Free Software – What you should know about it
Mike Godfrey and Andy Pepperdine
9 September 2006

The authors are members of both U3A in Bath and of BRLSI and are retired engineers in the area of computing and electronics. The talk covered the state of computing for the consumer, the effect of Microsoft's near-monopoly and a description of what software is and a brief history of it. It later describes how the Free and Open Software (FOSS) came about, the motivations of the volunteers who largely develop it and how it is maintained.

Introduction

There are about 1 billion PCs in the world, and everyone who has one is aware of the costs involved in keeping them clean, making sure that their licenses are up to date, and having to replace them regularly as later versions of software often create incompatibilities. How many are aware of true costs of ownership? Is there a cheaper alternative?

The mass media are beginning to notice the appearance of a new web-browser called Firefox. It can run on numerous types of PCs, is fast, is much less likely to be attacked by malformed web-sites, has several advanced features, but yet is completely free of charge for anyone who wishes to download it. Over 200 million copies have been taken off their website in 21 months, which is a remarkable achievement for something that is given away and where the consumer has to actively decide to get it and to install it even though Microsoft has supplied a web browser free in the system on delivery. How is this possible? Where does the resource come from? And what is the motivation of the volunteers who made it?

To answer that, we must examine what software is, and why it is not the same as any other kind of "good", and we'll start with a bit of history.

History

When electronic computers first came on the scene in the late 1940s, the only way of getting them to do something was to write a "program" (or software) in a low-level manner – a very tedious and error-prone process. To save each person repeating a similar job, it was normal to swap programs with other workers, and the discipline of computer science was founded in several universities. The algorithms that were invented were discussed in academic journals and were treated in the same way mathematics has always been treated. In fact they thought they were doing mathematics; the word algorithm was well-known with a clear meaning.

The essence of software is that it can be re-used by anyone for any purpose, whether it is the same job, or just a mathematically similar job. In addition, knowing what the algorithm looked like (because the source code was available) enabled each person to modify it for their own purposes. In this way, more and more software was written by re-using other people's work. It would have been more or less impossible to have built up all these programs without the ability to use previous work. And the more work that was done, the more there was available for others to use. Like scientists each engineer stood on the shoulders of giants.

During the 1950s and 1960s, commercial firms, such as IBM, began to sell computers to other companies. They were expensive and large. The software that went with them was given away as part of the package, and then each customer wrote more of their own to solve their own problems. By this time, software was being written in a higher level language, which made it easier, although

still labour intensive. It was recognised by everyone then that if they created standards for the languages, then other companies could come in and write software for clients on all machines without having to retrain to a different language. Details changed, but not the essence, and the market expanded for everyone to share.

However, it was getting harder and harder to maintain these systems as they became increasingly complex. But commercial realities meant that there was no interaction among companies on the details. Source code did not cross the boundaries, even though the ideas and techniques did. Then in 1969, Ken Thompson, Dennis Richie and others at Bell Labs in New Jersey wrote the kernel of a new operating system, which they called Unix. It broke new ground in that it was simple in concept, but yet extremely flexible. It provided exactly the sort of environment that helped software engineers understand the machinery of computers and operating systems. Since AT&T at that time was not allowed by anti-trust legislation to sell computing services, they distributed it free of charge to any university that asked for a copy.

At the same time, machines were beginning to come down in price to the point that small operations and university departments could think of owning their own for experimental and teaching purposes. The result was that many graduates obtained knowledge of the insides of Unix. With even lower costs of hardware, they even started to build their own machines at home, and Unix was there for them to use to make the machines do what they wanted.

During the 1970s, software began to attract more commercial interests, and with them the first signs of a tension between academic or hobby use and software for profit. In 1976, Bill Gates wrote his "Open Letter to Hobbyists" berating them for swapping ideas and source code without payment. He was sure that money could be made by selling the stuff. After all, once it is written, copying it is trivial and so any sale price must be almost all profit. He founded Microsoft with the business model of doing exactly that and by making advantageous commercial deals with the manufacturers of the then new PCs ensured that he got a huge slice of the market when ordinary consumers could afford one in every home.

Once there were enough machines in homes, an industry sprung up to sell games, educational programs, and system tools. All of these were based on the idea of licenses. The customer did not own the software, but was allowed to use it under certain conditions. In particular, they were not allowed to copy it to other machines.

Around that time, too, it was realised that customers often wanted a program for their own use, but that maintaining it did not take as much effort as writing it in the first place. This need was filled by freelance consultants who designed and wrote bespoke software, and then left, perhaps with a retainer to cover maintenance and added functions.

At present, the total software market is split roughly evenly among packaged programs, employed consultants, and small programs written for special purposes in all the IT departments of companies.

The advent of free software

In the late 1970s, much interesting work was being carried out at the Artificial Intelligence Labs (AI Labs) at MIT where Richard Stallman was working. Many of the staff were keen to start a commercial venture, with outside finance, to sell the products being created. Most of them joined the new company, but not Stallman, who was devastated by what he saw as the destruction of a close working group that was successful because it could swap ideas with anyone. He thought that the interplay of ideas among team members was the essence of an innovative and dynamic environment. But now he was locked out from the code that he himself had helped to write, but rather than giving up, he decided to do something about it. His first step was to re-write by himself the important tools that had been used and sell them for $150 per tape. These tools were quickly

seen by his university colleagues to be superior in many respects to many others around and bought them in quantity, which provided him with an income.

The next thing he did was to prove the cornerstone of all the subsequent free software development. Stallman is a visionary. He had understood why the development at AI Labs had been successful, and wanted to be sure that any code that he wrote would not again be exploited by anyone who was not prepared to share the results with him. He invented the notion of "Free" software, free in the sense of liberated and not tied down by restrictions. He wanted to be sure that anyone who received it was allowed to use it and modify it – so long as the results could also be freely distributed under the same conditions. Helped by Eben Moglen, a lawyer (now Professor of the History of Law at Columbia University), he created the General Public License (GPL), first issued in 1988. It relies on copyright law but inverts the usual restrictions and allows complete freedom to copy, use, and modify, but any changes must be made available under the same conditions. In other words, any work done under the GPL is there for everyone else to use in the same way. He was not interested in making money out of it, only in keeping a community of developers together to work on it.

With that as a sound legal base, he began to build a complete Unix system, with a few friends who thought the same way. The license has stood the test of time admirably and 70% of free software is now issued under the GPL. It ensures that the code will never disappear as long as anyone is using it. The license prevents companies from commandeering it and charging for its use; but it does not prevent a company from contracting to support it, or supplying hardware that uses it.

In the meantime, Unix had not had the benefit of such a well thought out license, and was now in the hands of several companies all trying to sell their versions of it, but not agreeing what the interfaces to it were. The market fragmented, and everyone suffered. The source code was getting locked away, and AT&T were no longer willing to give free access to it, even for educational purposes.

The market for software was getting divided, too. In the past, it was considered only as something necessary in order that the hardware can be used. But as the number of machines escalated, the number of potential customers for a program increased to the point where it was possible to write a product for very many customers. So they were put on shelves in boxes, neatly shrink-wrapped, just like many other consumer goods. Making these boxes was extremely cheap, but in order to recover the development costs, they were given restrictive licenses to prevent them from being copied around. Over time, these licenses have become more and more restrictive, to the point where most customers no longer understand what they can and cannot do with them. Microsoft had another money-making trick up its sleeve. They made deals with makers of PCs to put Windows on every machine shipped, and to pay a license for it. Microsoft no longer even had the costs of distribution for the Windows operating system. Today, the market for software is split almost evenly among off-the-shelf packages (like Microsoft), work written by consultants for specific customers (like the Passport Office), and things written by all the IT departments around the globe for use internally by their employers.

One person who was affected by this behaviour was Andrew Tanenbaum at the Free University of Amsterdam. He had been relying on Unix source code as part of his courses on operating systems and this was now not possible for legal reasons. After a while, he decided to write his own version, called Minix, a cut down and simple version for ease of understanding; and he released this under his own fairly free license, but kept strict control of its development because he needed it for his teaching. But the ideas behind Unix would not go away; they are just too valuable.

Thanks to Stallman, a considerable amount of software had been created under the GPL, but the most important piece was still missing – a kernel for the operating system. Everything still ran on non-free (by Stallman standards) operating system kernels.

All that changed in 1991. Linus Torvalds, then a student at Helsinki University, wanted to find out more about the machine he had just bought with money he had borrowed, but which he reckoned was going to be the future de facto standard for desktop machines, viz. Intel's 386. He had ordered a copy of Minix, and then started to examine the architecture both of the Minix kernel and the chip's hardware. In a typically global manner, the first person to give him any help was Bruce Evans from Australia. The Internet had also begun to show its worth to all higher teaching establishments – it had got away from its military origins and was in use by all computer science students.

After a few months, Linus was satisfied that he had not only mastered everything he needed, but had also completed his first version and sent it out for anyone to look at and comment on. In only one year from his purchase of his new machine, he had a kernel that was now the talk of the operating system mailing lists. It was given the name Linux by a fellow student at Helsinki who had some space available on a server that would allow anyone to get a copy.

Being a Unix-like system, it was eminently suitable as a base for all of Stallman's utilities and tools. All the interested engineers immediately saw that the combination was just what they were all waiting for, and to reflect the dual parentage it should strictly be called GNU/Linux.

Linux continued to spread itself around the Internet. By 1998, it was clear to many that it had become a serious contender for any application of computing. IBM made it run on all of its machines, thus giving the company a single platform to write applications for, instead of a collection of disparate ones. Almost all of the supercomputers now run Linux. Silicon chip manufacturers had adopted it as a way of testing their designs because they could modify the source and see what happened. Even the engineers inside Microsoft had noticed it and wrote an internal memo that laid out their vision of the future that would become dominated by the Internet run by Linux systems and that Linux would eventually push Windows off the desktop. It was leaked to the public and became known as the 1998 Halloween memo. Now, in 2006, free software is encroaching on more and more of the software market, as the Firefox experience shows.

Linus' genius was in realising that the key to success was in loosening the controls. When people sent him updates, modifications, or fixes for bugs, he welcomed them and incorporated them, stopping only to make sure that they would not break anything. The rate of progress after that was staggering. The way to involve the best people around the globe was now understood – keep everything completely open and encourage anyone to contribute. After only 15 years, it is suitable for desktops at home.

Motivation

Why does anyone write any code for Linux? In almost every case someone had an itch. Something didn't work properly. A piece of hardware was not supported. A program was missing. So they scratched the itch and wrote the code that was needed to make whatever it was work the way they wanted it. The users were now in control. And the GPL held it all together; none of the code could be exploited by anyone else. The users of a program could and did send comments and queries back to the developers, who were then “paid” by their reputation in the community, and a community is the right word for what was being unexpectedly built. Perhaps only Stallman had had the foresight to see that this was the natural result of his efforts.

Not all free software runs on Linux. A great deal of it works also (and some only) on Windows and on Apples of various flavours, as well as other lesser known systems. The Internet would not exist without it. Behind the scenes, most of the webservers that supply all the information to you when you surf the net run some form of Apache, a program that is now almost the de fact standard for the job; and it's free. It is almost impossible to send an email without it being passed on somewhere by a program called sendmail or a more modern replacement, all of which are free. When you type a URL into your browser it will ask a server to convert the name into a numerical address and the

server will run BIND (Berkeley Internet Naming Domain) another free piece of code.

The nature of software

Software is strange. It is not like any physical entity; much more like information. Its representation as source code is in a digital format, and can be copied at virtually zero cost. A single engineer needs only trivial investment to establish a research lab. The Linux kernel started in a student's bedroom. The Internet enables them to join forces across the globe. Physical location does not matter.

Writing software is akin to making a work of art; it needs careful thought and design, but it is never finished – there is always something else to add. It will die away only when insufficient people are using it.

It is easy to start a project and develop it to the point where it satisfies the original writer's itch. They can then let it out onto the world and see who had the same or similar itch. Responses will come back suggesting changes and additions. The users will also look at the code and provide additions themselves. This positive feedback gives an emotional fillip to the writer, who can then begin to form a community suited to its further development.

So far as free and open source is concerned, the users will never be caught by the supplier removing support or maintenance. The code is always present for anyone to take and continue to change as long as is necessary.

Standards are important, too. If a file format or protocol is freely available as a standard, without any patent or copyright encumbrances, then anyone could create a program to read and write it. Never again will files be lost due to being saved in an unknown format for which the source code has been lost.

Does free mean safe?

Everyone who has a PC at home knows the problems of unwanted emails, or spam. Almost all have also heard of viruses, or their colleagues trojans, worms, spyware and adware. Some may also know how they got there, and what they might do. We seem to have accumulated a lot of annoying junk with our valuable surfing tools. Without going into details, let's see whether we can sort out why this has occurred, and what could be done about it.

In its bid to get a PC on every desk, Microsoft concentrated on ease of use at the expense of security. They built a house, but put no lock on the front door. For the sake of minor inconvenience (like needing a key to get in) they allowed easy access to all and sundry. It is only recently that they have got on top of most of these issues, but it still leaves a lot of unchanged systems on desks throughout the world. These exposed systems fall prey to unscrupulous confidence tricksters and criminals who infect them with a virus, which lets them place a trojan on them. From that point, the machine is no longer doing only what the owner wants it to do, but is also obeying its criminal masters and sending out spam. Alas, spam will be with us for a long time yet.

FOSS, however, has been written by engineers for themselves. They use it personally. They are also aware of the theory of security and the reasons why certain designs are the best. They certainly do not want uninvited guests in their own systems; so they will make sure that what they write does not allow them in. And besides, their reputations are at stake.

Some people have said that it is only because Microsoft has such a dominant position that viruses can spread. But that is not the case. The designs are much better among the FOSS community. Consider the webserver Apache which is installed as the server for over half of the sites on the

Internet no matter how you measure it. One would think that if a worm was to attack web servers, it would be through Apache. But in fact, the only worms that have succeeded have used the minority player Microsoft's IIS. Apache has an enviable record of fixing security faults in a couple of days at most, and there have been only two serious alerts in the last 10 years.

The security model of Linux, like all versions of Unix, is well-known and been through over 30 years of computer science theory. It works. Another reason that viruses are so hard to write is that every distribution of Linux has been rebuilt with slightly different parameters, which makes it harder to tailor a virus to a single vendor's systems. Separating users from administrators also ensures that if anything should go wrong with a user account, then the damage is very limited. Unix style systems are very rarely re-booted for any other reason than scheduled breaks or hardware failure.

There is another aspect to security it is worth considering. If you use a licensed product, the End User License Agreement will probably say what you can do with the software. Typically, you can use it on one machine only, and then perhaps only until the vendor decides not to support it any longer at which point the license may terminate. If you try to transfer it to another person or machine, you may find you are not allowed to. But all your data files are yours, written with the help of this software. Transferring the files does not transfer the program that you have to use to read them. You will need another license, and another, and another,... so long as you have those files.

FOSS on the other hand places no restrictions at all on your use of it. The source is available publicly; there is no need for escrow. The file formats are standardised, so even if the particular program you are using fades into oblivion, there will always be others to use instead. This is security of access to your own work; you own the keys to your own filing cabinet.

What FOSS is ready now?

A great deal. The biggest repository of FOSS projects is SourceForge, which contains over 120,000 of them. Admittedly probably about two thirds are dormant, but that is still a lot work going on. If you use a search engine, you can find a program to do almost anything. They may still be rough around the edges, but they will be there. For Internet surfing and email, they are the leaders in function, usability and safety. Word processing and desktop publishing are equal to commercial offerings. Sound manipulation, image processing, and instant messaging are very good. FOSS has now reached into all areas with adequate tools, if not the very best.

It is readily found and downloaded. The costs may be in knowing how to install it and how to find help and support. Both of these questions will find ready answers in the on-line forums and mailing lists, which each project will have established. Putting a question will get an answer almost always within a day from someone who knows – they may be anywhere in the world, and you may not know where. They are all volunteers who are willing to help you get started and support you. Later, you may well be able to give something back by answering a question yourself.

Further Reading

Moody, Glyn. *Rebel Code: Linux and the open source revolution,* Penguin Books, 2002, ISBN 0-14-029804-5

Torvalds, Linus & Diamond, David. *Just for Fun: The story of an accidental revolutionary,* Texere, 2001, ISBN 1-58799-151-9

Brooks, Frederick, Jr. *The Mythical Man-Month: Essays on software engineering,* Addison Wesley, edited 1982, ISBN 0-201-00650-2

Goldman, Ron. & Gabriel, Richard. *Innovation Happens Elsewhere: Open source as business strategy,* 2006. Released under a Creative Commons license at http://dreamsongs.com/IHE/IHE.html Originally published by Morgan Kaufmann, 2005.

The present paper, the presentation given on 9 Sept 2006, a glossary, and a list of applications that can be used in place of some of the common proprietary offerings can all be found at http://www.pepperdine.eclipse.co.uk/FOSS

Andy Pepperdine
2006-08-28