5 Has Open Source Software a Future?

Brian Fitzgerald

Open Source Software (OSS) has attracted enormous media and research attention since the term was coined in February 1998. From an intellectual perspective, the concept abounds with paradoxes, which makes it a very interesting topic of study. One example is the basic premise that software source code—the "crown jewels" for many proprietary software companies—should be provided freely to anyone who wishes to see it or modify it. Also, there is a tension between the altruism of a collectivist gift-culture community—an "impossible public good" as Smith and Kollock (1999) have characterized it—and the inherent individualism that a reputation-based culture also implies. Furthermore, its advocates suggest that OSS represents a paradigm shift in software development that can solve what has been somewhat controversially termed 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 quality and reliability of OSS software, its rapid release schedule, and its availability at very little or no cost. Other supporters of OSS believe that it is an initiative that has implications well beyond the software field and suggest that it will become the dominant mode of work for knowledge-workers in the information society. These themes feature in the chapters in this volume by Kelty (chapter 21) and Lessig (chapter 18) earlier, and have also been reported by many others (Bollier 1999; Himanen 2001; Markus, Manville, and Agres 2000; O'Reilly 2000).

However, despite these claims, a closer analysis of the OSS phenomenon suggests that there are a complex range of problematic issues from the triple perspectives of software engineering, general business factors, and socio-cultural issues that serve to question the extent to which the OSS phenomenon will even survive, let alone prosper. This chapter identifies and discusses these challenges. This strategy of identifying these potential

Table 5.1
Problematic issues for OSS from software engineering, business, and sociocultural perspectives

Problematic issues from a software engineering perspective
▪ OSS is not really a revolutionary paradigm shift in software engineering
▪ Not enough developer talent to support increased interest in OSS
▪ Code quality concerns
▪ Difficulties of initiating an OSS development project and community
▪ Negative implications of excessive modularity
▪ Insufficient interest in mundane tasks of software development
▪ Version proliferation and standardization problems
Problematic issues from a business perspective
▪ Insufficient focus on strategy in OSS development community
▪ "Free beer" rather than "free speech" more important to OSS mass market
▪ Insufficient transfer to vertical software domains
▪ OSS a victim of its own success
Problematic issues from a sociocultural perspective
▪ OSS becomes part of the establishment
▪ Burnout of leading OSS pioneers
▪ Unstable equilibrium between modesty and supreme ability required of OSS project leaders
▪ "Alpha-male" territorial squabbles in scarce reputation cultures

"failure factors" in advance is deliberate, in that the OSS movement can address these issues if deemed necessary.

The chapter is laid out as follows. In the next section the specific challenges from the software engineering perspective are outlined. Following this, the potential pitfalls from the overall business perspective are discussed. Finally, the problematic issues in relation to the sociocultural perspective are addressed. These challenges are summarized in table 5.1.

Problematic Issues from a Software Engineering Perspective

OSS is often depicted as a revolution or paradigm shift in software engineering. This may be largely due to Raymond's distinction between the cathedral and the bazaar. Raymond chose the "cathedral" as a metaphor for the conventional software engineering approach, generally character-

ized by tightly coordinated, centralized teams following a rigorous development process. By contrast, the "bazaar" metaphor was chosen to reflect the babbling, apparent confusion of a mid-Eastern marketplace. In terms of software development, the bazaar style does not mandate any particular development approach—all developers 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. This duplication of effort reveals a further aspect of OSS that seems to set it apart from conventional software development—namely, the replacing of the classic Brooks Law ("adding manpower to a late software product makes it later" (Brooks 1995)) with the more colloquial so-called Linus's Law ("given enough eyeballs, every bug is shallow"). Brooks had based his law on empirical evidence from the development of the IBM 360 operating system (reckoned then to be the most complex thing mankind had ever created). Thus, according to Brooks, merely increasing the number of developers should exacerbate the problem rather than be a benefit in software development. However, as already mentioned previously, proponents of the OSS model have argued that it does indeed scale to address the elements of the so-called "software crisis."

OSS is Not Really a Revolutionary Paradigm Shift in Software Engineering

If all the positive claims about OSS were true, then we might expect that OSS could be categorized as a revolutionary paradigm shift for the better in software engineering. At first glance, OSS 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 most developers, informal co-ordination and control, much duplication and parallel effort. All of these are anathema to conventional software engineering. However, upon closer analysis, certain well-established principles of software engineering can be seen to be at the heart of OSS. For example, code modularity is critically important (and also an Achilles heel, as will be discussed later). Modules must be loosely coupled, thereby allowing distributed development in the first

place. Likewise, concepts such as peer review, configuration management and release management are taken to extreme levels in OSS, but these are well-understood topics in traditional software engineering. In summary, 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. Similarly, Linux benefited a great deal (probably too well, as the threat from the SCO Group in May 2003 to take legal action over breach of its Unix patents would suggest) from the evolution of Unix as problems in Unix were addressed over time (McConnell 1999). Overall then, the extent to which OSS represents a radical "silver bullet" in software development does not really measure up to scrutiny.

Not Enough Developer Talent to Support Increased Interest in OSS

The main contributors of the OSS community are acknowledged to be superbly talented "code gods," suggested by some to be among the top five 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 has long been recognized in the software field. Brooks (1987) suggests that good programmers may be a hundred times more productive than mediocre ones. Thus, given the widely recognized talent of the OSS leaders, the success of OSS products may not be such a complete surprise. Indeed, it is more critical in the case of OSS, as the absence of face-to-face interaction or other organizational cues makes it vital that there be an ultimate arbiter whose authority and ability is pretty much above question, and who can inspire others, resolve disputes and prevent forking (more on this later).

However, just as OSS is not actually a paradigm shift in software engineering, it is in fact somewhat reminiscent of the Chief Programmer Team (CPT) of the 1970s (Baker 1972). While the CPT appeared to show early promise, there just were not enough high quality programmers around to fuel it. Similarly, the explosion of interest in OSS and the identification of "itches to be scratched" may not be supported by the pool of development talent available. To some extent one could argue, that this is already the case. For example, the SourceXchange service provided by the major OSS player, Collab.Net, which sought to create a brokering service where companies could solicit OSS developers to work on their projects, actually ceased operations in April 2001. Likewise, a study by Capiluppi et al. (2003) of over 400 Freshmeat.net projects revealed that the majority were solo

works with two or fewer developers, and also with little apparent vitality, as in a follow-up study six months later, 97 percent showed no change in version number or code size.

Code Quality Concerns

The corollary of the previous discussion is obvious. While the early OSS pioneers may have been "best-of-breed" programmers, the eventual programming ability of the programmers who participate in OSS could become a problem. It is well-known that some programmers are net-negative producers (NNP). That is, the project actually suffers from their involvement as their contributions introduce problems into the code base in the longer term. Unfortunately, due to the popularity of OSS, a lot of these NNP programmers may get involved and succeed in getting their poor quality code included, with disastrous consequences. One could argue that OSS is more likely to be vulnerable to this phenomenon, as the formal interviewing and recruitment procedures that precede involvement in commercial software development are not generally part of the OSS process, although the long probationary period served on the high-profile OSS projects clearly serves a useful filtering purpose. However, it is by no means certain that this process could scale to deal with increased popularity of the OSS development mode. Indeed, some studies have now questioned the quality of OSS code in Linux, for example, the chapter in this volume by Rusovan and Lawford and Parnas (chapter 6), and also an earlier study by Stamelos et al. (2001). Other influential figures such as Ken Thompson (1999), creator of Unix, have put the case very bluntly:

> I view Linux as something that's not Microsoft—a backlash against Microsoft, no more and no less. I don't think it will be very successful in the long run. I've looked at the source and there are pieces that are good and pieces that are not. A whole bunch of random people have contributed to this source, and the quality varies drastically. (p. 69)

Tellingly, all these negative opinions are based on analysis of the actual source code, rather than anecdotal opinion. Of course, one could argue that open source is vulnerable to such criticism since the code is open, and that the proprietary code in closed commercial systems might be no better.

Difficulties of Initiating an OSS Development Project and Community

Increasingly, organizations might choose to release their code in an open source fashion. However, simply making large tracts of source code

available to the general development community is unlikely to be successful, since there is then no organizational memory or persistent trace of the design decisions through which the code base evolved to that state. Thus, making the source code of Netscape available in the Mozilla project was not sufficient in itself to immediately instantiate a vibrant OSS project (although some very good OSS development tools have emerged as by-products). In most OSS projects, changelogs and mailing lists provide a mechanism whereby new developers can read themselves into the design ethos of the project, in itself perhaps not the most efficient mechanism to achieve this. Gorman (2003) describes the phenomenon in the Linux virtual memory (VM) management subproject whereby new developers may complain about the use of the buddy allocator algorithm, which dates from the 1970s, for physical page allocation, as they feel the slab allocator algorithm might be better, for example. However, they fail to appreciate the design rationale in the evolution of the project which led to that choice.

Furthermore, the principle of "having a taillight to follow," which often guides OSS development as developers incrementally grow an initial project, may perhaps not be robust enough for development if OSS products are to be produced in vertical domains where domain knowledge is critical (an issue discussed in the next section). Linus Torvalds's apparent inability to successfully manage a small development team at Transmeta (Torvalds and Diamond 2001) suggests that the concept may be too ephemeral and individualistic to provide any continuities to general software project management.

Negative Implications of Excessive Modularity

Modularity is necessary in OSS for a number of reasons. Firstly, as previously mentioned, it allows work to be partitioned among the global pool of developers. Also, as projects progress, the learning curve of the rationale behind requirements, design decisions, and so on becomes extremely steep. Thus, to facilitate the recruitment of new contributors, developers need to be able to reduce their learning focus below the level of the overall project. Modularity helps achieve this; thus, it is a *sine qua non* for OSS. Indeed, many OSS projects were rewritten to be more modular before they could be successfully developed in an OSS mode, including Sendmail, Samba, and even Linux itself (Feller and Fitzgerald 2002; Narduzzo and Rossi 2003). However, the cognitive challenge in designing a highly modular architecture of autonomous modules with minimal interdependencies is certainly not trivial (Narduzzo and Rossi 2003).

Also, the increase in modularity increases the risk of one of the well-known and insidious problems in software engineering: that of common coupling between modules, where modules make references to variables and structures in other modules which are not absolutely necessary. Thus, changes to data structures and variables in seemingly unrelated modules can have major follow-on implications. In this way, OSS systems evolve to become very difficult, if not impossible, to maintain in the long run. Some evidence of such a phenomenon being potentially imminent in the case of Linux may be inferred from Rusovan, Lawford, and Parnas (chapter 6 in this volume), and also in a study of the modularity of Linux (Schach, Jin, and Wright 2002).

Insufficient Interest in Mundane Tasks of Software Development

Many software tasks are of the mundane variety—documentation, testing, internationalization/localization, field support. Although tedious and mundane, these are vital, particularly as projects mature and need to be maintained and updated by new cohorts of developers. The exciting development tasks could be cherry-picked by OSS developers. Despite the hope that nontechnical OSS contributors and users will undertake some of the documentation and testing tasks, this has not really happened; certainly, there is no parallel to the enthusiasm with which code is contributed. However, this is perhaps understandable in a reputation-based culture, where the concept of a "code god" exists, but that of "documentation god" does not. The more rigorous studies of OSS developers that have been conducted recently for example, those reported in earlier chapters in this volume by Ghosh (chapter 2) and Lakhani and Wolf (chapter 1), reveal that altruistic motives do not loom large for OSS developers, certainly if the main beneficiaries of such effort would be outside their immediate community. Furthermore, an earlier study by Lakhani and von Hippel (2003) which analyzed the provision of field support for Apache suggests that the actual cost of providing this service is much lower for developers than one might expect, while it also provides substantial benefits to their own work, which is a significant motivation.

Version Proliferation and Standardization Problems

The many different commercial versions of Linux already pose a substantial problem for software providers developing for the Linux platform, as they have to write and test applications developed for these various versions. Also, in the larger OSS picture, as products have been developed more or less independently, interoperability and compatibility problems

among different product versions pose very time-consuming problems. Smith (2003) reports an exchange with an IT manager in a large Silicon Valley firm who lamented, "Right now, developing Linux software is a nightmare, because of testing and QA—how can you test for 30 different versions of Linux?"

One could argue that standards are of even more critical importance to the OSS community than to traditional proprietary development, since the developers do not meet face-to-face, and any mechanism that can facilitate collective action, such as the development of common standards for integration, must be a welcome one. Indeed, Smith (2003) has written a very compelling argument for the open source and standardization communities to collaborate. The primary reason for the successful dissemination of the Internet and World Wide Web technologies has been adherence to open standards. It is no coincidence that the key components of the Web and Internet are open source software: the BIND domain name server, Sendmail, the Apache web server, and so on. Standards are key to interoperable platforms. However, the outlook on standards in OSS at present is not altogether encouraging. There are a variety of initiatives which are moving broadly in this direction—for example, the Free Standards Group (http://www.freestandards.org), the Linux Standard Base and United Linux (http://www.unitedlinux.com), the Linux Desktop Consortium (http://desktoplinuxconsortium.org), and the Free Desktop group (http://www.freedesktop.org). However, these initiatives are overlapping in some cases, and are not well integrated. Also, the agenda in some cases is arguably not entirely to do with the establishment of open standards.

Problematic Issues from a Business Perspective

Just as there are challenges from a software engineering perspective, there are also fundamental challenges to OSS from the overall business perspective. This is not altogether surprising, perhaps, when one considers that the open source concept was quite literally a phenomenonal overnight success in concept marketing that forced its way onto the Wall Street agenda. However, its origins are largely in a voluntary hacker community, and it is unlikely that such a community would be skilled in the cutthroat strategic maneuvering of big business.

Insufficient Focus on Strategy in OSS Development Community

OSS represents a varied mix of participants, who have very different agenda and motivations for participation—see chapters in this volume by Ghosh

(chapter 2), and Lakhani and Wolf (chapter 1). Also, being loosely connected and pan-globally distributed, there is not really the possibility for the detailed strategic business planning that conventional organizations can achieve. Indeed, there is evidence of some possibly inappropriate strategic choices. For example, despite the high profile, one could argue that competing with Microsoft on desktop applications—where they possess the "category killer" application suite—is an unwise strategic use of resources in the long term. In contrast, Microsoft has abstracted some of the better ideas from OSS and may have muddied the water sufficiently with its "shared source" strategy to confuse the issue as to what OSS actually represents. Recognizing the power of the social and community identification aspects of OSS, Microsoft has introduced the Most Valued Professionals (MVP) initiative, and will extend source code access to this select group. Also, their new Open Value policy extends discretion to sales representatives to offer extreme discounts and zero percent financing to small businesses who may be likely to switch to OSS (Roy 2003). These strategies are clever, especially when allied to studies that appear to show that the total cost of ownership (TCO) of Linux is cheaper than Windows over a five-year period. It is difficult for the OSS community to emulate this kind of nimble strategic action. Also, the dispute between the open source and free software communities over the definitional issues and the relative importance of access to source code has not helped to present a unified front.

"Free Beer" Rather than "Free Speech" More Important to OSS Mass Market

By and large, many software customers may not really care about the ideology of free as in "unfettered" software rather than free as in "zero cost." This is especially salient given the downturn in the economy, and also now that many IT budgets are being drastically reduced in the aftermath of their increased budget allocation in the runup to 2000. For these organizations, zero cost or almost zero cost is the critical condition. Thus, access to the source code is not really an issue—many organizations would have neither the competence nor even the desire to inspect or modify the source code, a phenomenon labeled as the Berkeley Conundrum (Feller and Fitzgerald 2002). Indeed, these organizations are actually unlikely to distinguish between open source software, shareware, public domain software, and very cheap proprietary software (see Fitzgerald and Kenny 2003). Not having any experience with soliciting customer or market opinion, OSS developers are unlikely to perceive these market subtleties, and many OSS developers may not wish to cater for the mass market anyway.

Insufficient Transfer to Vertical Software Domains

The early examples of OSS products were in horizontal domains—infrastructure software and the like. These back-office systems were deployed by tech-savvy IT personnel who were not deterred by FUD tactics or the lack of installation wizards to streamline the process. Also, since these were back-office systems and didn't require major budgetary approval, they were generally deployed without explicit management permission. Indeed, management might well have been very concerned at the departure from the traditional contractual support model with the perceived benefit of recourse to legal action in the event of failure (the legal issue is one which we will return to later).

In these initial OSS domains, requirements and design issues were largely part of the established wisdom. This facilitated a global developer base as students or developers in almost any domain with programming ability could contribute, since the overall requirements of horizontal infrastructure systems are readily apparent. Indeed, Morisio et al. 2003 suggests that the majority of OSS projects on SourceForge are in horizontal domains, developing software that produces other software. However, in vertical domains, where most business software exists, the real problems are effective requirements analysis and design, and these are not well catered for in open source. The importance of business domain expertise has long been known (Davis and Olson 1985; Vitalari and Dickson 1983). Students and developers without any experience in the particular domain simply do not have the necessary knowledge of the application area to derive the necessary requirements which are a precursor to successful development. While much has been made of the number of OSS projects aimed at producing ERP systems, the success of such initiatives is perhaps an open question. The GNUe project which has been in existence for about four years (http://www.gnuenterprise.org) has the worthy goal of developing an OSS ERP system together with tools to implement it, and a community of resources to support it (Elliott 2003), but whether a project that appears (at the time of writing) to comprise 6 core developers, 18 active contributors, and 18 inactive ones can fulfill its goal of producing a fully fledged ERP system is an open question. Certainly, it seems unlikely that a large pool of OSS hackers would perceive this as an "itch worth scratching."

OSS a Victim of Its Own Success

Ironically, the success of the OSS phenomenon is also a source of threat to its survival. The moral of Icarus melting his wings by flying too near the

sun comes to mind. While OSS was a fringe phenomenon, there was a certain safety in its relative obscurity. However, once it entered the mainstream and threatened the livelihood of the established players, the stakes shifted. O'Mahony (chapter 20 in this volume) identifies the incorporation of several OSS projects as resulting from a fear of legal liability. In a litigious culture, this fear appears to be well grounded, as at the time of writing, the SCO Group had sent a letter to 1,500 Fortune 1,000 companies and 500 global corporations advising them that Linux might have breached Unix patents owned by SCO, a potentially serious setback for Linux and open source software in general, as it could cause organizations to postpone deployment of open source software through fear of litigation. However, the extra overhead of incorporation has drawbacks. O'Mahony's study of the GNOME project supports the view expressed by Raymond (2001) 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.

General Resistance from the Business Community

There have been analyses of OSS which have likened it to Mauss's Gift Culture (for example, chapter 22 in this volume). However, in a gift economy, the recipient generally welcomes and accepts the gift. However, there is some evidence that users may not welcome the gift of OSS software. Users may fear being deskilled if they are forced to switch from popular commercial software to OSS alternatives (Fitzgerald and Kenny 2003). Also, IT staff may also have similar concerns about potential damage to their career prospects by switching from popular commercial products to OSS offerings.

There are a number of possible impediments to switching to OSS alternatives, including the cost of transition and training, reduced productivity in the interim, and general interoperability and integration problems. Also, the process may be more time-consuming than the average business user is prepared to tolerate. Personally, this author is aware of an MIT-trained engineer who worked for many years at the extremely high-tech Xerox PARC, and despite this impeccable technological pedigree, admits to spending about 17 hours installing a complete OSS solution of Linux and desktop applications, a process that required much low-level intervention, and he was still left with niggling interaction bugs between the components at the end of the process. Although the user-friendliness of

OSS installation is growing daily, these obscure difficulties will frustrate users for whom time is their most precious commodity.

Problematic Issues from a Sociocultural Perspective

Finally, there is a set of challenges to OSS from the sociocultural perspective. These are possibly the most serious challenges, as they are probably the most difficult to detect and counter in that they get to the heart of human nature and social interaction.

OSS Becomes Part of the Establishment

Undoubtedly, OSS has been attractive for many because of its antiestablishment image, and the brightest and most creative young minds have been naturally attracted to it. Iacono and Kling (1996) identify traits, such as being counter-cultural and challenging the status quo, as important for technology movements. However, as OSS has become more popular and mainstream, these bright young anarchists are likely to be far less interested. Also, as the skill level of the contributors diminishes, the badge of pride associated with being part of the community is greatly diminished. While the studies in this volume by Lakhani and Wolf (chapter 1) and Ghosh (chapter 2) do reinforce the notion of young participants in OSS projects, Ghosh's study reveals a more conformist family orientation as a significant component of OSS development now. History provides several examples of radical movements which became subsumed into the mainstream quite quickly—French Impressionism in the nineteenth century, for example.

Also, as money enters the equation in the context of record-breaking IPOs and huge investment by large corporations, the desire for financial reward could further upset the equilibrium. Red Hat, trumpeted as the patron of the open source software movement in the early days, could become the dominant monopoly in the market-place, which would raise its own problems. Moreover, the prices for the high-level packages and proprietary add-ons, together with increasingly restrictive conditions that some OSS providers are imposing, is increasingly bringing them in line with commercial software (Roy 2003). This may result in an Orwellian *Animal Farm* scenario, a story that began with a clear separation between the "good" animals and the "bad" humans, but that eventually progressed to a point where it became impossible to distinguish between the animals and the humans. Likewise, it may become increasingly difficult to distinguish the notional OSS "good guys" from the notional "evil empires" of commercial proprietary software.

Burnout of Leading OSS Pioneers

There is an obvious danger that the leading pioneers will burn out. Not just from the excessive workload—Linus Torvalds is estimated to receive at least 250 emails per day concerning the Linux kernel, for example—but as family commitments arise for a greater proportion of developers, it will be harder to commit the time necessary to lead projects. Also, if these pioneers are primarily in it for the passion, challenge, freedom, and fun, then as the phenomenon becomes more popular, these characteristics get downplayed far more. The threat of legal action has become much more of a reality now, making OSS development a far more stressful affair than in the past.

Unstable Equilibrium Between Modesty and Supreme Ability Required of OSS Project Leaders

OSS pioneer developers need to be modest to ensure that others will contribute. Indeed, Torvalds's initial email posting in 1991 inviting others to help develop Linux is a model of modesty and humility. If other potential OSS developers think their contribution is unnecessary or would not be welcome, they would not be motivated to help, and the project would never get off the ground. However, in addition to modesty and self-deprecation, OSS leaders need to be superbly talented and charismatic. The greater the perceived talent of OSS project leaders, the less likely that their authority will be questioned when they arbitrate on disputes, choose between competing contributions, set the direction for the project, and generally prevent forking. In the absence of rewards and incentives that apply in traditional software development, the supreme authority of a "code god" leader is important, especially given that developers may be distributed across different cultures and countries. However, this mix of social skills, modesty, charisma, and superb talent are not ones that are in common supply in any area of human endeavor, let alone the software arena.

Alpha-Male Territorial Squabbles in Scarce Reputation Culture

OSS is fundamentally a reputation-based economy, and the initiator of an OSS project potentially attracts the greatest reputation, so egoism is very much part of the mix. Unfortunately, as already mentioned, OSS is a male-dominated preserve. While at some levels it is presented as a collectivist Utopia, analysis of the mailing lists reveal a good deal of heated and robust dissension (chap. 22 this volume). Also, Michlmayr and Hill (2003) reveal that on the Debian project there is some resentment about the use of

nonmaintainer uploads, as these are generally interpreted as a sign of the primary maintainer not performing the task adequately. Michlmayr and Hill report that this stigma was not a part of Debian in the past, and suggest that it may be due to the growth of Debian from 200 to 1,000 developers. This possibility is in keeping with the general argument here that increased popularity and growth in OSS projects will contribute to the onset of such problems. The potential for discord is great. Already, there have been some complaints by OSS contributors about rejection of their code contributions on some projects (Feller and Fitzgerald 2002).

At a higher level, the internal dispute in the OSS community itself, with the well-publicized disagreement between the founders, does not augur well for the successful future of the movement, especially when added to the wrangling between the open source and free software communities over the intricacies and fine detail of definitional issues, which are increasingly less relevant as the OSS phenomenon continues to evolve.

Also, reputation may be a scarcer resource that may scale far less than first anticipated, in that only a small handful of people may actually achieve widespread name recognition. One of the findings of the FLOSS study (chap. 2, this volume) was that respondents were as likely to report knowing fictional developers (made up for the study) as much as actual developers when it got beyond the first few well-known names. This is likely to further stoke the flames of competition.

Conclusion

The OSS phenomenon is an interesting one with such enormous potential, not solely from the software perspectives, but also in its role as a catalyst for the new organizational model in the networked economy, and as an essential facilitator in creating the open information society and bridging the "Digital Divide." Other chapters in this book have addressed these issues eloquently, and it is this author's fervent hope that the OSS phenomenon survives, prospers, and delivers to its complete potential. This chapter has been written in the spirit of promoting more critical discussion of OSS and identifying the challenges that may need to be overcome.