The Mysteries of Open Source Software: Black and White and Red All Over[*]?

Brian Fitzgerald
University of Limerick, Ireland
{bf@ul.ie

Pär J. Ågerfalk
University of Limerick, Ireland
par.agerfalk@ul.ie

Abstract

Open Source Software (OSS) has attracted enormous media and research attention since the term was coined in February 1998. The concept itself is founded on the paradoxical premise that software source code—the 'crown jewels' for many proprietary software companies—should be provided freely to anyone who wishes to see it. Given this fundamental initial paradox, it is perhaps hardly surprising that the OSS concept is characterised by contradictions, paradoxes and tensions throughout. In this paper we focus specifically on the following issues in relation to OSS: the cathedral v. bazaar development approach; collectivism v. individualism, the bitter strife within the OSS community itself (OSS v. OSS), and between OSS and the Free Software Foundation (OSS v. FSF); whether OSS represents a paradigm shift in the software industry; whether the software is truly open—the Berkeley Conundrum, as we have termed it here; whether OSS truly is high quality software; and whether OSS is a 'one size fits all,' representing the future model for all software development.

1. Introduction

While the concept of free software is as old as software itself (Campbell-Kelly, 2003), there has been an explosion of academic and commercial interest in the topic since the coining of the term "Open Source Software" (OSS) in 1998. The observation that the average Navajo Indian family in 1950s America consisted of a father, mother, two children and three anthropologists certainly has resonance for OSS in so far as it would have seemed extremely unlikely just a few years ago that what would appear to be primarily a nerdy software topic could elicit so much interest from such a diverse range of research disciplines. Researchers from sociology, economics, management, psychology, public policy and law, and many others, have focused on the open source topic with great gusto.

The OSS concept abounds with contradictions, paradoxes and tensions. On the one hand, there are advocates who suggest that OSS represents a paradigm shift which can solve the 'software crisis' (i.e. systems taking too long to develop, costing too much, and not working very well when eventually delivered). These advocates point to the astonishing quality and reliability of OSS software, its rapid release schedule, and the fact that it is available without charge. At the extreme, the proponents of OSS identify it as the language of the networked community, suggesting that it will be the dominant mode of work for knowledge-workers in the information society (O'Reilly, 2000). However, countering this positive view, there are those who suggest that OSS is just the latest 'silver bullet' in the software industry, that it is over-hyped, a strategy employed by the weak with marginal products to compete with the strong. At the extreme, some influential commentators, such as Bob Metcalfe, founder of 3Com, have described OSS as "utopian balderdash" (Metcalfe, 1999), and it has even been suggested that OSS is "a disaster waiting to happen" (Sessions, 1999). Taking the specific case of the Linux operating system, without doubt the most high-profile OSS example, the original creators of the Unix operating system differ in their opinions, with Dennis Ritchie describing Linux as "commendable", while Ken Thompson has declared that Linux "is quite unreliable" and "will not be very successful in the long run." (Thompson, 1999).

Against this background of extreme positions, it is hardly surprising that OSS abounds with tensions and paradoxes. In this paper, we focus on the following:

- The cathedral v. bazaar issue: using Raymond's (1999[1]) original characterisation we consider the

[*] The title of the paper is drawn from a popular children's riddle (in English-speaking regions obviously), which uses the pun on the similar pronunciation of the words 'red' and 'read' (past tense) in English, i.e., Q: *what is black and white and red (read) all over?* A: *a newspaper*. It was chosen as a title for this paper as 'black and white' represents the contradictions, paradoxes and tensions that exist in OSS, and the 'red' is a reference to its apparent communistic nature.

[1] Raymond's work represents a seminal contribution in OSS. While the reference cited is dated 1999, the publications represents an amalgam of Raymond's landmark essays which appeared a number of years earlier, and which evolved in true OSS style over time as feedback was provided. Indeed, the early versions of The Cathedral and the Bazaar paper used the term 'free

extent to which OSS development is characterised by a cathedral or a bazaar development approach.

- The collectivist v. individualist issue: is OSS a collectivist phenomenon—an "impossible public good," as Smith & Kollock (1999) have characterised it, or is it a individualistic phenomenon fuelled primarily by reputation-seeking egotists?
- OSS v. OSS v. FSF: the considerable tension and in-fighting in the OSS community, with the resignation of key figures and the criticism of those seen to have usurped the spokesperson role, and also the acrimonious disputes between the OSS and FSF communities, who might also be expected to be on the same side.
- The extent to which OSS actually represents a paradigm shift in the software industry, specifically in terms of whether it subverts software engineering principles, or the general nature and perception of software distribution. In relation to the latter, we consider the extent to which open source software is truly open—the Berkeley Conundrum being the term we have coined for this. The issue at stake is whether software can be truly said to be open if no one actually downloads the source code.
- Whether OSS software is truly high quality software?
- Whether OSS is a 'one size fits all' approach, that is, the extent to which it may be the future model for all software development.

A dialectical approach is adopted in analysing and discussing these issues. The features that would appear to characterise OSS as collectivist, or as a paradigm shift in software engineering, for example, are presented, and these are paired with a discussion of the features that would tend to characterise OSS as the opposite.

2. Tensions and Paradoxes within OSS

2.1. Cathedral v. Bazaar

The conventional wisdom of software engineering suggests that given the inherent complexity of software (Brooks, 1987), it should be developed using tightly co-ordinated, centralised teams, following a rigorous development process (Paulk et al, 1993). Within the software family, the most complex product is probably that of an operating system – IBM's OS360, the subject of Brooks' *Mythical man Month* book was reckoned to be the most complex thing mankind had ever created at the time. Given this complexity, it is reasonable to assume that such a product would require a development process congruent with the fundamental software engineering principles. Raymond (1999) initially categorised this mode of development as a cathedral-style of highly formalized, well-defined and rigorously followed development processes. He contrasted this with a bazaar style of development, which better characterised the open source development approach. The bazaar metaphor was chosen to reflect the babbling, apparent confusion of a middle-Eastern marketplace[2]. In terms of software development, this style does not mandate any particular development approach—all are free to develop in their own way and to follow their own agenda. There is no formal procedure to ensure that developers are not duplicating effort by working on the same problem. In conventional software development, such duplication of effort would be seen as wasteful, but in the open source bazaar model, it leads to a greater exploration of the problem space, and is consistent with an evolutionary principle of mutation and survival of the fittest, in so far as the best solution is likely to be incorporated into the evolving software product (Kuwabara, 2000). Certainly, the bazaar model would appear to be capable of astonishing results. For example, Linux was begun five years after MS Windows NT, with no budget and relying on voluntary contributions; yet, new releases of the Linux kernel were being released more than once per day at one stage.

A cursory inspection might suggest that OSS is characterised by a bazaar development approach. However, it is certainly the case that OSS development is not completely homogeneous. There are significant inter-project differences in the way development is organised in Linux, Apache, and the various BSD open source projects (see Nakakoji & Yakamoto, 2001). Raymond (1999) identifies the significant bureaucratic overhead associated with the BSD projects, for example, where after the patch has been 'cleaned up,' a Change Log entry must be written and the assignment to the Free Software Foundation completed.

Also, there are significant intra-project differences. For example, development of the Linux kernel is organised very differently, and exhibits many characteristics of the cathedral approach, with modifications being co-ordinated through Torvalds himself. On the other hand, the Linux periphery, where applications and utilities are being developed (Dempsey et al., 1999), is actually characterised more by a bazaar development model.

software'. Raymond changed the term to 'open source' on February 9, 1998

[2] Ironically, leading software commentator, Bob Glass confessed that he was initially confused by Raymond's metaphors believing that 'bazaar' connoted commercialism while 'cathedral' sounded more like an ideologically (religiously) underpinned approach such as OSS.

Also given the threat of litigation, a real threat obviously given the SCO group's legal challenge to IBM and others, many open source projects, especially the more successful ones, are choosing to follow a process of legal incorporation for protection. O'Mahony's (2004) study of the GNOME project supports the view that extra constraints are not welcome to the OSS community, as they go against the overall hacker ethos. She reports a tension within the GNOME development community over the fact that the GNOME Foundation control the release coordination.

2.2 Collectivist v. Individualist

The OSS movement is often portrayed as a communistic collectivist approach; Bob Young, the founder of Red Hat adapted the communist manifesto to characterise it as "from the programmers according to their skills, to the users according to their needs" (Young, 1999). Also, Linux has been is described as an "impossible public good" (Smith & Kollock, 1999). Certainly, the massive parallel development—the Linux development community is estimated variously to exceed 40,000 or 750,000 contributors, while another OSS product, the Fetchmail utility, has had more than 600 contributors (Raymond, 1999)—and the devotion of time by skilled programmers without a direct monetary incentive seems to support such a collectivist view. Also, when Linux won an award as product of the year from *InfoWorld*, the editors at *InfoWorld* complained that they were unsure as to whom the award should be presented as there was no legal owner for Linux (Leibovitch, 1999).

Another factor in keeping with the collectivist notion is that there seems to be a requirement for modesty and self deprecation from the originators of open source projects as they have to convince others to volunteer their efforts in the belief that their input is required. That is, if a developer initiating an OSS project conveys the impression that the originators are on top of things and no help is needed, then the project will not get off the ground as an OSS project. In this vein, Torvalds openly sought help with Linux from the outset. Also, the suggestion that all contributions are valued reinforces the appearance of collectivism. Rather than just accepting strong technical coding contributions, the argument is that those who cannot write code can write documentation, fulfil the role of testers, or elaborate requirements. Thus, the traditional hierarchy in IS departments whereby the program coding activity is perceived as 'superior' to the testing and documentation activities is countered in the OSS approach, thus ensuring that these vitally important activities are not undervalued. Also contributing to the collectivist, public good perception of open source software is the fact that it is of huge importance in the so-called developing countries who cannot afford to pay the high prices demanded by the vendors of proprietary software. After all if the average annual salary is \$100, then \$150 for MS Windows is a significant outlay. Again, this ties in with the media portrayal, referred to earlier, of OSS as a David v. Goliath phenomenon, where the poor struggle with the fabulously rich .

However, there is also very strong evidence to support the view that OSS is fundamentally an individualist phenomenon. The closeness between the name Linux and Linus, for example, betrays an individualistic orientation . It is certainly unlikely that Microsoft would choose to market the much trumpeted Windows Longhorn as Billux for example.

Further evidence of individualist orientation is the undeniable fact that the OSS culture is fundamentally a reputation-based one, and is persuasively underpinned by the economics of signalling incentives on the part of individual developers (Lerner & Tirole, 2000). The signalling incentive term is an umbrella one capturing both ego gratification and career concern incentives, both of which are explained next.

The ego gratification incentive operates on the basis of peer recognition. Developers working on traditional development projects may face long delays in getting feedback on their work. After all, the average project development lifecycle has been estimated to be 18 months (Flaatten et al., 1989), and durations of up to five years are not unknown (Taylor & Standish, 1982). Thus, developers experience a significant 'rush' from seeing their code in use more quickly in OSS projects. Also, the recognition they do receive is from peers they truly respect often, rather than from managers and users within their own organisation. Bergquist and Ljungberg (2001) discuss the OSS developer motivational issue also in some detail and they refer to the phenomenon as obeying an attention economy, in that the more attention an OSS developer can attract the greater the enhancement of status that is achieved. Thus, in this context, OSS development may be more akin to *Egoist* Programming as opposed to Egoless Programming, the term coined by Weinberg (1971).

The career concern incentive relates to the fact that working on an OSS project may enhance future job prospects—after all, Linus Torvalds states that his reward for working on Linux has been that he will never have any difficulty in getting a job—his CV, as he puts it, contains just one word: Linux. Another facet of the career concern incentive might be that those participating in OSS projects may get offered shares in commercial companies.

Also, the collectivist notion that all OSS contributions are valued, and that literally thousands of globally-located developers and users contribute unproblematically to open source products does not bear up to examination. In the case of BSD, McKusick (1999) admits rather colourfully that 90 percent of contributions were thrown away, while "the rest were peed upon to make them smell like Berkeley". In a recent study of the Apache project, Mockus et al. (2000) found that almost 85 percent of modification request by users were totally ignored. The same scenario is also borne out in the Orbiten Free Software Survey where it transpires that the top 10 developers (fewer than 0.1 percent of the total number of developers) contribute almost 20 percent of the code base (Ghosh & Prakash, 2000). Alan Cox, a main figure in Linux development, admits that most contributions are worthless, suggesting that they actually support the argument that one should need a license to get on the internet, and that there are a lot of "dangerously half-clued people milling around", and that those of proven ability are well known within each product development community (Cox, 1998). Such evidence is not indicative of a collectivist atmosphere.

Likewise, the typical OSS developer is not an idealistic hacker working for free. Jorgensen's (2001) survey of the FreeBSD project developers found that 41 percent of developers were being paid for their OSS work by the companies in which they were employed.

The seemingly obvious attraction of OSS to poorer institutions in the developing countries is especially interesting as the issue is actually not as simple as it is often portrayed. An excellent description of failed attempts to initiate OSS projects in Ghana by Gregg Zachary (2003) identifies fundamental problems in the widespread belief among Ghanaian programmers and users that nothing of value could be done for free, and he concludes that OSS concepts would need to be considerably 'Africanized' in order to have a chance of success

The positioning of OSS, even if simplistic, as an opportunity for developing countries is however paralleled by the extreme interest of fabulously wealthy institutions in the technologically-advanced countries also. For example, OSS is of enormous interest to an organisation like NASA, the US space agency, who desire complete transparency in the software they use, and believe that by having complete access to the source code, they can test it exhaustively themselves. Likewise the National Security Agency (NSA) have developed their own version of the Linux kernel. These are both examples of organisations who have sufficient resources to purchase any software they wish, and are quite far removed from any ideology of collectivism.

Again, one might expect that a collectivist movement would exhibit liberal values in other aspects. However, the statement from Eric Raymond (2000) that "Linux is about getting freedom, personal firearms are about keeping it" doesn't sit all that comfortably within what one would normally expect of a liberal orientation.

The vast sums of money apparently being made by some OSS players is also not at all compatible with a collectivist, public good orientation. It is inevitable that those developers who have been contributing to OSS and who have not been able to benefit from this financial bonanza will become disillusioned. This is likely to lead to resentment and jealousy, and these issues are at the heart of the OSS v. OSS v. FSF tensions, discussed next.

2.3. OSS v. OSS v. FSF

Also, the delicate equilibrium of the pioneers of the OSS movement being able to co-operate and provide a unified front appears to have faltered. Bruce Perens, one of the originators of the open source term and a significant contributor to establishing the movement through integrating it with the already viable Debian Social Contract (Perens, 1999), quickly resigned from the Open Source Foundations, amidst rumours of negative opinions about other OSS pioneers. Also, Raymond has responded to criticisms that he has usurped too much of the OSS mantle. Thus, it appears that individual egos cannot be as easily set aside in the interest of the movement as a whole as originally expected.

While the debate within the OSS community itself has been somewhat acrimonious at times, so also has there been considerable tension between the OSS and FSF communities. Richard Stallman's status has been eroded due to the highly publicised success of Linux and Torvalds. However, it should be noted that much of what comprises the Linux distribution involves several GNU utilities produced by the FSF, to the extent that Stallman (quite justifiably) insists on terming the overall distribution as GNU/Linux reserving the term Linux for the actual kernel. However, much energy is expended on the nuances of differentiation between two communities who should co-operate to greater effect.

2.4. Is OSS a Paradigm Shift in Software Industry?

2.4.1. A Paradigm Shift in Software Engineering? The proponents of OSS point to the fact that very high quality software is being produced in a rapid time-scale and for free. These three aspects directly address the three main components of the so-called software crisis mentioned earlier. Thus, it would appear that OSS is the 'silver bullet' that can solve these problems. Further icing on the cake comes from the arguments also put forward by the extreme proponents of OSS—that feedback is very prompt, the testing pool is global, peer review is truly independent, the contributors are in the top 5 percent of developers worldwide in terms of ability, and they are self-selected and highly motivated. Given these factors, the argument that OSS truly is the 'silver bullet' becomes even more cogent. However, the truly amazing aspect of OSS is that this 'silver bullet' arises from a process which at first glance appears to be completely alien to the fundamental tenets and conventional wisdom of software engineering. For example, in the bazaar development style, there is no real formal design process, there is no risk assessment nor measurable goals, no direct monetary incentives for developers or organisations, informal co-ordination and control, much duplication in parallel effort. All of these are anathema to conventional software engineering.

Also, OSS appears to reverse Brooks' Law: in a classic treatise on the software development process, Brooks (1975) coined the widely-accepted law that "adding manpower to a late software product makes it later". Brooks' cited empirical evidence to support this from the development of the IBM 360 operating system. Thus, merely increasing the number of developers should not be a benefit in software development. However, the OSS community have proposed their own law, which appears to be at odds with Brooks, namely that "given enough eyeballs, every bug is shallow" (Raymond, 1999). Given these apparently contradictory axioms, one is reminded on Niels Bohr's contention that the opposite of a great truth is also true.

At any rate, the expected problems do not seem to manifest themselves in open source software. OSS products in general, and the Linux operating system in particular, do seem to turn the software crisis on its head in that exceptionally high quality and reliable software is produced in a very rapid timescale and for free. Some evidence to support this claim comes from the fact that Linux development began five years after Windows NT with no budget and relying on voluntary contributions. Yet, new releases of the kernel were being produced at the rate of more than one per day at one time.

However, 30 years of software engineering research cannot be easily discounted. Thus, an examination of the details of the OSS development process serves to question the extent to which software engineering principles are actually being fundamentally overturned. Firstly, the main contributors of the OSS community are acknowledged to be superb coders, suggested by some to be among the top 5 percent of programmers in terms of their skills. Also, as they are self-selected, they are highly motivated to contribute. The remarkable potential of gifted individuals have long been recognised in the software engineering tradition. Brooks (1987) suggests that good programmers may be a hundred times more productive than mediocre ones. The Chief Programmer Team more than twenty years ago also bore witness to the potential of great programmers. Also, in more recent times, the capability maturity model (CMM) recognises that fabulous success in software development has often been achieved due to the "heroics of talented individuals (Paulk et al., 1993). Thus, given the widely recognised talent of the OSS leaders, the success of OSS products may not be such a complete surprise.

The advancement of knowledge in software engineering has certainly been incorporated into OSS software. Linux, for example, benefited a great deal from the evolution of Unix in that defects were eliminated and requirements fleshed out a great deal (McConnell, 1999). Furthermore, some of the fundamental concepts of software engineering in relation to cohesion and coupling and modularity of code are very much a feature of OSS. Linux, by being based on Unix, is very modular in its architecture. Indeed, the manner in which different individuals can take responsibility for different self-contained modules within Linux, is acknowledged as being a major factor in its successful evolution. Further evidence of the importance of modularity arises from the Sendmail utility. This was first developed by Eric Allmen at Berkeley in the late 70s, and the source made available to interested parties. However, as it began to evolve through the contributions of others, problems in integrating contributions began to arise. Allmen resigned from his position and rewrote Sendmail completely to follow a more modular structure. This ensured that it could be a suitable candidate for the massive parallel development, characteristic of OSS, as developers could work largely independently on different aspects. Sendmail has evolved to its current position of dominance—estimated to route 80 percent of all Internet mail. These examples provide much evidence that open source software does obey the fundamental tenets of software engineering in relation to modularity.

Configuration management, another important research area within software engineering, is a vitally important factor within OSS, and is typically catered for by the Concurrent Versioning System (CVS), itself an open source product (Fogel, 1999). Also, the software engineering principles of independent peer review and testing are very highly evolved to an extremely advanced level within OSS.

In summary, then, the code in OSS products is often very structured and modular in the first place, contributions are carefully vetted and incorporated in a very disciplined fashion in accordance with good configuration management, independent peer review and testing. Thus, on closer inspection, the bazaar model of OSS does not depart wildly from many of the sensible and proven fundamental software engineering principles. The argument then that OSS begins as a bazaar with a chaotic development process chaos and evolves mysteriously into a co-ordinated process with an exceptionally high quality end-product is too simplistic a characterisation of what actually is taking place in practice.

2.4.2. Is the Software Truly Open – The Berkeley Conundrum? The inherent invisibility of software (Brooks, 1987) is exacerbated by its distribution in binary form (1s and 0s), which is the necessary format for efficient computer operation. The availability of the actual source code may appear to redress this invisibility. However, one could question the extent to which a large software package—the Linux distribution, for example, contains more than 10 million lines of code—is actually all that different from a binary executable despite its being available in source form. It certainly appears to be the case that the majority of users (perhaps not always given the choice) merely use OSS products in their binary executable form and ignore the source code. Also, the source code of much software sold in binary form has traditionally been made available to the purchasing organisation through an escrow agreement which protects the purchaser in the event of the software vendor not surviving. Thus, the concept of actual availability of source code is not revolutionary. Nor indeed does it appear to be the case that the vast majority of end-users are in a position to take advantage of source code availability, even if they wanted to. We have coined the term, The Berkeley Conundrum , to represent this scenario whereby if users do not actually download the software source code, is it really open?. This would suggest that the contention that OSS represents a paradigm shift in the nature, whatever about the perception, of software distribution is not really sustainable.

2.5. Is OSS High Quality Software?

Proponents of OSS often point to its high quality. Certainly, OSS products are been chosen by the technologically aware who are not susceptible to a sophisticated marketing strategy. Thus, the $100m. which Microsoft are reported to have paid the Rolling Stones for the rights to use their ‘Start Me Up’ song for the launch of MS Windows 95 would not be a worthwhile strategy in the OSS world. Furthermore, the Microsoft Hallowe’en Documents (1998) make it clear that the normal strategy of FUD will not work in relation to OSS, as those selecting the OSS software have such a high degree of technological literacy. Thus, it might be concluded that all OSS products are of very high quality.

However, the situation is not so straightforward. Firstly, Jorgensen’s (2001) study of OSS development revealed that simpler code gets more feedback—generally not all that useful presumably. Also, the same study showed that there was very little feedback on design issues, a very significant deficiency.

Also, the fact that OSS is the choice of the technologically literate, is the source of another problem. Nadeau (1999) gets to the heart of the problem in arguing that proprietary software vendors like Microsoft always have to gear their software to “the most ignorant customers” while OSS developers cater for the “smartest customers”, and can thus cut back on niceties such as a user-friendly interface. This phenomenon would appear to be borne out in the comments of a user who installed Linux and then posted a message on the newsgroup referring to the “thrilling adventure” of that installation.

Another factor relevant to the quality argument is that bug detection may be a chimera in OSS. It is likely to be the case that a sort of inverse Pareto principle is at work in that 80 percent of the bugs may be spotted by 99 percent of the OSS developers, whereas the more difficult 20 percent of the bugs are probably only capable of being identified by about 1 percent of the developers.

Hard evidence in relation to software quality in general, and OSS in particular, is difficult to unearth. However, a study by Stamelos et al. (2001) addressed this specific issue in relation to the SuSE Linux 6.0 release. Using the Logiscope code analysis tool, they examined over 600KLOC across 100 modules in the SuSE release. The results were as follows:

· 50 percent of components acceptable as is
· 31 percent required comments
· 9 percent required further inspection
· 4 percent required further testing
· 6 percent would have to be completely rewritten

These results are really quite average. Only half the modules actually meet the standard generally expected in the software industry!

Similarly, a study by Rusovan, Lawford and Parnas (2004) of the implementation of the Address resolution Protocol (ARP) in the Linux TCP/IP implementation identifies a number of software quality problems.

2.6. Is OSS the Future of Software –One Size Fits All?

Some have suggested that the incremental evolution that characterises the development of OSS is actually quite similar to the development model of a proprietary software company such as Microsoft, in that both follow a model of releasing beta versions and then getting feedback from a large number of customers. Also, the compelling evidence of successful examples of OSS could be used to argue that the approach needs to be more widely used. The argument often advanced is that software has zero reproduction costs but high service and maintenance costs, and the traditional model which is based on a high purchase fee and low maintenance fee is unsuited to the realities of the software business.

However, it is unrealistic to expect all software vendors to surrender the ‘crown jewels’ in the intellectual property that resides in the software source code. Even Raymond’s (1999) classic essays accept that, and such a strategy is evident in the moves by some players in OSS to create a variation on the standard license to achieve this—the Sun Community alliance, for example. Also, it appears that companies such as Apple with their Darwin MacOSX server, Netscape Mozilla, and even Sendmail Pro, the commercial equivalent of Sendmail, are using OSS as a means of achieving some R&D, and then taking any useful updates provided by the OSS community back into their proprietary offerings.

If we consider the provenance of most OSS products, it boils down to Raymond’s (1999) memorable phrase “developers with a personal scratch to itch.” Thus, almost always, OSS projects begin in this way. An examination of the products that have emerged reveals that the successful examples are typically general-purpose, horizontal infrastructure software. This is no accident. Given the finding by Jorgensen (2001), namely, that it is very rare to receive any feedback on design issues in OSS development, it would appear that OSS software is best suited to horizontal domains where design is almost a given, in that there is widespread agreement on design architecture, and the general shape of the software requirements is fairly well known and not problematic. This is probably essential if a large contributor base from a wide variety of industry backgrounds and students in academe are to contribute. On the other hand, in vertical domains where requirements and design issues are a function of specific domain knowledge that can only really be acquired over time—the case in many business environments, in fact—then there are not likely to be any OSS offerings.

3. Conclusion

OSS certainly seems to be capable of generating enormous revenues and profits for some software organisations. However, the question remains as to whether it is truly sustainable from an economic point of view in the long-term, particularly if it is expected to scale up to encompass the overall software industry. The discussion above would suggest that OSS is actually quite closely aligned to fundamental software engineering principles; thus, from that perspective, perhaps, there is less concern. However, a very serious question mark remains in relation to social/human issues. For example, it may not be possible to maintain the delicate balance between the self-deprecation and modesty required to elicit co-operation in an OSS project, and the egoistic motivations that inevitably arise in a reputation-based culture. Add in the prospect of some 'volunteers' making vast sums of money, and the balance becomes even more unstable. Also, even if OSS developers mange to remain largely aloof from all this, the possibility of actual burn-out also is a factor. All of these issues appear to have happened or are imminent within OSS.

However, an even bigger question is whether OSS is merely a transitory software phenomenon (the choice of a GNU generation), or whether it represents the new mode of work for knowledge workers in the electronic age.

4. References

1. Bergquist, M. and Ljungberg, J. (2001) The Power of Gifts: Organising Social Relationships in Open Source Communities, Gothenberg University, Sweden.
2. Brooks, F. (1975) *The Mythical Man-Month*, Addison-Wesley, USA.
3. Brooks, F. (1987) "No silver bullet: essence and accidents of software engineering." *IEEE Computer Magazine*, April, 10-19.
4. Campbell-Kelly, M. (2003) *A History of the Software Industry*, MIT Press, Cambridge, MA.
5. Cox, A. (1998) Cathedrals, bazaars and the town council, http://slashdot.org/features/98/10/13/1423253.shtml (Oct 1, 2000)
6. Dempsey, B.J., Weiss, D., Jones, P. and Greenberg, J. (1999) A quantitative profile of a community of open source Linux developers. *Technical Report TR-1999-05, School of Information and Library Science, University of North Carolina at Chapel Hill.*
7. DiBona, C., Ockman, S. and Stone, M. (1999) Introduction to *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, USA.
8. Feller, J. and Fitzgerald, B. (2000) A Framework Analysis of the Open Source Development Paradigm, *Proceedings of the 21st International Conference in Information Systems*, Brisbane, Australia, December 2000.
9. Flaatten, P., McCubbrey, D., O'Riordan, P. and Burgess, K. (1989) *Foundations of Business Systems*, Dryden Press, Chicago.
10. Fogel, K. (1999) *Open Source Development with CVS*, Coriolis Open Press, USA.
11. Ghosh, R. and Prakash, V.V. 2000. The Orbiten Free Software Survey. *First Monday*, 5:7.
12. Halloween Documents, 1998, "The Halloween Documents," http://www.opensource.org/halloween/ (May 1, 2000).
13. Irwin, R. (1998) "What is FUD?," http://www.geocities.com/SiliconValley/Hills/9267/fuddef.html (May 1, 2000).
14. Jorgensen, N. (2001) Putting It All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project, Roskilde University, Denmark.
15. Kuwabara, K. (2000) Linux: A Bazaar at the Edge of Chaos. *First Monday*, 5:3.
16. Leibovitch, E. (1999) The Business Case for Linux. *IEEE Software*, Jan/Feb 1999.
17. Lewis, T. (1999) The Open Source Acid Test. *IEEE Computer*, Feb 1999.
18. Lerner, J. and Tirole, J. (2000) The simple economics of open source, Harvard Business School Working Paper #00-059.
19. Leonard, A (2000), Salon Free Software Project: Chapter 1 Boot Time, http://www.salon.com/tech/fsp/2000/03/06/chapter_one_part_1.html.
20. Ljungberg, J. (2000) Open source movements as a model of organising, *Proceedings of 8th European Conference on Information Systems,* Vienna.
21. McConnell, S. 1999. Open Source Methodology: Ready for Prime Time? *IEEE Software*, Jul/Aug 1999.
22. McKusick, M. (1999) "Twenty Years of Berkeley UNIX: From AT&T-Owned to Freely Redistributable," in *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, USA.
23. Metcalfe, B. (1999) From the ether, 21 June 1999, http://www.infoworld.com/cgi-bin/displayNew.pl?/metcalfe/990621bm.htm.
24. Mitchell, R. (2000a) "New Numbers, Historic Ramifications," *Wide Open News,* February 10, 2000, http://wideopen.com/story/499.html (May 1, 2000).
25. Mitchell, R. (2000b) "Linux Widens Lead in Web Market Share," *Wide Open News*, April 17, 2000,

http://wideopen.com/story/739.html (May 1, 2000).

26. Mockus, A., Fielding, R. and Herbsleb, J. (2000) A case study of open source software development: the Apache server, in *Proceedings of 22nd International Conference on Software Engineering*, pp. 263-272.

27. Nadeau, T. (1999) Learning from Linux, http://www.os2hq.com/archives/linmemo1.htm (May 1, 2001)

28. Nakakoji, L. and Yakamoto, K. (2001) A taxonomy of open source software development, in Feller, J., Fitzgerald, B. and van der Hoek, A. (eds) *Proceedings of First Workshop on Open Source Software,* ICSE Toronto, 2001.

29. Netcraft survey (2001) The Netcraft Web Server Survey, http://www.netcraft.com/survey/, last accessed May 1, 2001.

30. O'Mahony, S (2004) Non-Profit Foundations and their Role in Community-Firm Software Collaboration, in Feller, J, Fitzgerald, B, Hissam, S, and Lakhani, K. (2004) (Eds) *Perspectives on Free and Open Source Software*, MIT Press, Cambridge.

31. O'Reilly, T. (2000) "Open Source: The Model for Collaboration in the Age of the Internet," *Wide Open News*, http://www.wideopen.com/reprint/740.html (May 1, 2000).

32. Paulk, M., Curtis, B., Chrissis, M. and Weber C. (1993) Capability Maturity Model for Software, version 1.1, *IEEE Software*, Vol. 10, No. 4, pp.18-27.

33. Perens, B. (1999) "The Open Source Definition," in *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, USA.

34. Raymond, E. (1999) *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, US, O'Reilly.

35. Raymond, E. (1999b) "Shut Up And Show Them The Code," *Linux Today*, June 28, 1999, http://linuxtoday.com/stories/7196.html..

36. Raymond, E. (2000) Article in *Atlanta Journal-Constitution*, 21 August 2000.

37. Rusovan, S, Lawford, M and Parnas, D. (2004) Open Source Software Development: Future or Fad? in Feller, J, Fitzgerald, B, Hissam, S, and Lakhani, K. (2004) (Eds) *Perspectives on Free and Open Source Software*, MIT Press, Cambridge.

38. Sessions, R. (1999) A lesson from Palm Pilot, *IEEE Software,* Vol. 16, No. 1, 36-38.

39. Smith, M. and Kollock, P. (Eds) (1999) *Communities in Cyberspace*, Routledge, London.

40. Stallman, R. (1999) "The GNU Operating Systems and the Free Software Movement," in *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, USA.

41. Stamelos, I., Angelis, L. & Oykonomou, A. (2001) Code Quality Analysis in Open-Source Software Development, Aristotle University, Greece.

42. Taylor, T, and Standish, T. (1982) Initial thoughts on rapid prototyping techniques, *ACM SIGSOFT Software Engineering Notes*, **7**, 5, 160-166.

43. Thompson, K. (1999) Unix and beyond: an interview with Ken Thompson, *IEEE Computer*, May 1999, http://computer.org/computer/thompson.htm.

44. Truex, D., Baskerville, R. and Klein, H. (1999) "Growing systems in an emergent organisation," *Communications of the ACM*, Vol. 42, No. 8, pp. 117-123.

45. Weinberg, G. (1971) *The Psychology of Computer Programming*, Rheinhold, New York.

46. Young, R. (1999) "Giving it Away: How Red Hat Stumbled Across a New Economic Model and Helped Improve an Industry," in *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, USA.

47. Zachary, G (2003) Barriers to the formation of open-source software communities in an African city: the case of Accra, Ghana, (personal communication).