic backgrounds. On the other hand, in vertical domains where requirements and design issues are a function of specific domain knowledge that can only be acquired over time—the case with many business environments—there are not likely to be as many OSS offerings.

Given OSS's potential for innovation, it is ironic that many early efforts replicated proprietary software products. However, unique features originated in these OSS clones that were typically ported back into their proprietary counterparts.

Global software development

GSD dramatically increases coordination, communication, and control challenges in software development.[16] Given the current outsourcing and globalization, GSD is an issue of increasing significance for organizations today.

OSS resolves coordination issues in GSD with simple communication tools—e-mail, newsgroups, and version control systems.[17] The "secret sauce" seems to lie in the coordination structures present in OSS. At the center is a team of experts with varied experience who tend to coordinate their work informally but are aware of one another's expertise. This relatively small core group does the vast majority of coding, but it is complemented by larger teams of bug-fixers and testers drawn from the user population. The latter boost productivity and reduce defect density but do not add interdependencies, as finding and reporting bugs does not involve code changes. Consequently, several studies have reported efforts to transfer OSS lessons to GSD within organizations.[17,18]

Inner source

The phenomenon of adopting OSS practices within a corporate setting is known as *inner source*,[19] also called corporate open source[20] and progressive open source.[21] While there is no standard set of OSS practices, some common ones include open sharing of source code, large-scale independent peer review, the community development model, and the expanded role of users.[22] Leveraging a product's users as codevelopers can improve quality and generate specialized new features that are important to a wider audience.[23] Although OSS practices are generally more applicable to large organizations due to their inherent geographic distribution, smaller organizations can also benefit from OSS development practices.[24]

Companies usually employ inner source to capitalize on the success of certain open source projects. However, there are important differences between open source and closed source development and their respective communities.[25]

In traditional software development, developers and user testers are typically in separate departments or locations. This can lead to employees being unaware of other projects and innovations, all too frequently resulting in a lack of mutual respect or voluntary interaction.

While early open source developers were users of actual products, as OSS has evolved, the situation has changed. In the absence of a traditional software development company, users need to become more intimately involved in the development process, as technical staff cannot simply send a checklist of requirements to the vendor. It is a widely held belief that deploying open source can lead to a sense of shared adventure, which is not a common scenario in the proprietary software arena. Also, it has been reported that OSS developers take greater pride in their work and feel a greater sense of responsibility to deliver high-quality code because peers they truly respect will review their efforts.[26]

Time-based release management

Release management has been the subject of little research in the software engineering field. Traditional models focused on the initial release of a software product and ignored subsequent releases,[27] but the industry now recognizes that a continuous-release strategy delivers both fixes and new functionality to users. This strategy also staves off obsolescence by maintaining the software's value.

The norm is to release a new version of software when it meets a specific set of criteria and has attained certain goals, usually features important to customers. In commercial software release management, this strategy requires delicate balancing as introducing a new release too early could erode the market share and revenue-generating potential of the existing one.[28]

To mitigate risk from some OSS practices such as the lack of deadlines, the reliance on volunteers, and ad hoc coordination and management, numerous OSS projects appear to have formalized their release management process.[29] This is an important part of quality assurance because developers stop adding new features during the preparation for a release and instead focus on identifying and removing defects. The feedback obtained after a release also provides information about which parts of the software might need more attention.

In the case of OSS, however, it is not obvious how a team of loosely connected, globally distributed volunteers can work together to release high-quality software, some of which consists of millions of line of code written by thousands of people, in a timely fashion. There is much evidence that this is a serious problem. For example, the Debian OS has increasingly experienced delays and unpredictability, with up to three years between stable releases. However, this pales compared to the compression utility gzip, with 13

years between stable releases (1993-2006).

Consequently, several OSS projects have radically changed their release management processes and moved to a time-based strategy. This approach sets a specific release date well in advance and creates a schedule so contributors can plan accordingly. Prior to the release, there is a cutoff date on which developers evaluate all features for stability and maturity and then decide whether to include them in the upcoming release or postpone them to the next one.

While the specific time-based approach differs from project to project, there is a common pattern of staged progress toward a release in which each stage is associated with increasing control over permitted changes. These control mechanisms are known as *freezes* because development is slowly halted. Freeze categories include

- *feature* freeze: no new functionality can be added—the focus is on removing defects;
- *string* freeze: no messages displayed by the program, such as error messages, can be changed—this allows translating as many messages as possible before the release; and
- *code* freeze: permission is required to make any change, even to fix bugs.[29]

In modular component releases, developers can fix and release defective modules while using a time-based strategy to combine components and test the integrated product, as is the case with Debian and GNOME (GNU Object Model Environment).[29]

The trend toward software as a service suggests that a release management strategy focused on big-bang features is not suitable, as customers prefer to obtain continuous improvements from a vendor website rather than buy a new shrink-wrapped product. A time-based release management strategy is ideal for regularly adding new functionality.

Open source software promises to be part of the software landscape for some time to come.[30] While the notion of OSS as a silver bullet might be an inaccurate stereotype, OSS projects clearly exhibit many of the fundamental tenets of software engineering. Likewise, the fact that OSS provides some category killer apps developed in a GSD context—recognized to be a complex development environment—suggests that conventional software engineering can draw lessons from OSS.

Acknowledgments

Thanks to Klaas-Jan Stol for providing useful feedback in the development of this article. Support for this work came from Science Foundation Ireland through its grant to Lero—the Irish Software Engineering Research Centre.

References

1. F.P. Brooks, "No Silver Bullet—Essence and Accident in Software Engineering," *Proc. IFIP 10th World Computing Conf.*, Elsevier Science, 1986, pp. 1069-1076.
2. F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.
3. E.S. Raymond, *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media, 1999.
4. K.-J. Stol and M.A. Babar, "Challenges in Using Open Source Software in Product Development: A Review of the Literature," *Proc. 3rd Int'l Workshop Emerging Trends in Free/Libre/Open Source Software Research and Development* (FLOSS 10), ACM Press, 2010, pp. 17-22.
5. I. Stamelos, L. Angelis, and A. Oykonomou, "Code Quality Analysis in Open Source Software Development," *Information Systems J.*, Jan. 2002, pp. 43-60.
6. S. Rusovan, M. Lawford, and D. Parnas, "Open Source Software Development: Future or Fad?," *Perspectives on Free and Open Source Software*, J. Feller, et al., eds., MIT Press, 2005, pp. 107-121.
7. N. Jørgensen, "Putting It All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project," *Information Systems J.*, Oct. 2001, pp. 321-336.
8. T. Nadeau, "Learning from Linux: OS/2 and the Halloween Memos," OS/2 Headquarters, 1999;

www.os2hq.com/archives/linmemo1.htm.

9. S. McConnell, "Open-Source Methodology: Ready for Prime Time?," *IEEE Software*, July/Aug. 1999, pp. 6-11.
10. B. Costales et al., *Sendmail*, 4th ed., O'Reilly Media, 2007.
11. K. Crowston and J. Howison, "The Social Structure of Free and Open Source Software Development," *First Monday*, Feb. 2005; http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1478/1393.
12. I. Nonaka, "A Dynamic Theory of Organizational Knowledge Creation," *Organization Science*, Feb. 1994, pp. 14-37.
13. G. Madey, V. Freeh, and R. Tynan, "The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory," *Proc. 8th Americas Conf. Information Systems* (AMCIS 02), Assoc. for Information Systems, 2002, pp. 1806-1813.
14. G. Hayes, "Firefox Has Too Many Developers," blog, 14 Dec. 2009; www.trollaxor.com/2009/12/firefox-has-too-many-developers.html.
15. P.J. Agerfalk and B. Fitzgerald, "Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy," *MIS Q.*, June 2008, pp. 385-410.
16. P.J. Agerfalk and B. Fitzgerald, "Flexible and Distributed Software Processes: Old Petunias in New Bowls?," *Comm. ACM*, Oct. 2006, pp. 26-34.
17. A. Mockus and J.D. Herbsleb, "Why Not Improve Coordination in Distributed Software Development by Stealing Good Ideas from Open Source?," *Proc. 2nd ICSE Workshop Open Source Software Eng.* (ICSE 02), ACM Press, 2002, pp. 35-37.
18. J.R. Erenkrantz and R.N. Taylor, "Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community," *Proc. 1st Workshop Open Source in an Industrial Context* (OSIC 03), ACM Press, 2003; http://flosshub.org/system/files/erenkrantz2003.pdf.
19. J. Wesselius, "The Bazaar inside the Cathedral: Business Models for Internal Markets," *IEEE Software*, May 2008, pp. 60-66.
20. V.K. Gurbani, G. Anita, and J.D. Herbsleb, "A Case Study of a Corporate Open Source Development Model," *Proc. 28th Int'l Conf. Software Eng.* (ICSE 06), ACM Press, 2006, pp. 472-481.
21. J. Dinkelacker et al., "Progressive Open Source," *Proc. 24th Int'l Conf. Software Eng.* (ICSE 02), ACM Press, 2002, pp. 177-184.
22. K.-J. Stol et al., "A Comparative Study of Challenges in Integrating Open Source Software and Inner Source Software," to appear in *Information and Software Technology*, 2011, doi:10.1016/j.infsof.2011.06.007.
23. T. O'Reilly, "Lessons from Open Source Software Development," *Comm. ACM*, Apr. 1999, pp. 33-37.
24. K. Martin and B. Hoffman, "An Open Source Approach to Developing Software in a Small Organization," *IEEE Software*, Jan. 2007, pp. 46-53.
25. W. Scacchi et al., "Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice*, Mar./Apr. 2006, pp. 95-105.
26. C. Melian, "Progressive Open Source: The Construction of a Development Project at Hewlett-Packard," PhD dissertation, Stockholm School of Economics, 2007.
27. K.D. Levin and O. Yadid, "Optimal Release Time of Improved Versions of Software Packages," *Information and Software Technology*, Jan./Feb. 1990, pp. 65-70.
28. M.S. Krishnan, "Software Release Management: A Business Perspective," *Proc. 1994 Conf. Centre for Advanced Studies on Collaborative Research* (CASCON 94), IBM Press, 1994, pp. 36-43.
29. B. Fitzgerald and M. Michlmayr, "Time-Based Release Management in Free/Open Source (FOSS) Projects," Lero Technical Reports, http://lero.ie/sites/default/files/Lero-TR-2011-04.pdf

30. B. Fitzgerald, "The Transformation of Open Source Software," *MIS Q.*, Sept. 2006, pp. 587-598.

Brian Fitzgerald *holds the Frederick A. Krehbiel Chair II in Innovation in Global Business and Technology and is vice president of research at the University of Limerick, Ireland. He is also a principal investigator at Lero—the Irish Software Engineering Research Centre and founding director of the Lero Graduate School in Software Engineering. His research focuses on software development, encompassing development methods, global software development, agile methods, and open source software. Fitzgerald received a PhD in computer science from the University of London. He is a fellow of the Irish Computer Society and the British Computer Society. Contact him at bf@ul.ie.*

Keywords: Open source software, Software engineering, Open innovation, Inner source, Global software development, Time-based release management