Open Source software and the Knowledge Economy: virtually free and virtually immeasurable.

Roger Bivand

Norwegian School of Economics and Business Administration

September 1999

Introduction

One of the most striking innovations to emerge with full vigour in the late 1990's, but building on earlier precedent, is Open Source software. While there are good technical reasons for releasing the source code of computer programs to anyone connected to the Internet, or willing to spend little more than the cost of a bottle of beer on a CDROM, the trend profoundly affects the measurement of research and technological development expenditure and investment. Since the software tools are not priced, knowledge, the practice of winning new knowledge, and the style in which new knowledge is won are sharply focused.

The European edge here comes potentially much less from large−scale funded programmes or frameworks, but rather from the honing of ideas in open collaboration and competition between at least moderately altruistic peers. The prime examples are the conception of the world wide web and its fundamental protocols at CERN, by an errant scientist simply interested in sharing research results in an open and seamless way, the transport mechanism for about 80% of all electronic mail `sendmail` − written from California but now with a world cast of thousands of contributors, and the Linux operating system, developed by a Finnish student of undoubted organisational talent, but − perhaps luckily − no training in the commercialisation of innovations, nor in business strategy.

For these innovators, it has been important not to be hindered by institutional barriers and hierarchies. Acceptance and esteem in Open Source communities is grounded in the quality of contributions and interventions made by individuals in a completely flat structure, where neither age, sex, nor any other social criterion play any role. Asking the right question leading to a perverse bug being identified or solved immediately confers great honour on the person making the contribution. In money terms, probably no commercial organisation could afford to employ problem chasers or consultants to solve such problems − they are simply too costly. The entry into the value chain is later, in the customisation, packaging and reselling of software, in software services rather than in the production and sale of software itself. This among other things is why Microsoft sees Linux as a major commercial threat, worse, one that cannot be bought off, because nobody owns it − as Redwood would put it, or everybody owns it, as Linus Torvalds might put it.

In countries and regions trying to catch up or leapfrog in social development, Open Source software and the ethics and logic of its communities offer a free and paradoxically valuable opportunity to join in activities with very low threshold costs,

but large future opportunities for earning the recognition of peers worldwide. It does however require far−sightedness in the administration of higher education and research institutions, in enabling Internet access for schools and universities, and in the demolition of empty academic hierarchical structures not serving the empowerment of people − not just young people − with ideas, commitment, and curiosity.

One clear conclusion is that Open Source software can radically increase the amount of research that can be got from unit research expenditure, as is witnessed by the rise of clusters of tightly networked commodity computers running Linux as a replacement for supercomputers, typically improving cost/performance ratios by an order of magnitude. Another is that Open Source is happening anyway, so that forcing innovation to think mostly about commercialisation, about how to restrict access and impose intellectual property rights, may be counterproductive. Stemming from this is the challenge to commercialise not the knowledge itself, nor its software representation, but rather the provision of services supporting its profitable and/or beneficial use.

Open Source software

It is clearly a paradox that one of the key components of the knowledge economy, the Internet protocol, is available without payment and without intellectual property rights protection as a consequence of US Government policy. Because of the requirement that software developed and delivered as part of research conducted chiefly for the US Department of Defence should be released to others free of charge, since it had been created using public funds, economic forces of quite shocking dimensions have been unleashed. The innovative programmers and computer scientists who created the Internet were bound by their DoD contracts to look away from the commercialisation of their ideas as software products as such.

This in turn generated fascinating cultural convolutions, as technological and commercial forces by turns attracted and repelled each other. The critical mass was however certainly provided by the active stipulation by the US Government that non−classified software resulting from research supported by public funding should be made available at media cost. These costs were initially measured in terms of reels of magnetic tape, and in relation to access to knowledge about the software in question. At the time there were many proprietary computer operating systems and relatively many programming languages, although Algol 60 and Fortran 66 were predominant in academic circles. Very few people were aware of the scattered and often poorly documented small repositories of source code for software.

It is worth recalling how software is made up. From the late 1960’s, data, including programs, were input into computer systems using punched cards, punched paper tape, magnetic tape, and interactively via teletype terminals, similar to telex terminals. Computers were large, complex and very costly, and were operated exclusively by specialists. Some computer programs were available commercially, but would necessarily have to be customised to each site’s specific configuration. Many users would use each computing installation, and unless they were fortunate enough to be able to use programs already available at their site, they would have to write their own in a programming language. In order to run the newly written and possibly very erroneous program, the user would have to compile it from source code in the chosen and possibly rather rare programming language, and then link it with libraries of

functions needed to allow it to be executed at that site with the data to yield results.

One of the results of this was that many early numerical results were of very poor quality, since there was no adequate way of checking to see whether the program was coded in accord with the intentions of the author. Since a computer is a deterministic machine, it will only do what it has been instructed to do, and if the instructions are wrong in some sense, the results of the computation will not correspond with those that should have been produced. The senses in which the instructions may be wrong are very varied, and include inadequate documentation of input and output data formats − i.e. the program performs correctly but the data is entered in the wrong order or format, or the results are output in a misleading way, inadequate documentation of library functions and their arguments, as well as more prosaic errors in the source code of the programs themselves.

One notable and important response to this by university and government scientists in particular in USA, but also in other Western countries, was to collaborate on building up standardised libraries of functions for numerical computing, initially LINPACK for linear algebra among others, and evolving to become BLAS and LAPACK. For engineering, space exploration, medicine, and countless other fields, the availability of well−tried source−code libraries was of vital importance: instead of having to program these functions from scratch, or indeed to rely on proprietary libraries of varying quality, scientists were able to collaborate in refining and honing the quality of the software they were creating in concert. The chief role was played by government laboratories, and by scientists within those organisations anxious to share their knowledge and experience with others on a mutual but non−commercial basis. Access to repositories of source code was typically announced in articles or notes in scientific journals, or in special sessions at scientific conferences. This resource extended by turns using e−mail in the mid 1980's, generic (FTP − file transfer protocol) and specific client−server retrieval mechanisms, before adopting the hypertext transfer protocol (HTTP) used by WWW clients and servers; it may be found at Netlib.

Until the 1970's, moving code or data between computers was accomplished by moving machine−readable physical records, such as punched cards or magnetic tape. Their formats were by and large proprietary, that is they could only be read on machines from the same manufacturer as those used to write them, also some standardisation did occur. Two representations of the alphabet were used: EBCDIC on machines produced by IBM and ASCII on most others. Few machines represented numbers in the same way, differing in the number of bits used in a machine word, the ordering of the bytes in the word, and the technical specifications of floating point numbers. When computers were eventually linked together, the early e−mail systems all had differing addressing formats, permitting the exchange of messages with machines from the same series and manufacturer, but inhibiting exchange with others. Two such networks were DECNET, used on very successful minicomputers sold by Digital, and BITNET (EARN − European Academic Research Network) on machines from IBM.

Growing interaction between users of machines from different firms flourished throughout the 1980's. This extended to collaboration among users to exchange software, not just as source code, but also as executable programs already compiled for a target architecture. The user groups also contributed to the development of discussion groups, supplementing e−mail − essentially a one−to−one medium, with ways of

distributing messages from one sender to many recipients, by deploying list servers, which send e−mail sent to a list address out again to all the members of the list. This was of course of great advantage both for the producers of the specific type of computers, of programs that were run on them, and the users of these. Communities of interest came into being, within which all could potentially contribute, and all could benefit. In time, the discussion lists run over e−mail were supplemented by news networks with a different division of labour between client and server programs. It is very typical that the software used to run e−mail and discussion group systems has been developed on a non−commercial, collaborative basis, and has been distributed at low or no cost.

By the mid 1980's, much of what has shaken the world in terms of information technology in the past five years was in place. Microprocessors had been developed, and the price of computer chips − integrated circuits − was falling steadily. More importantly, the key standard components both for programming, for software engineering, and for the reliable transport of information between computers over local and wide area networks, were all established, although not widely acknowledged by commercial actors. Over and above this, precedences were established for debating and deciding on vital protocols, which are the agreed interfaces permitting interested parties to interact, known as 'Requests For Comments', or RFC. Most of these arose in connection with academic and research communities working on developing the Unix operating system, created at Bell Laboratories, then owned by AT&T, now by Lucent, and first implemented on minicomputers made by Digital but now probably running on all but the most specialised contemporary computer hardware. The language in which Unix was written was 'C', which, like Unix, was created at Bell Laboratories, as indeed was its derivative 'C++', and from all of which AT&T was unable to generate any revenue. The relationships between the three researchers involved in Unix and 'C', Ken Thompson, Brian Kernighan, and Dennis Ritchie, are described by one of them as real but very subtle. Neither Unix, nor C could be restrained within commercial bounds. The same applies fully to the Internet Protocol, IP, and its extension TCP − Transmission Control Protocol, developed to help join together the proprietary networks like BITNET, DECNET, and USENET between machines running Unix, hence the name of Internet.

While many computer manufacturers sold both machines, operating systems, and other software in a single organisation, like IBM, AT&T sold neither computers nor software, but manufactured telephone exchanges and ran the US telephone system. Consequently, it did not have a business model for how to commercialise computer innovations coming from Bell Laboratories, a feature or failing inherited by Lucent. One result was that Bell Laboratories were permitted to issue source code licenses for the whole Unix operating system, including all the numerous compilers, typesetters, and utilities that accompanied it to organisations including universities at low cost. These were not permitted to give copies to others, but developed a culture of reading the source code in order to find out how it worked. A computer centre using an IBM machine with an IBM operating system would have to call IBM service personnel to sort out a problem, while a Unix shop with a source code license could, if the staff were sufficiently proficient, solve the problem themselves, and feed their innovation back into the user community. This interchange exploded in volume and quality when the AT&T Unix source code was supplemented by, and later fully replaced by the University of California Berkeley System Distribution during the 1980's. BSD Unix can be retrieved at no cost and with no guarantees from servers around the world as

source code and compiled on an extraordinarily large range of computers. BSD Unix has been absolutely crucial to the development of TCP/IP, and all of the services now associated with the Internet. Many successful commercial ventures have spun out of BSD Unix, one being Sun, whose slogan as early as the mid 1980's was the remarkably accurate: ''the network is the computer''.

During the same period, programmers began to develop strategies for the writing and exchange of free software. These evolved somewhat differently in Unix environments and in those using microcomputers, chiefly because Unix systems most often included adequate compilers to turn the source code of programs into executables, while personal computers stopped doing so − the earliest PC's and other microcomputers from the 1980's had well−documented interpreters of the BASIC programming language, but these often disappeared from view as time progressed. For this reason, most of the software developed within the Free Software Foundation, led by Richard Stallman, came from a Unix−like position, despite their use of the recursive name GNU − GNU's Not Unix − as their identifier. Apart from generating a very broad range of software, particularly a powerful range of compilers, they have been vital in defining the legal basis of circulating open source software, also known as copyleft. This differentiates between public domain software, provided free, without guarantee, and with no specific requirements placed on the person or organisation receiving it, and software distributed under the GNU Public License, which requires that any products, also commercial, derived from it must be accompanied by the full source code, including the source code to any enhancements. How strictly these rules are applied may vary, but an altruistic programmer publishing his or her work under these rules can be fairly certain that no−one can make money by making that work private, a process known at Microsoft as 'de−commoditization' and applied by them and others to reduce inter−operability and increase the sales of their products.

Independently from these efforts but linked to them in spirit, a scientist working at the CERN Laboratory in Europe developed HTTP, the protocol used for exchanging information on the World Wide Web. Hypertext as such had been available on a number of systems for some time, but this innovation was revolutionary in providing an adequate mechanism using TCP/IP to link documents containing information irrespective of where they are physically stored, provided they are located on some computer with a WWW or FTP server connected to the Internet, anywhere. Further development of the protocol is hosted at MIT and other universities as a consortium directed by Tim Berners−Lee, the CERN scientist who had the original idea in 1989/1990. The consortium is vendor neutral, working with the global community to produce specifications and reference software that is made freely available throughout the world.

Linux is perhaps the defining case of the benefits of open source software. Linux is the name used to describe a range of Unix−like operating system distributions based on a shared kernel − the program organising the work of the computer and making resources available to user programs − written by Linus Torvalds, then a Finnish student. More adequately known as GNU/Linux, the operating system is conservative in building on accumulated industry wisdom derived from decades of experience with Unix itself and other multi−user, multi−tasking operating systems, but revolutionary in allowing thousands of amator and professional software engineers to collaborate in a self−organising, goal−seeking, and evolving project. Linux is a success not particularly because of Linus Torvalds' technical brilliance, although he is certainly talented, but

because of his organisational intuition that the Internet permits lots of enthusiasts to gnaw at a problem at the same time, and that if they can all be encouraged − certainly not persuaded − to work in broadly the same directions, the fuzzy community can achieve both very high productivity and job satisfaction. Once this firestorm is lit, it is very difficult to extinguish.

Following this change in software engineering methodology, companies are having to assess whether they could benefit from opening their source code, and if so, how they should license their code. Copylefted code is not in the public domain, and restrictions are placed on those downloading and using it. The benefits of being part of a community do, not infrequently, outweigh the losses of not being the monopoly seller of a specific piece of software. Beyond Linux, which is predicted to outgrow all other server operating systems over the next four years, and in 1998 had a ‘‘market’’ share of about 17 percent, Apache − with about 60 percent of WWW server installations − is an interesting lesson in the value of community−building for providers of services rather than makers of products. How can it be that opening source code − giving access to the corporate ‘‘crown jewels’’, the product of much research and development expenditure − can be rational?

Open Source software and commercialisation

Bulletin boards were a precursor to the exchange of software from Internet servers, and are still used both among user communities and by commercial software providers interested in making maintenance updates available to owners of their products. Some of these services require registration before downloading may take place, others have open and restricted zones, positioning the open zones in marketing rather than support. Very many companies now provide after−sales support, in particular fairly full access to technical documentation, over Internet media, even though the product may have nothing whatever to do with computing. Broadcasting in this fashion turns out to be a cost−effective way of running customer support services, and by enhancing customer satisfaction, strengthens brand loyalty and the probability of repeat purchase. Adding a novelty or utility edge, such as Volvo’s ‘‘design your own car’’ website, or real−time consignment tracking from companies like UPS, really does seem to make a difference in terms of e−commerce. But Volvo will not let you have the car for free, and UPS wants cash for moving your consignment. It is here that the Open Source challenge makes its point.

As Eric Raymond has shown in three major analyses, most importantly in ‘‘The Magic Cauldron’’, software is generally but wrongly thought of as a product, not a service. Typically, software engineers work within organisations to provide and administer services internally, customising database applications, adapting purchased software and network systems to the specific needs of their employers. Very few are employed in companies that make their living exclusively from selling software, less than ten percent. Assuming that these programmers have equivalent productivity, one could say that corporate internal markets ‘‘buy’’ most of the software written, and that very little of that code would be of any value to a competitor, if the competitor could read it. It looks and feels much more like a service than a product.

If software is a service, not a product, then the commercial opportunities lie in customising it to suit specific needs, unless there really is a secret algorithm embedded

in the code which cannot be emulated in any satisfactory way. This is an issue of some weight in relation to intellectual property rights, which are not well adapted to software, in that software algorithms are only seldom truly original, and even less often the only feasible solutions to a given problem. Even without seeing each others' source code, two programmers might well reach very similar solutions to a given problem; if they did see each others' code, the joint solution would very likely be a better one. Reverse engineering physical products can be costly and time−consuming, but doing the same to software is easy, although one may question the ethics of those doing this commercially. Most often programmers are driven to reverse engineer software with bugs, because the seller cannot or will not help them make it work. An important empirical regularity pointed out by Raymond is the immediate fall in value of software products to zero when the producer goes out of business or terminates the product line. It is as though customers pay for the product as some kind of advance payment for future updates and new versions. If they knew that no new versions would be coming, they would drop the product immediately, even though it still worked as advertised.

Raymond summarises five discriminators pushing towards open source:

- reliability, stability, and scalability are critical;
- correctness of design and implementation cannot readily be verified by other means than independent peer review;
- the software is critical to the user's control of his/her business
- the software establishes or enables a common computing and communications infrastructure;
- key methods (or functional equivalents of them) are part of common engineering knowledge.

Any of these alone may be sufficient to cross the boundary for external economies to begin influencing behaviour in companies, termed network externalities in Raymond's article. If the company can benefit more in terms of its internal and/or external service quality and reliability, then there is now good reason to consider opening source. Cisco experienced this in work done to enhance the standard Unix print server software, to allow print jobs to be executed seamlessly across the worldwide corporate network, to permit administrators to monitor printer malfunctions, and to route print jobs automatically from printers not functioning correctly to neighbouring ones that could do the work. Having done the programming in−house, the programmers responsible became aware that the software would need maintenance, and that they themselves might not be there to do it. They recommended that the software be opened, potentially allowing others to benefit from work done within Cisco − a positive externality for other companies − but also for Cisco to benefit from peer review feedback as improvements to the software, and from future−proofing, in that the future quality of the software becomes a shared interest for all those who have started using it.

While externalities are not a key feature of MBA−style business economics courses, they are needed to understand these kinds of mutual benefits. They are also at the core, together with increasing returns to scale, of most analyses of innovation clusters, such as Silicon Valley. Increasing returns to scale also seem to play a role in the Open Source solution, in that opening source in settings with constant returns to scale would not benefit anyone. If however increasing the number of programmers working on creating, peer−reviewing, and debugging software yields more than proportional gains

in productivity, which it arguably does, then increasing returns to scale are part of the story. This in turn means that software projects of interest to few hackers will benefit less than those of interest to many. Another consequence for projects with limited local interest is that increasing returns to scale may be achieved globally, even when the number of hours that could be assigned to a given project in each IT department separately is very small. Pooling resources means that each of the collaborating programmers can provide better services internally to his or her organisation than would otherwise be the case.

So how can Open Source be seen as a commercial proposition? In macro−economic terms, it is included in the national accounts entries of organisations using services internally or trading in services. But can it be commodified? Raymond gives a number of routes to revenue and profit from Open Source, chiefly through the linking of Open Source and other products, training, customisation, and sales of media and support materials. Many of these build on a mature understanding of customer relations, that it is better in the medium and longer term to develop trustful relationships with customers, than to force them to accept shoddy products and limiting solutions. The linking of Open Source and other products concerns the sale of many kinds of hardware using digital technology, not just computers as such. Many kinds of products can be updated, and their effective working lives extended, by adding value in this way; it is however the case that opening driver software gives the customer confidence in continuing support for the product, if not necessarily from the producer, then at least within the broader user community.

With regard to the remaining components of revenue from Open Source, it cannot have escaped anyone’s attention that there is a lot of money being spent on computer books, on training courses, and on customisation. While software firms will forego rent on proprietary programs by opening them, they will find that opening is also an opening to the customer, and to free access to large amounts of information about customer needs, desires and problems, exactly what is needed to personalise customer relationships. In fact, many business customers are not even very interested in buying hardware or software any more; companies stopped seeing the point in building their own electricity generating plant as soon as the generating industry had standardised and stabilised. In computing industry terms, the equivalent of a wall power point is a wall network point, permitting arbitrary equipment to be attached and providing a so called web−tone, by analogy with telephony and the dialling tone. In the same way that banks ‘‘know’’ a lot about corporate treasury dispositions, but are bound to confidentiality, mass compute service providers will host corporate databases. The same or other service providers will host the ‘‘middleware’’ linking these to the specific needs of the customers’ individual employees. Much of the middleware is already Open Source − WWW and electronic mail servers, transport and addressing mechanisms, plug−ins using Perl or Java to access particular media or services − and more satisfies Raymond’s discriminators.

Open Source software, research and higher education

While the links between the knowledge economy and the commercial aspects of Open Source software are evident, if only emerging, there are very real challenges to the management of research and higher education in policy terms that need to be addressed. Most research and higher education organisations are currently being rationalised and

subjected to the styles of management practices introduced in commercial corporations years and even decades ago. In particular, budget discipline is a favoured tool in attempting to direct organisational units in directions seen as being appropriate. Given that these organisations clearly face a ‘‘missing market’’, in that neither potential students nor grant−giving bodies are analogues of customers in a fast−food restaurant, those responsible for management have a measurement problem. Costs can be measured on a historical basis, but depreciation − say of investment in immaterial assests such as time spent reading scientific journals − is hard to structure. Revenue is equally hard to assess realistically in many cases, given the vagaries of grant processing, and the lumpiness of grants themselves. Organisational units are asked and expected to behave market−fashion often without market information, and often without the understanding contemporary managers have for the cost/revenue relationship over time.

In this situation, it is worth reflecting that Bell Laboratories have provided posterity with a methodology of software engineering − simplicity, clarity, generality, the Unix operating system, and the ‘C’ programming language, all without any coherence between commercial policy and scientific excellence. The same remark applies to the development of Internet infrastructure on US Department of Defense research contracts, and to the innovation of the World Wide Web at CERN, although here there was no research project, just a scalable solution. In addition to these suggestive examples, managers of these kinds of organisations need to take account of the importance of independent peer review in quality assurance, since most often there is no adequate market that will walk away from patently bad products or services, nor any other formal methods for proving that research results are ‘correct’. To attempt to impose organisational intellectual property rights on research results derived from the work of individuals or groups has already been condemned as an abrogation of academic freedom, but in the light of the role of externalities and increasing economies of scale above, such policy choices can be directly harmful to the results of the organisation.

Universities and research institutions appear to ‘compete’ in grant processes, and thereby seem to have an interest in locking potential competitors out, by securing privileged access to knowledge. While such advantage may be quite real in the case of laboratory skills and quality − the institution does deliver services of higher quality, or when the institution has secured the services of high−flying academics, this model is not directly transferable to software. Given the steadily increasing importance of software in teaching and research, it seems clear that care is needed in constructing management tools for activities which may produce or modify software. It does make sense for institutions to develop expertise in customisation, in training, and in publishing materials of benefit to software users on a for−profit basis, as sketched above. It does not in general, however, make sense to mandate source closure in research programs or projects, in the same way that mandating openness might be mistaken. The question as to whether software deliverables, or software developed in the process of creating deliverables should be opened is one that is relevant in all grant processes. It is also highly relevant in evaluation routines associated with program and project execution.

In both these cases, the grant−making body should consider at least two factors: the importance of Open Source for enhanced efficiency in providing the software needed in a project, and the importance of peer−review in the scientific process generally. An

example of relevance to the first consideration is the Avalon project, in which physicists at Los Alamos National Laboratory were able to provide themselves with a supercomputer built from standard personal computers, with Alpha rather than Pentium processors, linked together and running Linux and other Open Source software. It was ranked in June 1999 as the 160th most powerful supercomputer in the world for 10 percent of the cost of the commercial equivalent. With reference to the second consideration, it is sobering to read reports of work on the reliability of statistical software, showing that some research results may not be quite what they seem (see two articles by B. D. McCullough in *American Statistician* , 1998, pp. 358−366, 1999, pp. 149− 159).

It can thus be argued that the management of the boundary between what the institution ‘‘owns’’, what can sensibly be commercialised on a for−profit basis, and research productivity and efficiency deserves attention. Otherwise, naive and rather outdated management practises can endanger research quality and productivity with regard to software innovation and incremental improvement by seeing products where one should see services. There are clearly cases in which source code should not be opened, although when the public purse has funded the research involves, the number of real cases will in practice be very few, even for projects with very small communities of interest. It is of importance for the enabling, for the empowering of actors in the knowledge economy, that unnecessary barriers to the diffusion of knowledge be removed, and that new ones not be permitted to emerge. As a corollary, researchers should perhaps be given incentives in career terms to contribute to the pool of knowledge by opening source code, and by contributing to the improvement of software in their domain of science, in the same way that publications are rewarded.

Conclusions

Open Source software is a challenge to all organisations providing or consuming information and communication technology services, organisations that are at the heart of the emerging knowledge economy. It is a challenge not only because of the apparent difference in culture between anarchic programmers and buttoned−down business people − a difference that is more in the eye of the beholder than in reality − but more importantly because the knowledge economy is primarily a service economy. The winners in the knowledge economy will be organisations who serve best through knowledge, not those who try to earn rent from owning knowledge. If the ‘knowledge as product’ paradigm, with strong emphasis on policing intellectual property rights, does come to dominate with reference to software, then it is arguable on welfare grounds that this should not be permitted to happen. Happily, market forces themselves in the commercial sphere are moving strongly towards a service provision understanding of the knowledge economy, as evidenced by the interest shown in Open Source as a software development strategy by leading players. There remains a lurking danger that research managers and administrators, because of the ‘missing market’ problem, will try to make research institutions treat knowledge as a product, not a service, something that in any case runs across the grain of academic tradition.

Hyperlink listing:

The Practice of Programming website	http://tpop.awl.com
Free Software Foundation	http://www.fsf.org/
GNU General Public License	http://www.fsf.org/copyleft/copyleft.html
Avalon supercomputer	http://cnls.lanl.gov/avalon/
Linux resources	http://www.linux.org/
Netlib	http://www.netlib.org/
Open Source Software	http://www.opensource.org/
Internal Microsoft report on Open Source software	http://www.opensource.org/halloween/
Dennis Ritchie's comments on his colleagues	http://reality.sgi.com/relph/humor/krt.html
The Cathedral and the Bazaar	http://www.tuxedo.org/~esr/writings/cathedral−bazaar
Homesteading on the Noosphere	http://www.tuxedo.org/~esr/writings/homesteading
The Magic Cauldron	http://www.tuxedo.org/~esr/writings/magic−cauldron
World Wide Web Consortium	http://www.w3.org/