Why Use QuantLib?

N. P. Firth

OCIAM, Mathematical Institute, 24-29 St. Giles', Oxford. OX1 3LB UK
firth@maths.ox.ac.uk
Summertown Solutions Ltd., Suite 140, 266 Banbury Road, Oxford. OX2 7DL UK
neil@sumsol.co.uk

Abstract. As the open–source movement gains momentum more projects are emerging in the domain of mathematical finance. There are a number of pricing libraries available for financial derivatives, including QuantLib. QuantLib has an effective project structure and is achieving a critical mass of users and developers. Many user groups would benefit by adopting the project. [1]

1 Introduction

For most people working in the field of mathematical finance it becomes necessary to code up derivatives pricing algorithms. While it is possible to design and write new code independently it is more powerful, and sometimes easier, to use and build upon existing projects, if these projects are well implemented.

There are a number of libraries of varying quality for pricing financial derivatives available on the internet. We look at the aims of these projects, and at the aims of open–source projects in general. We consider only open–source libraries, as they allow implementations of algorithms to be peer reviewed, and do not require any licensing fees. At this time open–source libraries for pricing financial derivatives do not cover as many products, or implement as many models as closed–source packages. For a survey of closed–source software providers see Davidson (2004). Most real derivative pricing systems are implemented in C++ however, software tools such as Mathematica (2004) or Matlab (2004) are used by some companies for particular applications. These softwares are more useful for prototyping and are proprietary and therefore, beyond the scope of this paper.

In section 2 we discuss some of the terms and issues involved in open–source software development, and survey existing projects for financial derivatives. In section 3 we look at the different motivations of participants in open–source projects, and which are relevant to financial mathematics. In the following section (section 4) we relate these issues to the QuantLib project. We look at potential user groups and briefly discuss the practicalities of contributing to QuantLib. Conclusions are given in section 5.

2 Open–source derivatives libraries

Open–source software can be developed under one of several licenses and hosted on internet servers in various ways. The choice of license and project structure has an influence on the success of the project. Existing derivatives libraries have differing project structures.

[1] This paper last updated June 7, 2004

2.1 Free software

The profile of free software has increased greatly in recent years, mainly due to the emergence of GNU[2]/Linux as a viable alternative operating systems to Microsoft Windows. The GNU project for free software was started in 1984 by Richard Stallman at MIT (Stallman, 2002; GNU, 2004). The definition of free software is given on the GNU website[3]:

> "Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer."
> Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:
> - The freedom to run the program, for any purpose (freedom 0).
> - The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
> - The freedom to redistribute copies so you can help your neighbor (freedom 2).
> - The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.
>
> A program is free software if users have all of these freedoms. Thus, you should be free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission.

A crucial part of Stallman's project was the development of legal licenses to ensure the ongoing freedom of software.

2.2 Open–source software

More recently, the Open Source Initiative (2004) (OSI) has put a more business friendly face on free software. The term was decided upon in 1998[4],

> ... it was time to dump the confrontational attitude that has been associated with "free software" in the past and sell the idea strictly on [...] pragmatic, business-case grounds ...

The precise definition used by the Open Source Initiative (OSI) runs to ten clauses[5]. Raymond (2001, 2000a,b,c) has written an accessible account of the open–source development process. The OSI maintains a list of software licenses which they approve as satisfying the definition, and which are allowed to carry the OSI certification mark[6].

[2] GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced "guh-no" (GNU, 2004)

[3] see `http://www.gnu.org/philosophy/free-sw.html` for the full GNU definition of free software

[4] a history of the Open Source Initiative is available at `http://www.opensource.org/docs/history.php`

[5] see `http://www.opensource.org/docs/definition.php`

[6] see `http://www.opensource.org/docs/certification_mark.php`

2.3 Developing, debugging and code maintenance

Brooks (1995) explains the difficulties faced in large software projects in his book *The Mythical Man Month*. If a project falls behind schedule, adding more developers makes the situation worse as the cost of communication explodes. Boehm (1987) found that 40 to 50 percent of time in most projects is spent on testing and debugging.

There have been many project management solutions suggested, such as the waterfall model (Royce, 1970), the spiral model (Boehm, 1988) and more recently extreme programming (Beck, 1999). However, (Sommerville, 1992) states that, in fact, ongoing maintenance costs are typically two to four times as much as total development costs. The time spent initially coding a solution is a small fraction of the effort involved in a successful software system.

Raymond (2001) discusses how some of these software development problems are overcome by the network effects present in a successful open–source project. Bugs are found quickly when the project has many users, but the communication burden between developers need not explode as you have few core developers and many more users (Raymond, 2001, 2000a). As many users are technically literate the quality of bug reports is increased, and fixes found more easily. The accelerated release cycle brings bugs to light more quickly and allows fixes to be disseminated rapidly.

Dependent libraries of projects are forwardly compatible and code is maintained after particular developers lose interest (Raymond, 2001). This means that it is possible to leave a project temporarily and return later to find that it works with the latest versions of other related projects.

However, Schach et al. (2003) find that the instances of global coupling within the 17 modules of the Linux kernel has been growing exponentially with kernel version number. If this trend continues Linux may become increasingly hard to maintain. Future studies will show whether this effect applies to all open–source development or just Linux.

2.4 Licensing

Careful choice of license is critical to the survival and success of open–source projects. For example, if a license is not deemed open–source by the Open Source Initiative (2004) then the project will have difficulty attracting developers as in the case of Netscape in 1998. Netscape released the source code of their browser under the Netscape Public License, this was not an open–source compatible license, according to the Open Source Initiative (2004), and few developers took up the project. Netscape changed the license to the Mozilla Public License, which was acceptable to the Open Source Initiative (2004), and the project is now successful. Choice of licensing by companies using open–source software is discussed in Bonaccorsi and Rossi (2003b).

We need to define the idea of *Copyleft*,

Copyleft is a general method for making a program free software and requiring all modified and extended versions of the program to be free software as well[7].

The three most popular free licenses are, as described on the GNU website[8]:

GNU General Public License or GNU GPL for short. This is a free software license, and a copyleft license. We recommend it for most software packages.

GNU Lesser General Public License or GNU LGPL for short. This is a free software license, but not a strong copyleft license, because it permits linking with non-free modules. It is compatible with the GNU GPL. We recommend it for special circumstances only.

[7] see `http://www.gnu.org/copyleft/copyleft.html`

[8] see `http://www.gnu.org/licenses/license-list.html`

Modified Berkeley Software Distribution License or modified BSD license for short. This is the original BSD license, modified by removal of the advertising clause. It is a simple, permissive non-copyleft free software license, compatible with the GNU GPL. If you want a simple, permissive non-copyleft free software license, the modified BSD license is a reasonable choice.

A full list of the OSI certified licenses is available from Open Source Initiative (2004)[9].

2.5 Hosting

The code of an open–source project has to be hosted on a server, somewhere on the internet. The GNU project hosts around 2,000 projects at a site called Savannah (2004). The largest such software development website is SourgeForge.net (2004). Of the approximately 50,000 open–source projects hosted at SourgeForge.net (2004):

- 71% are licensed under the GPL.
- 11% under the LGPL.
- 7% under the modified BSD license (SourgeForge.net, 2004).

In comparison there are just 2345 projects that are not deemed open–source by Open Source Initiative (2004). Non open–source projects can have great difficulty attracting and retaining members as illustrated by the Netscape example above (Raymond, 2001; Bonaccorsi and Rossi, 2003b).

Hosting a project at a dedicated site such as SourgeForge.net (2004) makes the project more accessible and it is easier to follow open–source management practices as described in Raymond (2000b). Setting up a project at SourgeForge.net (2004) indicates a commitment to the open–source community and an acceptance of 'open–source values' (Raymond, 2000b). This helps to attract developers and users (Bonaccorsi and Rossi, 2003b).

2.6 Existing projects

There are four main projects for derivatives pricing on the internet:

Financial Numerical Recipes is an introductory collection of algorithms written in C++ (Ødegaard, 2004). Released under the GPL license.

Premia is an implementation of many algorithms in C (Premia, 2004). Owned by the Institut National de Recherche en Informatique et en Automatique (INRIA) and the Centre d'Etude et de Recherche en Mathématiques Informatique et Calcul Scientifique, Laboratory of the Ecole Nationale des Ponts et Chausses (CERMICS, ENPC) in France and released under a restrictive license incompatible with the GPL. Commercial users pay a license fee.

Project Martingale is aimed at mainly academic experimentation with mathematical finance models from a probabilistic point of view. Monte Carlo models in C++ and Java (Martingale, 2004). Released under the GPL license.

QuantLib is a free/open–source library for modelling, trading, and risk management in real-life (QuantLib, 2004). Written in C++ and released under the modified BSD license.

Financial Numerical Recipes is a good introduction to derivatives pricing, and is very accessible. However, it is not aimed at producing a real-life derivatives system, neither is Project Martingale. Both projects lack a clear open–source project management framework, and do not have user mailing lists with a large number of subscribers.

The Premia project has a restrictive license, and is a development project of INRIA and ENPC, so is only useful as a reference implementation, or as a comparison. Contributions are not sought, and if they were, the fact that the license is

[9] see `http://www.opensource.org/licenses/index.php`

not classified as open–source by the Open Source Initiative (2004) would make it difficult to attract developers.

QuantLib has an open–source license, is accessible to users, and the project administrators welcome developers. Currently QuantLib has 14 registered developers and around 400 users subscribed to the mailing list(Ametrano, 2004). It is achieving the critical mass of users needed to ensure the success of the project.

QuantLib is also in use at a number of financial institutions (Ametrano, 2004; Ballabio, 2004). While these institutions use the library they will ensure its quality for the financial products in their portfolios.

3 Motivations of participants

Individuals and companies involved in open–source projects have differing motivations. Ideas from anthropology have been used to describe the processes involved in academic research and open–source software. These ideas are extended to the field of mathematical finance.

3.1 Gift economy

The term *gift economy* was first coined in 1925 by Mauss (1925), when discussing the 'potlatch' gift–giving ceremonies of the American Northwest coast native tribes. Tribes that had sufficient food, and other goods, gave away surpluses to neighbouring tribes. After a time this practice became a custom and the gift–givers came to gain social reputation by the giving of ever more costly gifts. Gifts carry the implicit obligation to reciprocate. This contrasts with the idea of an *exchange economy* where we exchange money for goods and services. Arrow (1972) gives an illuminating discussion of the merits of the two systems in public policies for blood donation.

Hyde (1983) discusses academic research as a gift economy. Scientists were originally wealthy men who had no need to work for a living. Some felt driven to investigate their environment and to further knowledge by publishing their results. Note that we always talk of "giving" an academic paper at a conference. A brilliant scientist is thought to waste her effort if she does not publish her research. Einstein is respected because he gave the world the theory of relativity.

Raymond (2001) applies these ideas to the open–source software movement. In developed economies programmers have spare time to code and have come to contribute code to open–source projects. These contributers often need or desire a particular solution, they code it themselves, and release the code to a project. Coders receive other contributed code in return. Coders gain reputation from the quality and utility of the software they contribute. The most famous open–source developer is no doubt Linus Torvalds the creator of the Linux kernel used in the GNU/Linux operating system.

These informal accounts are now being backed up by empirical research. Lakhani and Wolf (2003) find that creativity is the greatest motivating factor in participation, followed by creating a public good. Hertel et al. (2003) suggests that motivations are similar to other social movements, such as the civil rights movement. In particular Hertel et al. (2003) find that developers believe code should be free, they benefit from code reciprocation within the community, they improve their coding skills and gain a reputation within the community. Both studies find that lack of time prevents participation in open–source projects.

3.2 Academic Research

In the previous section we discussed research as a gift economy. Researchers do not extend the bounds of knowledge unless they disseminate what they have discovered and allow others to verify their results. What is the best method of dissemination?

For many years research has been published in paper journals and knowledge shared verbally at conferences. Knowledge is also disseminated through funding reports.

Today the most efficient way of disseminating research is through the internet. Lawrence (2001) finds that online articles are around three times more likely to be cited than articles of similar quality not available online. Articles are also made easier to find by online citation indexes such as CiteSeer (2004) (a description of how CiteSeer works is given in Lawrence et al. (1999)). The point of research is reduced if results are not published on the internet.

As well as being available research also needs to be peer reviewed. Poor research is as easily disseminated as high quality research. Henneberg (1997) discusses the history and some of the failings of the peer review system.

3.3 Open–science

The Open Science Project (Open Science, 2004) is a group of scientists, mathematicians and engineers who recognise the importance of computer software to scientific development in their field. The project aim is that computer software developed to obtain or process scientific results should be released under an open–source license. This enables the peer review of results by other researchers, and allows results to be verified by other researchers (Kiernan, 1999). The publication of any visualisation techniques and data sets in the public domain has already been recommended by Brown et al. (1995).

Academic papers that are available online are more highly cited (Lawrence, 2001). It is easy to imagine that similar trends will appear between citations and participation in open–science projects. The author has experienced a marked increase in requests for preprints and speaking invitations since contributing to QuantLib.

Clearly purely theoretical papers are still vital to progress, and should be available on the internet. However, mathematical finance as a research area is driven by practical needs so ideas must be implemented in order to be used. It is at this computational stage that open–science ideas should be adopted.

Survey

About two thirds of 29 Mathematical Finance researchers' websites (from the United States and the European Union) informally surveyed have some of their publications available in pdf format on their website (see Table 1). However, only in a single case was any code at all made available. Clearly the benefits of disseminating research over the internet have not yet been recognised by all researchers in the field.

Level of electronic availability of research	Number of researchers	Percentage of researchers
No website	3	10%
Website only	6	21%
Papers online	19	66%
Papers and code	1	3%

Table 1. Dissemination of research in Mathematical Finance

3.4 Motivations for companies

Possible business models for companies using open–source software are discussed in Raymond (2000c). The actual motivations of companies involved in open–source projects have been investigated empirically by Bonaccorsi and Rossi (2003a) in a

survey of 146 Italian companies. They found that the top three motivating reasons, using their exact wording, were:

1. Because Open Source software allows small enterprises to afford innovation.
2. Because contributions and feedback from the Free Software community are very useful to fix bugs and improve our software.
3. Because of the reliability and quality of Open Source software.

The first of these reasons probably does not apply in the financial sector, as banks are generally early adopters of technology and have large budgets for information technology. However, smaller institutions, such as asset management firms, or firms in developing countries, may benefit from the innovation in open–source software.

The second reason is certainly not altruistic. Firms participate in open–source projects so they can get free labour.

The third most popular reason is valid for firms of all sizes. The most mature open–source projects are stable and reliable.

Overall Bonaccorsi and Rossi (2003a) found that firms in general took more from the projects than they contributed. However, they also argue that successful projects are robust to firms being 'free–riders', and not contributing code back to the project.

3.5 Potential disadvantages

Raymond (2000b) notes that there is only 'room' for one project satisfying a particular need. Once a project achieves a critical mass it is difficult for a new project with similar aims to get started. If a developer picks the wrong project his work may not be used, and the project may become inactive.

A project becoming inactive does not necessarily mean that it has failed. The project may be finished, the program does what it was designed to do and can be used without further development.

If a developer is unable to recruit fellow developers to a project she is still free to develop the project herself, as she would have done before thinking about open–source solutions. However, the project will be backed up, and be under proper version control, and someone may become interested in it in the future.

There is a danger that an open–source project deteriorates because of poor code quality. This is probably due to a weak developer community and weak project administration. If the project administration neglects or mishandles the project it may fork (when an existing project splits into two competing projects), causing a great deal of disruption (Raymond, 2000b).

4 QuantLib

QuantLib is a derivatives pricing library implemented in C++ . As the project website (QuantLib, 2004) says,

> The QuantLib project is aimed at providing a comprehensive software framework for quantitative finance. QuantLib is a free/open–source library for modeling, trading, and risk management in real-life.

QuantLib is released under the modified BSD license (QuantLib, 2004). This enables the code to be modified and redistributed by banks or software houses without having to the release code for their proprietary models. A copyleft license (such as the GPL) would only permit the modification to take place within the company, if any code was redistributed all modifications to the source would also have to be released under the copyleft license. As in all open–source projects there is a conflict between the interests of individuals and corporations, and between contributers and free–riders. Individuals may take the code, but are not forced to

contribute code in return. QuantLib is not noticeably suffering from the presence of free–riders.

There are currently no competing projects to QuantLib with similar aims. QuantLib is the only pricing library which aims to be, and is, used in practice. The current project administrators are managing the project well, so the danger of a fork, or other conflict is low. The project administrators are also maintaining a high level of code quality.

One danger is the emergence of a competing project of higher quality, with a restrictive, non-free, license, which could divert effort from QuantLib. However, the emergence of such a project is unlikely at the moment, and as we have seen, a non-free license would discourage developers from contributing.

4.1 Why use QuantLib?

Various user groups have different reasons for using QuantLib. For example, academics would use QuantLib to disseminate research and allow results to be verified. Whereas companies may use QuantLib in practice to run their business. The advantages and disadvantages for such user groups using QuantLib are discussed in the following sections.

MSc or PhD students

QuantLib is an ideal tool for the technically competent student. Student coding is thrown away and never used again, albeit, most of it is not worth keeping. However, if the quality code is contributed to a viable open–source project it takes on a life of its own. The student avoids having to implement basic facilities such as date calculations. As discussed in section 2.3 the student also benefits from the free debugging and code maintenance that is part of the open–source development process (Raymond, 2001, 2000a).

The student also gains an audience for their research and algorithms through the open–science process. As already mentioned, the author has noticed a marked increase in speaking invitations and requests for preprints since contributing to QuantLib.

Another attraction of the QuantLib project is exposure to and use of practical pricing models in C++ . These are essential skills for any student hoping to obtain a quantitative job in a financial institution. In the future QuantLib may well be used in projects for Mathematical Finance Masters degrees. Good C++ skills increase employability, and improved employability is a major factor in the choice of Masters students returning to study after some time working. Improved employability increases the reputation of the institution offering the professional course, therefore attracting the best students in a virtuous circle.

Academics

Academic faculty would also gain from the code debugging and software maintenance benefits. They would be able to build models on an established base and be able to focus on interesting research problems. They will be able to continue computational work after their PhD students have left the university.

Working code will be verified, tested, and adopted more quickly than academic papers alone. Academics researching in computational areas such as mathematical finance can greatly increase the impact of their work if both their papers and software implementations are available on the internet.

As noted in the previous section, QuantLib may also become a useful tool in Masters courses in computational finance, attracting students paying large fees to the university.

Financial institutions

Buy–side firms and sell–side firms have different requirements and interests, so are considered separately. Obviously financial institutions have the option of using commercial software solutions for their needs. Institutions, particularly in developing countries, may find that the fact that QuantLib is free influences their decision to adopt it as their pricing library. Using an open–source solution also gives control back to the institution and avoids vendor lock–in.

Sell–side institutions

Sell–side institutions will not want to release proprietary pricing techniques into the code base. However, they may benefit from using QuantLib as a benchmark.

QuantLib is not a zero sum game. Just because sell–side institutions may not contribute greatly to the project, that does not detract from QuantLib's usefulness to others. Also, just being a user and submitting bug reports is very helpful.

QuantLib is useful for training potential recruits, and enabling those recruits to have some quality coding experience before starting work, thereby saving on training costs. A bank involved in QuantLib also gains access to the most technically competent students, ready filtered, without going through recruitment agents. Developers who already know the library also require less training, which is another saving.

Buy–side institutions

There is more motivation for a buy–side institution to contribute code for pricing algorithms than a sell-side institution. QuantLib is already used in production at a number of institutions (Ametrano, 2004; Ballabio, 2004). Each individual asset management house does not have as many resources to devote to model development as an Investment Bank, so pooling resources makes sense. Asset managers are able to tell whether they are being quoted good prices by sell–side firms.

Software and consultancy firms

Profitable businesses can be run installing, extending, or customising QuantLib for particular financial institutions. Various service based business models for open–source companies are discussed in Raymond (2000c). StatPro (2004) already offers products for fund managers built on top of QuantLib. In the future there may be opportunities for firms offering tools to link QuantLib to other systems, for example via XML standards such as FpML (2004).

Financial regulation

A possible future use for QuantLib is in financial regulation. Capital requirements for portfolios of derivatives could be specified using QuantLib pricing algorithms. A reference portfolio could utilize the QuantLib pricing library. QuantLib may become a method for transfering best practices from developed countries to emerging economies where financial regulators are less experienced.

4.2 Contributing to QuantLib

There are many different levels of contribution to open–source projects, though essentially there is one main distinction, between users and developers.

A successful open–source library needs users testing the library in different environments, and for many uses. A good bug report is a valid contribution to the project in itself.

The process for developing and contributing code is more involved. There are technical prerequisites that have to be met before contributing code. Learning the existing code in the library takes time and effort. A developer must be able to write C++ and configure the necessary libraries and tools, such as SourgeForge.net

(2004) and CVS (2004). To get involved in a project there must also be the desire to contribute quality code, as a developer is unlikely to contribute code that reflects badly on his programming ability. Therefore, though code contributions are vetted by the project manager they are welcomed and are usually worth including.

5 Conclusions

The impact, and scientific utility, of computational research in mathematical finance is much greater if it is implemented in a financial derivatives project such as QuantLib, than if it were not available on the internet. Research available in QuantLib is easier to verify, as anyone can download, read, and run the code. It is also simpler for future researchers to improve code, and test new algorithms against existing implementations in a systematic manner. The Open Science Project aids the transfer of knowledge from universities to commerce.

Companies benefit from a stable code base. They are able to take advantage of the latest developments from academic research and improve the speed of implementations as needed. Financial institutions are not restricted in what they can do with the library, they have control and can add or change features as required. They also avoid vendor lock–in. Such an approach is not possible with closed–source packages.

Acknowledgments

This work is supported by Nomura International plc and the EPSRC. The author would like to thank the QuantLib project leaders Ferdinando Ametrano and Dr. Luigi Ballabio. for their co-operation and useful suggestions. The author would also like to thank Dr. Ruth Stalker Firth for many useful discussions.

References

Ametrano, F. private communication, 2004. URL `http://www.ametrano.net/`.

Arrow, K. J. Gifts and exchanges. *Philosophy and Public Affairs*, **1**(4):343–362, Summer 1972.

Ballabio, L. private communication, 2004.

Beck, K. *Extreme Programming Explained: Embracing Change*. Addison Wesley, 1999.

Boehm, B. W. Improving software productivity. *IEEE Computer*, **20**(9):34–57, 1987.

Boehm, B. W. A spiral model of software development and enhancement. *IEEE Computer*, **21**(2):61–72, 1988.

Bonaccorsi, A. and Rossi, C. Altruistic individuals, selfish firms? The structure of motivation in Open Source software. Technical report, Sant'Anna School of Advanced Studies. Institute for Informatics and Telematics, 2003a. URL `http://opensource.mit.edu/papers/bnaccorsirossimotivationshort.pdf`.

Bonaccorsi, A. and Rossi, C. Licensing schemes in the production and distribution of Open Source software. An empirical investigation. Technical report, Sant'Anna School of Advanced Studies. Institute for Informatics and Telematics, 2003b. URL `http://opensource.mit.edu/papers/bnaccorsirossilicense.pdf`.

Brooks, F. P. *The Mythical Man Month and Other Essays on Software Engineering*. Addison Wesley, 1995.

Brown, J. R., Earnshaw, R., Jern, M., and Vince, J. *Visualization: Using Computer Graphics to Explore Data and Present Information*. John Wiley & Sons, 1995.

CiteSeer. CiteSeer website, 2004. URL `http://citeseer.ist.psu.edu/`.

CVS. Concurrent Versions System website, 2004. URL http://www.gnu.org/software/cvs/.

Davidson, C. The Risk annual software survey 2004. *Risk*, **17**(1), 2004.

FpML. Financial Products Markup Language website, 2004. URL http://www.fpml.org/.

GNU. Gnu website, 2004. URL http://www.gnu.org.

Henneberg, M. Peer review: the holy office of modern science. *naturalSCIENCE*, 1997. URL http://naturalscience.com/ns/articles/01-02/ns_mh.html.

Hertel, G., Niedner, S., and Herrmann, S. Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel. *Research Policy*, **32**:1159–1177, 2003.

Hyde, L. *The Gift: Imagination and the Erotic Life of Property*. Vintage Books, 1983.

Kiernan, V. The 'open source movement' turns its eye to science. *The Chronicle of Higher Education*, November 1999. URL http://chronicle.com/free/v46/i11/11a05101.htm.

Lakhani, K. R. and Wolf, R. G. Why hackers do what they do: Understanding motivation and effort in Free / Open Source software projects. Technical report, MIT Sloan School of Management, 2003. URL http://freesoftware.mit.edu/papers/lakhaniwolf.pdf.

Lawrence, S. Online or invisible? *Nature*, **411**(6837):521, 2001.

Lawrence, S., Giles, C. L., and Bollacker, K. Digital libraries and autonomous citation indexing. *IEEE Computer*, **32**(6):67–71, 1999.

Martingale. Project Martingale website, 2004. URL http://martingale.berlios.de/.

Mathematica. Mathematica website, 2004. URL http://www.wolfram.com/.

Matlab. Matlab website, 2004. URL http://www.mathworks.com/products/matlab/.

Mauss, M. *The Gift*. Routledge Classics, 1925. Translated from *Essai sur le don*.

Ødegaard, B. A. Financial numerical recipes, 2004. URL http://finance.bi.no/~bernt/gcc_prog/.

Open Science. Open Science website, 2004. URL http://www.openscience.org/.

Open Source Initiative. Open Source Initiative website, 2004. URL http://www.opensource.org/.

Premia. PREMIA website, 2004. URL http://cermics.enpc.fr/~premia/.

QuantLib. QuantLib website, 2004. URL http://www.quantlib.org/.

Raymond, E. S. The cathedral & the bazaar, 2000a. URL http://www.catb.org/~esr/writings/cathedral-bazaar/.

Raymond, E. S. Homesteading the Noosphere, 2000b. URL http://www.catb.org/~esr/writings/cathedral-bazaar/.

Raymond, E. S. The magic cauldron, 2000c. URL http://www.catb.org/~esr/writings/cathedral-bazaar/.

Raymond, E. S. *The Cathedral & the Bazaar*. O'Reilly UK, 2001.

Royce, W. W. Managing the development of large software systems: Concepts and techniques. In *Proceedings of WesCon*, August 1970.

Savannah. Savannah.gnu.org website, 2004. URL http://savannah.gnu.org/.

Schach, S. R., Jin, B., Wright, D. R., Heller, G. Z., and Offutt, A. J. Maintainability of the Linux kernel. Technical report, Department of Electrical Engineering and Computer Science, Vanderbilt University, 2003. URL http://www.vuse.vanderbilt.edu/~srs/preprints/linux.longitudinal.preprint.pdf.

Sommerville, I. *Software Engineering*. Addison Wesley, fourth edition, 1992.

SourgeForge.net. SourceForge.net website, 2004. URL http://sourceforge.net/.

Stallman, R. M. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Free Software Foundation, 2002.

StatPro. StatPro website, 2004. URL http://www.statpro.com/.