Rishab Aiyer GHOSH, University of Maastricht
UNCTAD-ICTSD Dialogue on *Moving the pro-development IP agenda forward: Preserving Public Goods in health, education and learning*
Bellagio, 29 November – 3 December 2004

THE OPPORTUNITIES OF FREE/LIBRE/OPEN SOURCE SOFTWARE FOR DEVELOPING COUNTRIES

DRAFT

1. Executive summary

This Background Paper for *The Fourth Bellagio dialogue on development and intellectual property: Moving the pro-development IP agenda forward: Preserving Public Goods in health, education and learning* on the topic of open source aims to provide a concise but well grounded understanding of the opportunities for developing countries.

In recent years, free/libre/open source software (FLOSS) has developed as a novel form of collaborative production. Since its origin in collaboration between individual volunteers, it has seen tremendous success, both in terms of the commercial and technical strengths of the produced software itself, but also as a model of organisation and development. In particular, it has received much attention from developing countries for two reasons: the software itself may be cheaper to use and support than proprietary software applications; and free software may be a novel way for developing countries to leap-frog into the global information economy.

The second point builds upon the modifiable nature of free software, which allows it to be adapted to local needs forming the basis for viable local businesses. Free software's modifiable nature is crucial to one of its main advantages as described by developers themselves – as a learning environment. The ability to experiment with software development at no cost and draw upon a huge, encouraging global community builds a massive apprentice-teacher network that also serves as a novel form of technology transfer.

This background paper concludes with a brief discussion on the mainly political obstacles to the successful adoption of free software in developing countries, and the possibilities for replicating this model in other domains.

2. What is free software / open source?

Free software originated as a formulated concept in 1984 when Richard Stallman used the term for the announcement of the GNU project[1], and later set up the Free Software Foundation. "Free" refers to the freedoms users of software should have: to run the program, for any purpose; to study how the program works, and adapt it to your needs; to redistribute copies; the freedom to improve the program, and release your improvements to the public, so that the whole community benefits.[2]

Software is written as human-readable *source code* and translated into machine-readable *object* or *binary* code in order to run. As humans can only study, adapt and improve software if they have access to the source code, the freedoms of free software require that this source code be available whenever software is distributed, without restriction.

As can be seen from this definition, "free" refers to freedom, not to price. Indeed, the free software definition – and its most common legal embodiment, the General Public License (GPL) – does not prevent charging for the software you distribute. You can charge for your software, but you must provide recipients with the source code and the four freedoms, including the right to redistribute the software to others without restriction.

Continuing misunderstandings result from an unfortunate ambiguity in the English language over the meaning of the word "free", which eventually led to the development in 1998 of Open Source as a "marketing term":[3]

- The case that needs to be made isn't about the concept of open source, but the name. Why not call it, as we used to, free software?…
- The real reason for the re-labeling is a marketing one. We're trying to pitch our concept to the corporate world now… the term "free software" has been misunderstood by business persons, who mistake the desire to share with anti-commercialism — or worse, theft.

The Open Source Initiative published a 10-part definition for open source, which turns out to be equivalent to the four freedoms[4], and in practical terms all software licences approved by the FSF as free software are also approved by the OSI as open source.[5] It is quite apparent that the Anglophone press, business and government communities prefer the term open source, while developers themselves prefer "free software".[6] Many countries, in particular those using Latin languages such as French where a clear distinction is made between *libre* (freedom) and *gratuit* (zero-price) never faced this linguistic problem and have continued to favour the terms equivalent to "free software": e.g. logiciel libre (French), software libre (Spanish) or software libero (Italian). The use of "open source" as an English

[1] http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html
[2] http://www.fsf.org/philosophy/free-sw.html
[3] http://www.opensource.org/advocacy/case_for_hackers.php
[4] http://www.opensource.org/docs/definition.php
[5] http://www.fsf.org/licenses/license-list.html and http://www.opensource.org/licenses/index.php
[6] FLOSS developer survey of over 2700 developers: http://flossproject.org/floss1/stats_1.html

loan word is increasing, even in official French Navy tender documents![7] The Free/Libre/Open Source Software EU project in 2001[8] coined a new catchall term and FLOSS has since then become very popular especially in developing countries that wish to retain the emphasis on the value of freedom without the ambiguity of free-of-price.

It should be noted that the *software* and licence is the same regardless of whether it's called free, libre or open source. So is, by-and-large, the method of software development. But as political movements they differ: free/libre software movements emphasise freedom and ethical values ("sharing knowledge"), while open source as a political movement is much weaker, preferring to emphasise pragmatism or avoid political discussion altogether.

3. How and why does the collaborative development model work?

Free software provides a context in which collaboratively creating knowledge, something inherently human, comes to be seen as a novelty. Novelty excites, and the headlines grabbed by the open source and free software movement have resulted in a renewed public interest in collaborative creation as a whole. Businesses are looking at collaboration, not just in free software, where the likes of IBM, Oracle and Sun – otherwise holders of vast intellectual property domains – have investment plans of billions of dollars. The pharmaceutical and biotechnology industry has organised consortiums for genetic information, where individual discoveries are shared in a common pool rather than – as used to be the norm – secretly squirreled away in in-house labs for further commercial exploitation. Commerce matters, for free software has shown that collaboration can be profitable simply by virtue of leading to greater human creativity.

This novelty stems partly from the tools of the free software movement which have been not just technical but legal – the General Public Licence, or GPL, the most common free software copyright licence, requires redistribution of copyrighted free software to be on the same "share-and-share-alike" terms as the original software. This twists the fences of intellectual property around so that rather than enclosing private spaces out of the commons, the GPL protects the commons and prevents private appropriation.

Novelty is good for exciting interest, and it would be positive indeed if this interest leads to a greater awareness of the importance of collaboration in creativity in all areas of activity. But the novelty itself is misplaced. Humans have been collaboratively creating and owning knowledge for as far as we've been able to communicate, and such knowledge forms the basis of our ability to function as societies today in more or less every field of endeavour one cares to examine[9].

So how, in practice, does free software get developed? I focus only on the "collaborative/community" development aspect, although much free software is now developed by companies. Free software is originally developed by some individual or well organised group, who release their kernel into the community – what I have described

[7] http://ted.publications.eu.int/udl?REQUEST=Seek-Deliver&LANGUAGE=FR&DOCID=167413-2004

[8] www.flossproject.org

[9] See, e.g., Ghosh, R.A., 2005. Editor, *Collaboration, Ownership and the Digital Economy.* Cambridge: MIT Press

elsewhere as the "cooking-pot"[10] – under a free software licence. The licence allows others to contribute to the software. They use it, adapt it to their own needs, and publish their changes back into the "cooking-pot". Depending on the level of governance the originators build around the software, there may or may not be a semi-formal protocol around accepting modifications to an "official" version of the software. Some free software projects have an official versions governed by a foundation (e.g. most GNU projects by the FSF, Apache by the Apache Software Foundation, the Linux kernel by Linus Torvalds acting as "benevolent dictator").

But these governance structures depend on their voluntary acceptance by the community they serve, as contributors are free to distribute their modified unofficial versions of software. Contributors are motivated by many things, from the need to improve the software for one's own use, to generating recognition and reputation, to learning and developing technical and other skills. While most contributions are very small, it is the small contributions, typically from users rather than developers, which are essential. As Linus Torvalds told me in the first interview on the economics of Linux, "the large user-base has actually been a larger bonus than the developer base"[11]. The fact that individual contributions are relatively small in comparison to the ability to access the joint output of thousands is what keeps the "cooking-pot" going, constantly producing more as a result of voluntary informal collaboration.

4. How can free software benefit developing countries?

4.1. Lowering costs and improving access

The advantages of free software are largely in the freedoms it provides, rather than its cost. Still, it is usually the case that organisations are attracted to free software by its low cost and then, with greater familiarity, appreciate its other benefits.

Free software is not necessarily free of charge. You can sell free software, but since you can't stop your buyers from reselling or giving it away at no charge, simple economics shows that any free software package is likely to be available, somewhere, for a price of zero. This is the price for the software itself; of course there is a cost to the CDs on which it is distributed, printed manuals or if you get everything off the Internet, your communication costs. These costs are all very close to zero. Moreover, they apply only once. After you have one copy of a free software package you can redistribute it infinitely, thus the marginal cost is zero.

Free software opponents are quick to point out, correctly, that the cost of software licence fees are only one component of the total cost of using a given software package: the so-called *Total Cost of Ownership*, or TCO. There are installation costs; training costs; support and maintenance costs; costs of other hardware or software required for using the particular software package. Most studies of TCO (usually conducted in Western countries, in

[10] Ghosh, R.A., 1998. "Cooking pot markets: an economic model for the trade in free goods and services on the Internet", *First Monday*, volume 3, number 3 (March), http://www.firstmonday.org/issues/issue3_3/ghosh/

[11] Ghosh, R.A. 1998. "Interview with Linus Torvalds: What motivates free software developers?", *First Monday*, volume 3, number 3 (March), http://www.firstmonday.org/issues/issue3_3/torvalds/

the private sector) show that these associated costs are huge, and that licence fees account for 5-10% of TCO.

TCO studies are rarely taken seriously by IT managers any more, since their results seem to depend on who sponsors them[12] as well as the details of the specific organisations subject to study. The concept of TCO is important, but the actual total cost really differs from organisation to organisation.

What is apparent though is that the largest parts of TCO in Western countries are labour costs: training, support, maintenance, and installation. Clearly, when labour costs are high, labour-intensive components of the total cost represent a high share of the total cost, making the licence fee itself less crucial. As a result, since the only certain saving with open source software is the (zero) licence fee, the cost advantages are not necessarily always clear. In contrast, when labour costs are low, the share of the licence fee in the total cost of ownership is much more significant, even prohibitively so. This is the case in public sector organisations, economically disadvantaged regions, small businesses and most dramatically, developing countries.

This relationship is neatly demonstrated by comparing licence fees with a country's GDP per capita (i.e. the average individual income)[13]. As is quickly apparent, in developing countries, even after software price discounts, the price tag for proprietary software is enormous in purchasing power terms. The price of a typical, basic proprietary toolset required for any ICT infrastructure, Windows XP together with Office XP, is US$560 in the U.S. This is over 2.5 months of GDP/capita in South Africa and over 16 months of GDP/capita in Vietnam. This is the equivalent of charging a single-user licence fee in the U.S. of US$7,541 and US$48,011 respectively, which is clearly unaffordable. Moreover, no likely discount would significantly reduce this cost, and in any case the simple fact that a single vendor controls any single proprietary software application means that there can never be a guarantee that any discount offered is intended to be sustained for the long term, rather than as a temporary measure used to tempt consumers into a lock-in situation at which point in time the discount can be reduced.

This simple calculation is presented in the table below for selected countries. The table also includes the piracy figures published by the Business Software Alliance (BSA). It should be noted that there is a correlation between the piracy rate and the effective software licence fee — the more expensive software is, the higher the piracy rate. This is common sense, but does not seem to be reflected in the BSA estimates of the "losses" to the software industry based on piracy, which assume that all the estimated unlicensed copies of software in a country should (or could) be replaced with paid licensed copies. Ironically, the logical conclusion of the increasingly stringent international campaign for strong enforcement of copyright is the reduction of piracy rates not through the take-up of licensed, proprietary software, but through the use of free software. Anecdotal evidence shows this is the case in Argentina, Peru and other countries especially in Latin America, where a campaign for strong copyright enforcement has coincided with poor economic conditions.

[12] Some studies showing that Windows is cheaper than Linux have been sponsored by Microsoft

[13] Ghosh, R.A., "Licence fees and GDP per capita: The case for open source in developing countries", *First Monday*, volume 8, number 12 (December), http://firstmonday.org/issues/issue8_12/ghosh/

Country	GDP/cap	PCs ('000s)	Piracy	WinXP Cost*	Cost in GDP months
Brazil	2915	10835	56%	6777	2.3
China	911	24222	92%	21678	7.4
European Union	19926	87764	n.a.	991	0.3
Ghana	269	66	0%	73442	25.0
Indonesia	695	2298	88%	28412	9.7
Nigeria	319	889	71%	62014	21.1
Romania	1728	801	75%	11433	3.9
United States	35277	178326	25%	560	0.2

Note: US$560 price from amazon.com in June 2003. GDP Data from World Bank World Development Indicators Database, 2001; Piracy data from Business Software Alliance. * Windows + Office XP equivalent US$ cost calculation = $560 * (U.S. GDP per capita/Country GDP per capita).

4.2. Adaptation to local needs

Once free software is selected for its low cost, other benefits become quickly apparent. One of them is the ability to adapt software for local needs. Proprietary software companies are usually global, concentrated in a few parts of the world. This is the nature of the software market, which, thanks to network effects and proprietary standards tends towards natural monopolies. These global companies make investments on the basis of global returns, and may not pay sufficient attention to local needs.

The tendency of proprietary vendors to ignore local needs is especially marked in developing regions[14]. As proprietary vendors are motivated by global profit-maximisation strategies they often don't care about local issues and user needs – unless they matter in "a global context". For example, they may pay little attention to the needs of Xhosa speakers in South Africa, who form a relatively small market in global terms. And since their software is proprietary, no local user or local business is in a position to add such support.

Many FLOSS developers too have absolutely no interest in software usability for Xhosa speakers. However, unlike their proprietary counterparts, FLOSS developers allow and encourage those with locally relevant motives to adapt their software. So local users – and, importantly for developing local ICT economies – small local businesses are entirely capable of providing services and adapting the software to local needs. In the case of Xhosa and several other southern African languages this is being done for Linux by the South Africa–based project "translate.org.za".

Local adaptation can go well beyond language interfaces, however. In the well-known case of Extremadura, a poor region of Spain[15], a local version of GNU/Linux was developed, called GNU/LinEx. Uniquely, all the usual icons for common applications were replaced by images more familiar to locals (and easier to pronounce) than "Mozilla" and "GIMP" and "Browser". Instead, there were images of local painters and writers (to launch

[14] Although, famously, after a threatened boycott of Microsoft products the company was forced to develop "expensive" support for Nynorsk (New Norwegian) which was already supported by free software applications in 2002; see http://dot.kde.org/972035764/ and http://news.bbc.co.uk/2/hi/technology/2615363.stm

[15] http://www.linex.org/linex2/linex/ingles/index_ing.html

the paint and word-processing applications) and a bird known in local legend to travel far and wide to search (web browser). As a result, this free software environment has been used to train over 70 000 housewives, unemployed and retired persons the use of computers for the first time, making the interface more approachable than that of the standard Windows (or the standard Mac or GNU/Linux).

4.3. Locally retaining a higher share of the value added

Such local adaptation supports the creation of new, local businesses, which are able to provide commercial support for and build upon free software thanks to its low entry barriers, in a way that would not be possible with proprietary software. This effect is heightened by any public support of the open source software sector. For example, the take-up by the Extremadura Region in Spain of open source through its support for the LinEx project has led to an economic regeneration in a relatively poor region of the European Union (receiving, in April 2004, the award of the European Regional Innovation Award). This has not just allowed the implementation of activities for a lower price, but activities especially in education and training which were simply not possible with proprietary software; it has also led to the growth of a number of small businesses to provide commercial support, since with free software there is no need to approach one sole vendor for support — approaching local entrepreneurs is possible and an obvious choice.

Of course, proprietary software also supports local businesses (excluding businesses who are *users*, who exist regardless of the type of software). What are the types of businesses that can be based upon proprietary software? Building new products and services above the platform is one, equally applicable to free software – 100% of this value is local. Sales commissions are another, rarely possible with free software, and of relatively low value. While 100% of the commissions are locally retained, they represent a small proportion of the total value added and every dollar of sales commission represents several dollars of imports. Finally, support, integration and customisation – where with proprietary software the local value added is limited by the proprietor's control of the software. "Deep" high-value support requires "deep" high-value access to the software, which only the proprietor has.

With free software, the "deep" support that can be provided by "deep" access to the code available to all local businesses can generate enormous value, all of which is retained locally. No royalties or licences have to be paid.

For local businesses producing their own software, rather than supporting other software too free software is often a better value proposition. Free software allows providers to reuse code rather than build from scratch, and to reuse a huge base of code written by others. Re-using (and modifying) allows the creation of much better end-user solutions for the same effort than writing from scratch, which local businesses are forced to do if they choose to develop proprietary software. Put together, this provides better value for money for customers (who benefit from software representing a large base of cumulative development) and better profit margins for local service providers (who can focus on adding new features faster rather than replicating basic ones, allowing them to charge more for less work).

4.4. Developing local skills

The Free/Libre/Open Source Software (FLOSS) study in 2002[16], a comprehensive study of developers and users, showed that the most important reason for developers to participate in open source communities was to learn new skills — "for free". These skills are valuable, help developers get jobs and can help create and sustain small businesses. The skills referred to here are not those required to use free software, but those learnt from participation in free software communities. Such skills include programming (of course), but also skills rarely taught in formal computer science courses, such as copyright law and licenses (a major topic of discussion in many free software projects). Teamwork and team management are also learnt – after all, the team management is required to coordinate the smooth collaboration of 1500-plus people who rarely see each other is more intensive and far subtler than what is required to coordinate smaller teams employed in a single software company.

Some findings from the FLOSS survey are appropriate here: 78% of developers join the free software community "to learn and develop new skills"; 67% continue their participation to "share knowledge and skills". These learnt skills have economic value to developers – 30% participate in the free software community to "improve … job opportunities"; 30% derive income directly from this participation and a further 18% derive indirect income – such as getting a job unrelated to free software thanks to their previous or current participation in free software developer communities. Being a Linux kernel developer proves a certain level of skills in many ways far better than having a computer science degree from MIT, and employers benefit from such informally learnt skills. 36% of organisations polled in the FLOSS User Survey "totally" or "somewhat" agree that employees can work on free software projects on employer time. These are not necessarily IT companies – 16% of low IT-intensity companies (e.g. retail, automobiles, tourism, construction) "totally" agreed with this point.

4.4.1 Informal apprenticeships – employers benefit, but don't pay the cost

FLOSS communities are like informal apprenticeships – but the apprentice/students and master/teachers contribute their own time "for free", without any monetary compensation for the training process. There is certainly a social cost, but it is borne voluntarily by the participants themselves and not paid for directly by those who benefit (such as current or future employers, or society at large). Everyone can benefit equally from this training – any employer can hire someone informally "trained" through participation in the free software developer community. However, not everyone invests equally in it. As many "teachers" may have been formally trained at university or at work, which is explicitly paid for, explicit costs are being borne for some proportion of community participants who have been formally trained.

In the larger perspective, this training system where all parts of society benefit from the products of the system, but only some explicitly pay for it, represents a subsidy – or technology transfer – from those who pay for formal training to those who do not (or cannot). Within countries, this represents a technology transfer from big companies who often formally pay for training to small and medium-sized enterprises (SMEs), who can less afford formal training expenses. Globally, this represents a technology transfer from the usually richer economies who can afford formal training, to the usually poorer ones who cannot.

[16] Ghosh et al, 2002. FLOSS Final report, part IV. http://flossproject.org/report/

There are also sectoral benefits, especially within poorer economies. Poor countries may have formal computer training during computer science degree courses, but perhaps not in other subjects, such as biology. Anecdotal evidence (in the case of biology, from India) suggests that the use of free software platforms during formal training in non-computer subjects may encourage informal learning of computer skills by students, thereby increasing their understanding of their own course subject (by better being able to conduct biology experiments through more sophisticated computer analysis). FLOSS usage can thus provide students of other subjects to informally learn computer skills, programming skills and enhance their competence in their formal training

4.4.2 But do we all want to program?

The simple answer is "of course not". But how will we know, unless we can try? This question is usually understood within the framework of proprietary software, where creativity requires climbing a very high barrier of investment – time and money – in order to become, say, a professional C programmer. However, HTML is in fact a programming language, and involves inserting "tags" such as "<b>" in the middle of text (this "command" would make the nearby text bold). The World Wide Web took off in the mid 1990s thanks to the hundreds of thousands of people – artists, scientists, children, poets, but not necessarily computer programmers – who created their own individual web pages. They could do this because the structure of the web was open, so people could learn to write their own sites just by copying and changing other sites. In other words, laypersons with few computer "programming" skills could write their own web pages – HTML programs – by looking at someone else's web page (HTML program!) and clicking "View Source" on their browser. It was truly open source, in that the HTML source code for every web page is visible to everyone who sees the finished web page.

"Programming" covers a very broad range of skills from HTML to C. In a free software environment, users are allowed entry to the world of producing at any degree with little investment in time or effort. Of course, your level of creativity and your ability to create depends on the degree of effort you invest. But there is no bar over which you have to leap in order to be able to produce at all. In a proprietary environment, you have to decide to be a programmer, then buy development software, then spend lots of time and effort – all of which is contains risk and forms an entry barrier. With free software, you can tinker. You don't need to buy tools. And you can use them to the extent you choose. Learning skills in this environment, you risk losing only your time and effort

However, since the barrier to entry is low (HTML is the best example!) you can control the degree of your investment – paddle at the shallow end of the pool or dive in deeper. In proprietary environments, the dividing line between user and developer is much sharper – the pool has only a deep end, you have to dive in or stay out altogether. Of course an environment doesn't have to be entirely proprietary or free – many people learnt HTML on a proprietary Windows machine. But the library of web pages available in HTML itself was all "free software". The extent to which skills were easy to develop with a low barrier to entry was limited by the extent to which the environment was open: the barrier-free ability to create stops soon after HTML. In a fully free, open source environment, there is no such barrier; you can start with HTML and end up changing the Linux kernel, the core of the computer's operating system.

The FLOSS study showed that developers who provided "learning new skills" as their reason for joining the community often show "sharing skills" as an equally or more important reason for continuing their community participation. This is correlated with the duration of their participation in the community, and thus represents a shift from "apprentice" to "mentor"

roles. In a reflection of the development process for individuals, countries that profit most from open source are those that contribute back to the community and knowledge base, and there is a built-in incentive (and low barriers) for a shift from being a recipient of skills to being a skills donor. So the process of "subsidy" is very dynamic, and is likely to lead not to a dependency relationship but rather to an equal relationship based on, among other things, local specializations for locally relevant issues.

Should a society encourage passive users of "black-box" software or active participants in the global ICT community? Being active requires being able to create – and choose with the least barriers the level of creativity. Clearly, the lower the entry barrier for creativity, the higher the potential creativity that will occur. Developing countries need to avoid being locked out of skills and competencies. Skills development requires access to the ability to create – so while you don't have to be a programmer, but you should have the choice.

5. What problems will developing countries face in adoption?

Adoption of new technologies can often be easier when older technologies are not present, removing the need to account for sunk costs in old systems, retrain and "unlearn" skills applicable to old systems, and so on. So the adoption of free software in developing countries should not pose many practical problems that are not faced in any case in the adoption of information technology *per se* – hardware, training, basic infrastructure such as electricity and telecoms.

The main practical problems faced in rich countries have related to retraining end-users to get used to new interfaces. For instance, users have faced initial difficulties adapting to OpenOffice Writer after years of using Microsoft Word. However, this is not really related to quality, feature availability[17] or indeed free software at all. Indeed, people complained when they first moved from Wordperfect to MS Word, and still complain when they move from Apple Macs to Windows or vice versa. This major hurdle in adopting free software is hurdle of changing habits, and developing countries, where most people haven't used a computer at all, can simply bypass it altogether. This was clearly demonstrated by the Extremadura example referred to above, where people with no previous computer experience easily learnt to work with a pure free software system, GNU/Linux, OpenOffice etc.

There is also something patronising in the suspicion of free software and insistence on the "latest" technology, whatever that is. While the "latest" is usually free software, these days, it should be noted that there is nothing wrong with "old" technology per se as long as it works, other than the injury it causes to egos. In the US, 15% of homes had PCs in 1989 and 22.8% had PCs in 1993[18]. In 1989, these were typically PC XTs or less with no hard disks and 640kb RAM, monochrome monitors and command-line interfaces. Today, in Africa or South Asia, under 5% of homes have PCs[19]. It is patronising to suggest that what American homes were capable of using in 1989 or 1993 is too complicated or not "modern" enough for people in poor countries to use today. Indeed, one of the remarkable advantages of GNU/Linux and

[17] Even Microsoft compared OpenOffice to MS Office 97 – arguably, few use any features introduced since then: http://www.eweek.com/article2/0,1759,1499560,00.asp

[18] http://www.census.gov/prod/2001pubs/p23-207.pdf

[19] http://devdata.worldbank.org/data-query/

other free software is that it is possible to take "old" technology and get far more value out of it today.

The main problems developing countries are likely to face in adopting free software are not technical, but political. Free software is something that people – politicians as well as businessmen – do not immediately understand and it requires some explanation before its advantages are apparent. Developing countries face increasing pressures from the rich world, through lobbying, in multilateral fora such as WSIS and tied aid (even support programmes at UNDP and UNESCO) to avoid free software or adopt short-term solutions that favour proprietary software. One example is the promotion of computers using proprietary software in schools, provided free of charge by the vendors. What happens once students leave the schools knowing only how to use specific, familiar, proprietary software applications, but no longer have access to the free-of-charge version? Software companies see giveaways to schools as an investment in future customers, maximising future returns for themselves, rather than an investment in future skilled citizens, maximising returns for the local society in which they live[20]. It is not only proprietary software companies that benefit from proprietary software sales: a colleague from an unnamed country noted cynically that free software, by reducing costs of software procurement, reduces potential kick-backs, which is not appealing to many politicians.

With political support, however, developing countries can use free software to lower costs, build local skills, create local businesses and enable their active participation in the global information society based on local strengths. The Extremadura example is apposite in this context, as their ICT policy was supported in the highest levels of regional government. Political support needs to be harnessed by highlighting the difference between access and participation. To quote Felipe Gonzalez, Prime Minister of Spain from 1982 to 1996, speaking at a conference on free software: "Access [to ICTs] is not enough, it is the ability to create, to add value, that is important"[21].

6. Lessons for domains other than software

The collaborative production of free software has been seized upon by the culture and other creative communities recently as a possible new model for domains other than software. One should be careful about assuming such transferability, as there are many differences. For one, software is purely functional and the only criterion that almost every free software programmer judges his software product by is that it runs well. So, free software programmers will never object if another programmer does something that clearly improves their code, because there is such a thing as a clear improvement. It will not be thought as defacing your work or abusing your moral rights as an author. So, free software licenses, for instance, do not require attribution and that is something that is quite often important in the cultural domain.

[20] Bill Gates quoted in Fortune Magazine, 1998, about tolerating piracy in China as a way to create new consumers: "Although about three million computers get sold every year in China, people don't pay for the software. Someday they will, though. And as long as they're going to steal it, we want them to steal ours. They'll get sort of addicted, and then we'll somehow figure out how to collect sometime in the next decade" http://archives.cnn.com/2000/TECH/computing/02/23/microsoft.china.idg/

[21] Recorded in English by the author from simultaneous translation. Open Source World Conference, 18-20 February 2004, Malaga, Spain. http://www.opensourceworldconference.com/2004/

In the issue of collaboration and rights, unfortunately in most Western societies collaborative ownership of production is not very common, so it is unclear how collaborative ownership can be defined. In free software people do not think about how Linux is owned, as long as they know they have access (in fact, Linux is a collective work with each contribution being implicitly copyrighted by each individual contributor). And, finally, there is of course the economic issue that it's quite possible to make a living out of free software, because of the way the software runs and the system of services you can build around the free software. Equivalent revenue models for other domains are not clear.

Where collaborative development models from free software do work well – indeed, free software was only one of many examples for the "cooking pot" model – is in pure information products, such as discussion groups and online information sources. Wikipedia is a great example of this. But most other domains of production, including pharmaceuticals, have tangible physical output even if they are based on some form of intellectual creativity and cannot fully benefit from the problem of infinity[22] that resulted in the free software solution.

There have been some examples of "free hardware"[23] such as India's Simputer[24]. These borrow from free software the concept of sharing design specifications – the "source code" of hardware, if you will – but unlike software, where translating from the source code to a program you can run is trivial and cheap, in hardware this is expensive with high barriers to entry. So it is not clear how much openness or freedom such efforts can spread.

Similarly, attempts to pool data on genetic resources between biotech firms (ENSEMBL[25] or the SNPS consortium[26]) bring the benefits of collaborative production mainly to those large players who can afford to collaborate in this resource-intensive domain. Certainly in this and other areas related to pharmaceuticals there is potential to reduce costs to society by encouraging re-use rather than re-invention, encouraging efficient (and competitive) collaboration rather than wasteful competitive replication. But it should be clear that free software emerged as a model of collaboration defined by the peculiarities of its domain of production – information – and any other domain of endeavour, while drawing on the lessons of free software, must develop its own unique systems of collaboration.

[22] Ghosh, R.A. 1995. "The problem with infinity", *Electric Dreams*, June 19. http://dxm.org/dreams/dreams63.html

[23] http://opencollector.org/Whyfree/

[24] http://www.simputer.org/simputer/faq/

[25] http://www.ensembl.org/

[26] http://snp.cshl.org/