MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

National Institute for Space Research (INPE), Brasil

IST
School of Information Sciences and Technology
Building Leaders for the Digital Global Economy

School of Information Sciences and Technology (IST), Penn State University

Joint Research Project on "Public Policies for Open Source Software and Open Data Access"

Working Paper Series, #1

April, 2005

Draft Version 3, 30 April 2005

Structural Constraints in Open Source Software Development and their Public Policy Implications

Gilberto Câmara
Image Processing Division - Brazilian National Institute for Space Research
São José dos Campos, SP, Brazil
gilberto@dpi.inpe.br

Frederico Fonseca
School of Information Sciences and Technology - The Pennsylvania State University
State College, PA, USA
ffonseca@ist.psu.edu

1. Introduction

The development of open source software (OSS) has received a substantial attention recently. Following the successful examples of projects such as Linux, Apache and Perl, there has been a substantial interest by policy-makers and researchers on the dynamics of the production of open source software (Benkler 2003). A topic of particular interest is the adoption of open source software systems in developing nations, as a means of reducing licensing costs and of promoting indigenous technological development, by having access to the source code of these systems. A recent on intellectual property rights and international development commissioned by the government of the United Kingdom underpins such policies with an explicit recommendation:

> *"Developing countries and their donor partners should review policies for procurement of computer software, with a view to ensuring that options for using low-cost and/or open-source software products are properly considered and their costs and benefits carefully evaluated" (Barton et al. 2002).*

Many studies that discuss the development of open source software portray an idealized view, taking OSS to be a product of a committed group of individuals. These individuals would operate on a distributed network, where each programmer works on a small but meaningful module. The programmers are isolated, communicating by means of a central repository and mailing lists. The incentives to participate operate on an individual level (Weber 2002). Some authors go as far as identifying in open source software a new mode of organizational structure, denoted by *commons-based peer production* (Benkler 2003). Others claim that the globally distributed skill induced by open source will loosen the grip of the richest countries on innovation (Kogut et al. 2001). This article takes a critical appraisal of these idealistic views. We consider them untenable, because of the *structural* characteristics of software development. We argue that there are two defining properties of any open source software: the potential for *reverse engineering* and the potential for *distributed development*. These properties vary widely for different types of software products. The Linux model of a widely distributed developer community requires both conditions to be fully satisfied. Otherwise, developing successful OSS requires completely different strategies. By assessing the adherence of each type of software project to these two conditions, we can build a taxonomy for open source projects, and we establish more realistic policies to promote the use of open source.

We consider the following questions: (a) What are the structural factors for OSS development? (b) How do these conditions influence the sustainability of open source projects? (c) What are the consequences of these finding for policies that promote the use of OSS in developing countries?

2. Structural Factors in Open Source Software Development

Decades of experience indicate the hardest parts of software production are achieving a clear *conceptual design* (Brooks 1982) and establishing a feasible strategy for *modular development* (Parnas 1972). The open source movement has not refuted this overall panorama of software development, and therefore these are the two key limiting factors for construction of a successful product: the previous existence of conceptual designs of similar products (the potential for *reverse engineering*) and the problem granularity (the potential for *distributed development*).

The issue of reverse engineering arises naturally when designing a new software product. In segments where a strong consolidation has already taken place, a single product will have a very large of the commercial market share, as in the case of personal productivity suites (where Microsoft's MS-Office is dominant). In other areas, such there are already established standards, as SQL for relational database management systems. In these circumstances, open source developers will be aiming directly at a 'market substitution' strategy and will develop a product that maintains, as much as possible, the features of the leader on the commercial sector. In this case, there is a strong incentive for newcomers to license their products as open source. The potential for reverse engineering is largely dependent on two components:

- The *post-mature* component: a private company develops a software product, for with it holds full intellectual property rights. As this product becomes popular, its functionality and conceptual model becomes well established, and it becomes part of the "*public commons*". The popularity and usability of the software motivates other institutions to develop a public domain equivalent, as in the case of *Open Office*.
- The *standards-led* component: The establishment of standards consolidates a technology and allows compatible solutions from different producers to compete in the marketplace. An example is the SQL database standard, which has motivated products such as *mySQL* and *PostgreSQL*. Another example is the POSIX standard for operating system interfaces, which has served as a guidance to *Linux*.

The second factor affecting software development is the *potential for distributed development*, which is dependent on the software structure. In a simplified view, a software product has a kernel and additional functions that use it (its periphery). An operating system such as Linux has a well-defined kernel for process control and a periphery consisting of programs such as device drivers, applications, compilers and network tools. By contrast, database management systems have a strong kernel of highly integrated functions (such as the parser, scheduler, and optimizer) and a much smaller periphery. Therefore, each type of software product has a *periphery/kernel* ratio that constrains the potential for distributed development, since the kernel requires a tightly-organized and highly-skilled programming team. This situation is consistent with empirical studies that strongly dismiss the idealized conception of open source projects as based on a loose network of developers operating worldwide. Out of more than 400 developers, the top 15 programmers of the Apache web server contribute 88% of added lines (Mockus et al. 2002). Fitzgerald (2004) calls these top programmers 'code gods' and considers that overcoming this problem one of the challenges of OSS. Others (Sagers 2004) have a more positive attitude towards this ratio (few highly skilled/many average-low skilled programmers) and think that restricted access to main parts of the code improves coordination, which affects positively the success of a software project.

Assessment of these two factors helps our understanding of open source software and aids policymakers in establishing appropriate policies to promote its use. We present our model in Figure 1, where we recognize four types of open source software projects:

- *High reverse engineering, high distribution potential (the High-High case)*: here we find the most prototypical open source projects, those that fit the Linux model. Many of the developers will have a separate job, and do their work in their "spare" time, or in time allocated in agreement with their employer. We call them *community-led projects.*
- *High reverse engineering, low distribution potential (The High-Low case)*: here we find a large number of projects, including databases, office automation tools, and web servers. This segment has a large presence of private companies, which aim at entering the marketplace with products similar to the commercial

market leaders. These companies benefit from the reduced risk involved in reverse engineering. There can be outside collaborators, but the main design decisions take place within the institution and in some cases should also address the commercial objectives of these corporations. We call them *corporation-led* projects. Examples include the mySQL and PostgreSQL database management systems, and the GNOME user interface from Ximian corporation.

- *Low reverse engineering, high distribution potential (the Low-High case)*: projects with a high-degree of innovation (usually there is no commercial counterpart) and that share a relatively simple software kernel. They originate in academic environments by researchers and graduate students. Examples include the GRASS GIS software and the R suite of statistical tools. We call them *academic-led* projects.
- *Low reverse engineering, low distribution potential (the Low-Low case)*: usually developed by small teams under a public R&D contract, targeting a niche application and addressing specific requirements, or aiming to demonstrate novel scientific work. They have a very high mortality rate, since most of them are restricted to the lifetime of a research grant. We call them *innovation-led* products.

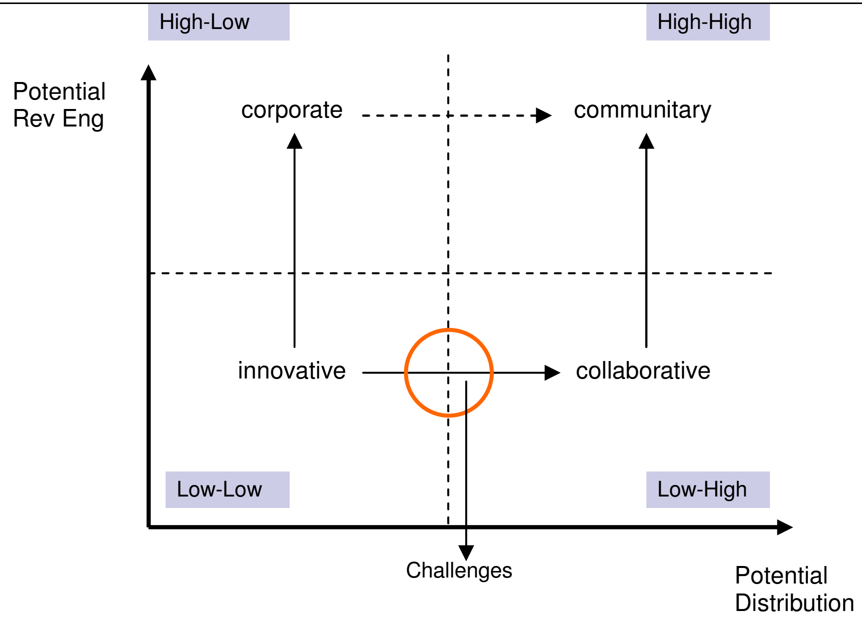

Figure 1 – A Structural Model for Open Source Software Development

A software project, in its lifetime, may migrate between these categories. An *innovation-led* product might evolve to a *corporation-led* one by incorporating characteristics of market products. Such is the case of the PostgreSQL database management system, which derives from a Berkeley research project (Stonebraker et al. 1986) with added support for the SQL standard and market requirements. A *corporation-led* software might evolve into an *community-led* one if their original developers make the necessary investment and adjustment in intellectual property rights to make it accessible to a larger community. This is case of the *Mozilla* browser and associated tools, originally from Netscape. The Apache web server is an example of an innovation-led project that evolved to a community-led one. A team of programmers decided to take the source code of the National Center for Supercomputing Applications Web server, update it, and release it to the public. It is renamed the "Apache" Web server because of all the patches used to upgrade it.

3. Sustainability of Open Source Projects: A Structural Perspective

One of the more interesting consequences of the structural perspective for OSS is that it provides a way to assess the sustainability of projects. This would allow policy makers to take a more active standpoint in supporting open souce.

High-high

The "high-high" situation is the archetypical open source project. It arises when a software project has both a stable design (usually arising from a standards-led situation) and when it is structurally possible to break it in many independent modules that are suitable for large-scale team development (Narduzzo et al. 2005). This is the case of Linux, where developers had a stable design standard as a basis for the project (the POSIX standard) and where the very simple and efficient kernel allowed the concurrent development of drivers for external components such as hardware devices. Additionally, the close relationship of Linux to other UNIX flavors such as BSD allowed the easy conversion of a whole suite of applications, such as BIND, sendmail, and the GNU software tools (Oram et al. 1995). However, there are strong limits to large-scale modularity in most software projects. In his classic book *The Mythical Man-Month*, Frederick Brooks stated his famous law: "*Adding people to a late software project just makes it later*" (Brooks 1972). His chief argument was the added costs of communication between any new software developer and the group he joins. Therefore, in order for the communication costs to be minimal, the design has to minimize communication overhead among group members, a situation that requires a very careful module design, which is not realistic in practice. As Brooks states in another classic work ("*No Silver Bullet*"), software design is very hard because the state space of a medium-scale software project is much larger than the human capacity to model it (Brooks 1982). To sum up, the "high-high" situation is very difficult to achieve. Indeed, it would be counterproductive if all open source projects would fit into this category, since there would be very little innovation coming out of the open source movement, that would be limited to reverse-engineering existing designs or following accepted standards.

High-low

The "high-low" arises in two situations mentioned above: when a commercial software has a large market-share or when a software technology becomes stable enough to for standards to appear. When a single commercial product has a very large part of the market, as in the case of personal productivity suites, switching costs will prevent a new commercial product from capturing market share, even if sold at smaller prices. In this case, there is a strong incentive for newcomers to license their products as open source. When a standard is established, as in the case of the SQL language for relational database management systems, the design effort is reduced for the developer and the switching costs are minimized for the user. In both case, developing an open source product may be part of a private company's business strategy and not a community-led effort.

Low-High

The "low-high" situation occurs when a network of developers produce innovative software on a collaborative basis. This situation requires a combination of factors: a technical community which has consolidated links (they may meet regularly at scientific conferences), a knowledge domain whose basis is stable, and a product whose design allow scalability. One prime example is the R suite of statistical tools. The basis for this software is the commercial product S-Plus®, whose elegant and simple design (Chambers 1998) enabled the statistical community to design the R suite tools (Ihaka et al. 1996), based on the same basic commands as S-Plus®. Given a stable, well-documented design, the statistical community has extended the basic R functionality into a large set of tools. Other examples on this quadrant include the GRASS GIS suite of programs.

Low-low

The “low-low” case occurs in many scientific areas, where products are usually associated to research projects leading to innovative results. The open source license is the natural way for distributing a software prototype produced by a research institution. These products are mostly prototypes demonstrating the feasibility of a new design and are not design for commercial use, in many cases lacking end-user tools such as adequate documentation. In order to take them to the marketplace, their innovative features require a large investment in issues such as documentation and reliability. Such investment may be outside of their developers’ resources. The research community is usually not interested in a direct involvement in long-term open source projects, and maintaining and supporting an open source software project requires considerable resources, beyond the reach of most academic research groups. These projects have a very high mortality rate, since most of them are restricted to the lifetime of a research grant. In many cases, in order for a research prototype to evolve into an open-source product, a team of developers must be taken over from the original research team and established as support and maintenance infrastructure for the product. Therefore, it is very unlikely that an open source project that stays in the “low-low” will succeed. Therefore, although many open source projects may start on the “low-low” quadrant, they must to migrate to more favorable situations. Migration to the “high-low” quadrant occurs frequently when a commercial company decides to use a market strategy based on open source licensing, and takes over the development, as discussed above. Migration to the “low-high” quadrant depends on the stabilization of the software’s kernel and the adoption of the product by a community of individuals. These issues are discussed further in the next section.

4. OSS Structural Constraints and Project Sustainability

The structural constraints discussed above have important consequences for public policy, especially in the developing countries. In order to be able to benefit from OSS development, the policies must be appropriate for each particular situation. In this discussion, we define software project sustainability as "the capacity of a software project to adapt to major changes in its current team and financial support structure".

Dealing with the High-High Situation

This is the simplest case, since products in this range usually have a large community of developers, which is capable of dealing with major changes in team structure. It is conceivable that, in the unlikely event that Linus Thorvalds would resign from his rôle as the chief programmer of Linux, that there would be qualified replacements for the job. Therefore, developing countries are safe to assume that adoption of "high-high" OSS is a safe and sustainable option.

Dealing with High-Low Situation

This situation presents a large challenge to developing nations and policy makers worldwide. As explained above, a large proportion of OSS products in this category are associated with private companies. The programmers have a full-time job as software developers for a company, which in turn will be dependent on revenues associated to services it might provide. A prime example is the mySQL® relational DBMS, which is a product of a private company. This is a situation where the open source credo is not fully applicable, since the open source users may become as dependent on a private company as in the case of proprietary software. Should that company's business strategy fail and the project be terminated, its users would face a difficult situation. If possible, developing nations should avoid adopting "high-low" software products whose long-term sustainability is doubtful, especially if these products are strongly associated to private companies.

In some areas, there are few current alternatives to “high-low” software, as in the case of the Open Office suite. In this case, it is important to address the question of governance models associated to such products. It is being increasingly recognized that the governance model of an OSS product is just as important as the product itself (Franck and Jungwirth, 2002). There has been an increasing emphasis on governance models that increase the power of stakeholders in the software and reduce the main developers’ capacity for independent decision. In this case, by actively participating as stakeholders in such governance boards, developing nations could reduce their liabilities when adopting “high-low” software produced by private companies.

Dealing with the Low-High Situation

The low-high situation represents a favorable condition, since the modularity of the software design and the existence of an established community indicate that the software projects in this area will be sustainable. Since most of the new developments in this area are extensions of the kernel, the product tends to grow without major risks. The main challenge here for developing nations is the amount of expertise required to use these software products, since they contain a fair amount of innovation. For example, in order to benefit from the set of applications available in the R suite of statistical tools, users in developing nations need to be technically proficient in advanced statistics techniques. This requires policy makers in the developing world to be aware that significant investments are needed in human resources, if the “low-high” OSS products are to make a significant impact in their nations.

Dealing with the Low-Low Situation

The low-low situation affects developing nations in two different contexts. First, users in developing nations may be tempted to adopt products in this category that have been produced by researchers in he developed world. Since “low-low” project tend to unlikely to be sustainable in the medium term, their adoption entails a significant risk. Before adopting such software on a larger basis, software developers must assess the likelihood that these projects migrate to the "low-high" or the "high-low" quadrants. If sufficient resources are available in a developing nation, a team of skilled local programmers could envisage to undertake the task of establishing a stable product from a research prototype.

A second situation that arises frequently is the case of projects initiated in developing nations. These projects might be financed by public grants, usually associated to local research groups. Unaware of the structural characteristics of OSS products, policy makers might naively believe that, after an initial incentive, an OSS product will blossom by itself. In many cases, after an initial one to three year grant, the project might die out, without attracting a large enough community (or a commercial company) that would ensure long-term sustainability.

5. Public Policy Implications of the Adoption of OSS

The preceding sections have examined the nature of open source software development and outlined the main characteristics of its production. We have argued that most mature and successful OSS products require the establishment of organizational structures dedicated to their production. The consequences for developing nations are significant. Many developing nations are currently actively considering policies to support or enforce the adoption of OSS by public institutions (Dravis 2002). The arguments in favor of OSS adoption by public institutions include (Ghosh et al. 2002):

- *Lower cost:* adoption of personal computers based on OSS for public use can reduce initial entry cost by as much as 50%.
- *Independence from proprietary technology*: many governments are increasingly concerned with over-dependence in some important markets to a small number of vendors.

- *Availability of efficient and low-cost software:* the virtuous examples of some products (such as Linux and Apache) have encouraged statements about the widespread availability of OSS software for public use.
- *Ability to develop custom applications and to redistribute the improved products*: Given the "open" nature of OSS, skilled local programmers could adapt the software to fit local needs, and thus increase the efficiency of the services provided by the improved products.

While the authors consider that there is enough empirical evidence to support the first two claims, the issues regarding "software availability" and "ease of customization" are far more problematic and require a much closer examination. Most successful open source software tools are infra-structural products, such as operating systems, programming languages and web servers. By contrast, the number of mature OSS that support end-user applications is much smaller (Schmidt et al. 2002). Operating systems, compilers and Web servers respond to the needs of technically qualified IT professionals, who can more easily adapt to the demands of products where support might only be available on the Internet, and requires expertise in the English language.

Additionally, there are inherent market failures and cultural issues in open source software production, which limit the availability of the products needed by developing nations. Therefore, if governments in developing nations aim to profit from the potential benefits of open source, they must intervene and dedicate a substantial amount of public funds to support the establishment and long-term maintenance of open source software projects.

Finally, the issue of social production of technology should be addressed, especially in regards to developing nations. The naïve view of open source products is concerned only with the software development process, with limited regard for its usage. Many open source developers take the view that since their product is superior to commercial ones, it will be automatically be adopted by potential users. In real life, the development and user communities are different and most users have limited technical knowledge. Concerns such as documentation, local support, training material and best-case examples dictate user choice. In developing nations, language barriers are an additional limiting factor. As a result, the effort needed to place open source software in the hands of users worldwide very often falls completely outside the capabilities of committed programmers teams.

As we discussed before, there is a dual role for OSS in developing countries. Government policies need to address both OSS as a technology and as a final product. The example of Linux is not reproducible in all situations. In developing countries there will be plenty of situations with a low-low profile regarding the potential for reverse engineering and distributed development. This is the case with applications in education, public health, environment, and security. There are inherent market failures and cultural issues in open source software production, which limit the availability of the products needed by developing nations. Therefore, if governments in developing nations aim to profit from the potential benefits of open source, they must intervene and dedicate a substantial amount of public funds to support the establishment and long-term maintenance of open source software projects. Only with an assured long-term support, "low-low" projects might migrate to a "low-high" or to a "high-low" situation, and thus increase their long-term sustainability.

6. Conclusions: Public Policies for OSS in Developing Countries

The role of government in developing countries is an important one. First, government has a strong buying power that can drive the market. Second, state sponsored universities are predominant and the most important source of research funds is the government. Third, as Wilson (2004) argues in his analysis of the struggle of developing countries to follow the information revolution, political institutions and policies at the national level are as important as technology.

Nevertheless the role of OSS for developing countries cannot be restricted to government mandated use of Linux as have been reported recently (Rossi 2004). For instance, the Brazilian government is recommending that its agencies have Linux installed in all new computers from 2004 on. In Thailand, the government is aiming at having 5% of its computers running Linux. OSS has a much more important role. OSS can help developing countries master the technology of software development and support the development of applications that leverage local knowledge. Therefore, development policies should address the broader aspects of OSS.

For instance, Sagasti suggests some principles to guide the design and implementation of strategies to create and acquire endogenous science and technology capabilities in developing countries (Sagasti 2004). He states: "strategies and policies for establishing an endogenous science and technology base must be fully incorporated into the design of a comprehensive development strategy for the country" (p.85). Isolated technology projects have less chance to succeed or at least to be sustainable in the long run. Since OSS is a technology from which tangible benefits can be harvested early, its integration on long-range policies is more likely to happen. OSS can be used to build products that will give a large portion of the population access to information that it would not have otherwise. These kinds of products are likely to have a positive impact on the public opinion making it easier for government to include support for OSS in its developmental policies.

Another principle suggested by Sagasti (2004) is that "the cumulative process of building endogenous science and technology capabilities requires continuous and sustained efforts over a long time" (p.86). This principle brings the problem of sustainability. Is OSS sustainable as a long-range development plan for developing countries? In order for development projects to be sustainable it is necessary to incorporate indigenous knowledge and techniques in the process of implementation of new technologies. Although OSS has a great potential for doing this it also needs support in terms of having government-funded research and training.

An analysis of the challenges facing OSS (Fitzgerald 2004) suggests other directions for policies in developing countries. Fitzgerald mentions the key role of project leaders. These are individuals with leadership and programming skills. Policies need to address this important point providing for selection, training, and support of leaders that will help bring together two kinds of knowledge, technical and practical. Project leaders will embody emancipatory knowledge. They will help disseminate technical knowledge and will make sure that local knowledge is embedded in the products of software development.

The view of OSS as a product of a team of committed individuals is not realistic. Most products are built either by a very small team of individuals or by corporations. Large collaborative networked teams are responsible for a small number of products. Additionally, most projects aim at reverse-engineer existing designs or at complying with standards. Given the constraints in open source software production, such advances will not happen spontaneously and will require public intervention to fund innovation. Open source software in developing nations needs strong and wise policies to be successful. It is a combination of institutional vision, qualified personnel and strong links to user community. OSS in developing countries needs to be government-funded to be viable.

References

1. Barton, J., Alexander, D., Correa, C., Mashelkar, R., Samuels, G., and Thomas, S. "Integrating Intellectual Property Rights and Development Policy," Commission on Intellectual Property Rights, UK Department for International Development, London.

2. Benkler, I. "Coase's Penguin, or, Linux and The Nature of the Firm," *Yale Law Journal* (112:Winter 2002-03) 2003.

3. Brooks, F. *The Mythical Man-Month.* Wesley Publishing Company, Reading, MA, 1972.

4. Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*) 1982.

5. Chambers, J.M. *Programming with Data.* Springer-Verlag, New York, NY, 1998.

6. Dravis, P. "A Survey on Open Source Software," The Dravis Group, San Francisco, CA.

7. Fitzgerald, B. "A Critical Look at Open Source," *IEEE Computer* (37:7) 2004, pp 92-94.

8. Ghosh, R.A., Krieger, B., Glott, R., and Robles, G. "Open Source Software in the Public Sector: Policy within the European Union," International Institute of Infonomics, University of Maastricht, The Netherlands, Maastricht.

9. Ihaka, R., and Gentleman, R. "R: A Language for Data Analysis and Graphics," *Journal of Computational and Graphical Statistics* (5:3) 1996, pp 299-314.

10. Kogut, B., and Metiu, A. "Distributed Knowledge and the Global Organization of Software Development," MIT Working Paper, Massachusetts Institute of Technology, Boston, MA.

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

12. Narduzzo, A., and Rossi, A. "The Role of Modularity in Free/Open Source Software Development," in: *Free/Open Source Software Development,* S. Koch (ed.), Idea Group, Hershey, PA, 2005.

13. Oram, A., and Loukides, M. *Programming with GNU Software* O'Reilly, Sebastopol, CA, 1995.

14. Parnas, D. "On the Criteria to be Used in Decomposing Systems into Modules," in: *Communications of the ACM*, 1972, pp. 1053-1058.

15. Rossi, M.A. "Decoding the "Free/Open Source (F/Oss) Software Puzzle" a Survey of Theoretical and Empirical Contributions," *Quaderni dell'Istituto di Economia* (424) 2004, pp 1-40.

16. Sagasti, F.R. Knowledge and Innovation for Development: The Sisyphus Challenge of the 21st Century E. Elgar, Cheltenham, UK, 2004.

17. Sagers, G.W. "The Influence of Network Governance Factors on Success in Open Source Software Development Projects," Proceedings of the The 25th International Conference in Information Systems (ICIS 2004), 2004, pp. 427-438.

18. Schmidt, K.M., and Schnitzer, M. "Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market," Seminar for Economic Theory, Ludwig Maximilian University, Munich.

19. Stonebraker, M., and Rowe, L.A. "The Design of POSTGRES," in: *ACM-SIGMOD International Conference on the Management of Data*, Washington, D.C., 1986, pp. 340-355.

20. Weber, S. "The Political Economy of Open Source," BRIE Working Paper 140, University of California, Berkeley, CA.

21. Wilson, E.J. The Information Revolution and Developing Countries MIT Press, Cambridge, Mass, 2004.