Open Source Systems and Engineering: Strengths, Weaknesses and Prospects

Javaid Iqbal[*]
S.M.K.Quadri[**]
Tariq Rasool[***]

Abstract

Purpose: *This paper reviews the open source software systems (OSSS) and the open source software engineering with reference to their strengths, weaknesses and prospects. Though, it is not possible to spell out the better of the two software engineering processes, the paper outlines the areas where the open source methodology holds edge over conventional closed source software engineering. Then, the weaknesses are also highlighted, which tilt the balance the other way.*

Design/Methodology/Approach: *The study is based on the works carried out earlier by the scholars regarding the potentialities and shortcomings of OSSS.*

Findings: *A mix of strengths and weaknesses make it hard to pronounce open source as the panacea. However, the open source does have very promising prospect; owing to its radical approach to the established software engineering principles, it has spectacularly managed to carve a "mainstream" role, that too in just over a few decades.*

Keywords: *Open Source Software (OSS); Open Source Development Paradigm; Software Engineering; Open Source Software Engineering.*

Introduction

Open source traces back to early 1960s, yet as a term, *"open source initiative"* was coined in 1998 **(Open Source, n.d)**. The history of open source is closely tied to that of UNIX. The rise of open source paradigm marks the end of the dominance of the proprietary-driven, close source software setup that dominated the arena over many decades. A new ideology that promises a lot in terms of economics, development environment and unrestricted user involvement, has been evolving in a big way thrusted into the big picture by loosely-centralized, cooperative, and gratis contributions from the individual developer-user to startle the purists in the field of software engineering. Eventually, open source software phenomenon has systematically metamorphosed from a *"fringe activity"* into a more mainstream and commercially viable form. The open source initiative succeeded spectacularly well.

[*] Assistant Professor. P.G Department of Computer Science, University of Kashmir- North Campus (India). email: pzjavaidz@yahoo.co.in
[**] Head. P. G. Department of Computer Sciences, University of Kashmir, Jammu and Kashmir. 190 006. email: India. quadrismk@hotmail.com
[***] Lecturer. P.G Department of Computer Science. Islamic University of Science and Technology. email l: India. tariq_babu1@rediffmail.com

Defining Open source Software

The term open source software (OSS) refers to software equipped with licenses that provide existing and future users the right to use, inspect, modify, and distribute *(modified and unmodified)* versions of the software to others. It is not only the concept of providing "free" access to the software and its source code that makes OSS the phenomenon that it is, but also the development culture **(Raymond, 1999)**. **Kogut and Metiu (2001)** also comment on open source as right offered to the users to change the source code without making any payment. **Nakakoji, Yamamoto, Nishinaka, Kishida and Ye (2002)** refer to OSS as software systems that are free to use and whose source code is fully accessible to anyone who is interested in them.

Open Source Software Engineering

The open source development (OSD) model fundamentally changes the approaches and economics of traditional software development marking a paradigm shift in software engineering. Open source is a software development methodology that makes source code available to a large internet-based community. Typically, open source software is developed by an internet-based community of programmers. Participation is voluntary and participants do not receive direct compensation for their work. In addition, the full source code is made available to the public. Developers also devolve most property rights to the public, including the right to use, to redistribute and to modify the software free of charge. This is a direct challenge to the established assumptions about software markets that threaten the position of commercial software vendors **(Hars & Ou, 2001)**. **Torvalds *et al* (2001)** acknowledges that OSS is not architected but grows with directed evolution. An open source software system must have its source code available free, for its use, custom-tailoring or its evolution in general by anyone whosoever is interested. Thus, from the point of view of a purist in traditional software engineering, open source is a break-away paradigm in terms of its defiance of conventional software engineering and non-adherence to the standardized norms and practices of the maturing software engineering process that we have been carrying along with so much of devotion all the way through our legacy systems. The open source development model breaks away from the normal in-house commercial development processes. The self-involved/self-styled open source developer-user uses the software and contributes to its development as well, giving birth to a user- centered participatory design process.

What Leads to the Success of OSS?

Many important factors have catapulted the paradigm of OSS development to the forefront in software industry, which include cost, time, manpower, resources, quality, credit acknowledgement, spirit of shared enterprise etc:

- **Cost:** OSS products are usually freely available for public download.
- **Time:** The fact that OSS is massive parallel development and debugging environment wherein the parallel but collaborative efforts of globally distributed developers allow the development of OSS products much more quickly than conventional software systems, considerably narrowing down the gestation period.
- **Manpower:** With the development environment being spread across the globe, the best-skilled professionals work under the global development environment. This means more people are involved in the process.
- **Resources:** Again, more skilled professionals offer their resources for the development of OSS products.
- **Quality:** OSS products are recognized for their high standards of reliability, efficiency, and robustness. **Raymond (2001)** suggests that the high quality of OSS can be achieved due to high degree of peer review and user involvement in bug/defect detection.
- **Credit Acknowledgement:** The fact that people across the globe work on OSS find a chance collaborating with their peers gaining immediate credit acknowledgement for their contribution.
- **Informal Development Environment:** The informal development environment, unlike the organizational settings, liberates the developers from formal ways and conduct; more students see a chance working on real-time projects at their places.
- **Spirit of Shared Enterprise:** Organizations that deploy OSS products freely offer advice to one another, sharing insights regarding the quality upliftments and lessons learnt.

Open Source Software Development Process *versus* Conventional Software Development Process

Open source development is attracting considerable attention in the current climate of outsourcing and off-shoring (globally distributed software development). Organizations are seeking to emulate open source success on traditional development projects, through initiatives variously labeled as inner source, corporate source, or community source **(Dinkelacker & Garg, 2001; Gurbani, Garvert & Herbsleb, 2005)**. The conventional software development process encompasses the four

phases comprising the software development life cycle. These phases are *planning, analysis, design, and implementation*. In open source software development process, these phases are accomplished in a way that is probably blurry in a sense as the first three phases of planning, analysis, and design are, kind of, blended and performed typically by a single developer or small core group. Given the ideal that a large number of globally distributed developers of different levels of ability and domain expertise should be able to contribute subsequently, the requirement analysis phase is largely superseded. Requirements are taken as generally understood and not in need of interaction among developers and end-users. Design decisions also tend to be made in advance before the larger pool of developers starts to contribute. Modularization of system is the basis for distributing the development process. Systems are highly modularized to allow distribution of work and thereby reduce the learning efforts to be made by new developers to participate (they can focus on particular subsystems without needing to consider the system in its totality). However, over-modularization can have reverse effects by increasing the risk of common coupling, an insidious problem in which modules unnecessarily refer to variables and structures in other modules. Thus, there has to be a balanced approach vis-à-vis modularization.

In proprietary software, software quality testing is limited within a controlled environment and specific scenarios **(Lerner & Tirole, 2002)**. However, OSS development involves much more elaborate testing as OSS solutions are tested in various environments, by various skills and experiences of different programmers, and are tested in various geographic locations around the world **(Lakhani & Hippel, 2003; Lerner & Tirole, 2002; Mockus, Fielding & Herbsleb, 2002; West, 2003).**

In the OSS development life cycle, the *implementation phase* consists of several sub- phases **(Feller & Fitzgerald, 2002)**:

- **Code:** Writing code and submitting to the OSS community for review.
- **Review:** Strength of OSS is the independent and prompt peer review.
- **Pre-commit Test:** Contributions are tested carefully before being committed.
- **Development Release:** Code contributions may be included in the development release within a short time of having been submitted—this rapid implementation being a significant motivator for developers.
- **Parallel Debugging:** The so-called *Linus' Law,* "given enough eyeballs, every bug is shallow" as the large number of potential

debuggers on different platforms and system configurations ensures bugs are found and fixed quickly.

- **Production Release:** A relatively stable, debugged production version of the system is released.

A common classification of the various stages of open source software is planning (only an idea, no code written), pre-alpha (first release, code written may not compile/run), alpha (code released works and takes shape), beta (feature-complete code released but low reliability- faults present), stable (code is usefully reliable, minor changes) and mature (final stage- no changes).

Strengths of Open Source Software

According to **Feller and Fitzgerald (2000)**, OSS is characterized by active developers' community living in a global virtual boundary. OSS has emerged to address common problems of traditional software development that includes software exceeding its budget both in terms of time, and money, plus making the production of quick, inexpensive, and high quality reliable software possible. The advantages and unique strengths of open source software systems include *release frequency, solution to software-crisis, scalability, learnability* and *customer input* and so on.

- **Release Frequency**

One of the basic tenets of open source system is *"release early, release often"* **(Raymond, 1999)**. It is this tenet which helps a significant feedback on a global level to shape up the open source product. With the exceptional globally distributed test-users, who report their fault findings back, the frequent release policy is very feasible. However, high-release frequencies are infeasible for production environments. For these types of uses, stable releases are provided, leaving the choice about tracking new releases and updates in the hands of the users.

- **Solution to Software-Crisis**

The recurring problems of exceeding of budget, failure to meet deadlines in development schedule, and general dissatisfaction when the product is eventually delivered especially in highly complex systems always demands an alternative to circumvent these problems so that the so-called *"software crisis"* is dealt with. Open source software model does promise a solution in this regard. The source of its advantage lies in concurrence of development and de-bugging **(Kogut & Metiu, 2001)**. In fact, OSS is massively parallel development and debugging.

- **Scalability**

According to *Brooks Law, "adding people to a late project makes it later".* The logic underlying this law is that as a rule, software development productivity does not scale up as the number of developers increases.

However, the law may not hold well when it comes to software debugging and quality assurance actions. Unlike, the software development productivity, quality assurance productivity does scale up as the number of developers helping to debug the software increases. Quality assurance activities scale better since they do not require as much interpersonal communication as software development activities (particularly design activities) often do. In an OSS, there is a handful of core developers (who need not centralized but could be spread across the globe) are responsible for ensuring the architectural integrity of the software system. Then, there is a multitude of user-developers who form a user community across the globe. This community conducts the testing and debugging activities on the software released periodically by the core team. There is an obvious dynamism in the roles of the developer-at core and user- in community, in the sense that their roles may change in the context of above discussion.

- **Learnability**

A very good thing about open source software development is that it is an inherent learning process for anyone involved with it. A member does contribute to the software development but at the same time learns from the community. Thus, open source is a global campaign for skill-set development. According to **Edwards (2000)**, *"open source software development is a learning process where the involved parties contribute to, and learn from the community".*

- **Customer Input**

The informal organizational structure of core and community does not introduce any delays in the reporting of bug by a user to the core, who can immediately fix it. Moreover, the use of some impressive internet-enabled configuration management tools [e.g. GNU-CVS (concurrent versioning system)] allows a quick synchronization with updates (issued by core) on part of community. This mechanism of immediate reward, by way of rapid bug-fixing, in open-source user community helps upholding the quality assurance activity. There are no restrictions on bug-fixing by them when the source code is open or they can design a test case of the same for use by core. Such a positive influence of the user community supplements debugging process in its entirety thereby leading to a visible improvement in software quality.

This discussion should not drop a notion that OSSS are a panacea. Such systems do have their weaknesses too, in fact, plenty of them.

Weaknesses of Open Source Software

Open source is by no means a panacea and does have its own weaknesses. As expected, most of the weaknesses are the result of lack of formal organization or clear responsibilities.

- **Diversity in Communication**

The globally distributed development environment brings in the developers from different cultural backgrounds and differing time-zones, having never met in person. Moreover, even if they cross these barriers, they hit a stumbling block when skillful community members find it hard to communicate in English. As a result, misunderstandings do crop up and the communicated content may be misconstrued. This may set in a feel of lack of cooperation, good manners, and useful information among the community.

- **Uncoordinated Redundant Efforts**

With little coordination among the open source team, independent parties sometimes carry out tasks in parallel without knowing about each other. This consumes additional resources, but it may prove to be a blessing in disguise as there may be several solutions to choose from. The choice amongst the alternatives makes the selection difficult.

- **Absence of Organizational Formalism**

It surfaces multi-pronged weaknesses. Absence of laid down formal rules and conventions, makes it hard for the community to work on systematic lines. This may manifest as lack of organizational commitment in terms of a time-schedule, and a diverging organizational focus. Without a time-schedule and without a concerted focus (spearheading), the distributed nature helps offset priorities, which may be either nonexistent or severely skewed towards the personal biases of influential contributors. In this un-organizational setting where no one is boss and no one is bossed, forcing the prioritization of certain policy matters is not possible.

- **Non-Orientation of New-comers**

The new-comers do not undergo skill-setting and behavior-shaping orientation training. The new-comers have to learn the *nitty-gritty* involved very subtly. The tightly-knit community can do well, sharing their cultural backgrounds, but the new-comers are a problem. In fact, every one competes for attention and talent; these barriers to entry are very damaging to a project.

- **Dependency on Key Persons**

The bulk of the work is done by a few dedicated members or a core team -- what Brooks calls a *"surgical team"* **(Jones, 2000)**. Instead, we find that many projects critically depend on a few key persons who have the level of intimate knowledge that is required to understand all parts of a large software system. It is usually the core contributors who are the key persons. However, this dependency can become a liability if these key persons are unable to continue work on the project for some reason. It may be impossible to reconstruct the implicit knowledge of these persons from their artifacts (source code, documentation, notes, and emails) alone. This often leads to project failure.

- **Leadership Traits**

Open source leaders who, lead by persuasion alone, are judged on the basis of their technical skills, vision and communication skills. **Raymond (1999)** points out that the success of the Linux project was to a large degree due to the excellent leadership skills demonstrated by its founder *Linus Torvalds*. The scarcity of good leaders is one of the growth-inhibiting factors in open source enterprises.

Prospects

The open source phenomenon raises many interesting questions. Its proponents regard it as a paradigmatic change where the economics of private goods built on the scarcity of resources are replaced by the economics of public goods where scarcity is not an issue. Critics argue that open source software will always be relegated to niche areas; that it cannot compete with their commercial opponents in terms of product stability and reliability **(Lewis, 1999)**. Moreover, they also argue that open source projects lack the capability to innovate.

The OSSS prospect sounds encouraging when the absence of direct pay (compensations) and monetary rewards as well as property right claims have never been a bottleneck for its pervasiveness. It has direct implications on social welfare. Open source may hold key to the so-called *"software-crisis"*. The flourishing of this model to the extent of a significant market-share in absence of any marketing/advertising makes the prospect even sounder. OSS having been known for operating systems and development tools have already stepped into the arena of entertainment applications' development. Actively growing interaction between academic institutions and the IT industry has contributed significantly in research and development of such systems and the progress is going great done. Open source is internet-based and hence together with ICT (Information and Communication Technology) has a lot of scope in terms of development and economics.

Conclusion

As an emerging approach the open source paradigm provides an effective way to create a globally distributed development environment wherein the community on a specific open source software project is interacting constantly and providing feedback to activities such as defect identification, bug fixing, new feature request, and support requests for the further improvement. This activity is rewarded by peer recognition of their work and immediate recognition through credit acknowledgement creating a promotional influence on effective development practices across the community.

Open source has its strengths and weaknesses. The strengths come from its innovative development in and across a global development community of user-turned-developer. The weaknesses stem from the daring defiance of established and matured conventional software engineering principles and practice. However, though a good mix of strengths and weaknesses hold the open source in balance, the prospects of this paradigm are promising. *Fostering innovation to improve productivity* seems to be mission-statement of open source.

References

Dinkelacker, J., & Garg, P. (2001). Applying Open Source Concepts to a Corporate Environment. In *Proceedings of 1st Workshop on Open Source Software Engineering*, Toronto, May 15, 2001 Retrieved from http://opensource.ucc.ie/icse2001

Edwards, K. (2000). Epistemic Communities, Situated Learning and Open Source Software Development. Department of Manufacturing Engineering and Management, Technical University of Denmark, 2000

Feller, J., and Fitzgerald, B. (2002). Understanding Open Source Software Development, Addison-Wesley; London, 2002.

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. In Proceedings of the 21st Annual International Conference on Information Systems, pp. 58–69, Brisbane, Australia, 2000.

Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2005). A Case Study of Open Source Tools and Practices in a Commercial Setting. In Proceedings of the 5th Workshop on Open Source Software Engineering, St. Louis, MO, May 17, 2005, pp. 24-29.

Open Souce. (n.d). The Open Source Definition. *Open Source Initiative.* Retrieved from http://www.opensource.org

Hars, A., & Ou, S. (2001). Working for free?-Motivations of Participating in Open Source Projects. In Proceedings of the 34th Hawaii International Conference on System science-2001

Jones, P. (2000). Brooks' Law and open source: The more the merrier? IBM Developer Works, May 2000.

Kogut, B., & Metiu, A. (2001). Open Source Software Development and Distributed Innovation. April 2001

Lakhani, K. R., & Hippel, E. Von. (2003). How open source software works: "free" user-to-user assistance. *Research Policy*. 32 (6), 923–943.

Lewis, T. (1999). The open source acid test. *Computer*, 32 (2), 125-128. doi:10.1109/2.745728

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*. 50 (2),197–234.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*. 11 (3), 309–346.

Nakakoji, K., Yamamoto Y., Nishinaka, Y., Kishida, K., & Ye. Y. (2002). Evolution Patterns of Open-Source Software Systems and Communities. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2002)* (Orlando, FL, 2002), 76-85.

Raymond, E. S. (1999). *The cathedral & the bazaar: Musings on Linux and open source by an accidental revolutionary*. Beijing: O'Reilly.

Raymond, E. S. (2001). *The cathedral and the bazaar: Musings on Linux and Open Source by an accidental revolutionary*. Beijing: O'Reilly.

Torvalds, L. et al. (2001). Software Development as directed Evolution Linux Kernel Mailing List, December 2001

West, J. (2003). How open is open enough? Melding proprietary and open source platform strategies. *Research Policy*. 32 (7), 1259–1285.