WHY ACADEMIC SOFTWARE SHOULD BE OPEN SOURCE

Sébastien Paumier

Institut Gaspard-Monge, Université Paris-Est Marne-la-Vallée

Everyone has heard about “free software” and “open source”, but what those terms actually mean is not always very clear, even for people that produce software. First of all, I will briefly explain those terms, and the reader will see that the open source philosophy has many ideas in common with Science. Then, I will define what I mean by “academic software” and I will present some arguments in order to show that Science would benefit a lot if scientists that produce software released them as open source.

Free Software and Open Source

According to Richard Stallman, the creator of the Free Software Foundation, the term “free software” stands for a set of constraints[1]:

- *freedom to run the program, for any purpose; (1)*
- *freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this; (2)*
- *freedom to redistribute copies so you can help your neighbor; (3)*
- *freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this. (4)*

The main idea is that such software can be used and improved by anyone, with no restriction but one: if you modify the code of a free software, you have to publish the code of the modifications.

Free software can be used without any restriction. An example of common restriction is “for non commercial use only” (freedom 1).

Free software lives its own life, no one being able to lock its code. If you want to build your own version of a free software, you can, as does Novell that produces its own version of Open Office (freedoms 2 and 4).

A free software is free of access without any constraint. For instance, you don’t have to register any information (freedom 3). You can freely get it and share it. A major consequence is that free software is free of charge.

Basically, “free software” and “open source” mean the same. The only difference is a philosophical one. The free software community’s motivation is generosity: bringing free knowledge to humanity. The open source community’s is that it is more efficient to work in common, everyone bringing one’s own expertise in a “let do the guy who knows” perspective.

However, the term “free software” may be confused with “software that costs zero”, so I prefer the term “open source”.

Academic software

Many academic scientists produce software, sometimes in the frame of industrial partnership, sometimes not, for instance to implement an idea as a proof-of-concept for a scientific paper. In this paper, “academic software” means any piece of software written by an academic.

With such a definition, open source academic software seems to be incompatible with industrial constraints. However, open source software can be integrated into private software, if it has such a license as GNU’s LGPL[2]. This license allows users to link private code with open source code, without having to publish it as open source. So, in a cooperation context, academics can produce open source software, while industrial partners produce private code that will not be free software.

I make a distinction between academic software and software that involve scientific knownledge written by non-academics, because in the latter case, programmers are often engineers paid by private companies whose goals are different. Producing scientific knowledge is one thing; producing reliable user-friendly software is another. There is often a gap between a prototype and a final product, and it is not shocking that companies produce private scientific software, because they offer value added like graphical design, handling of many file formats, plugins, maintenance, etc. There are many examples of open source projects that have a commercial destiny. For instance, the open source Linux distribution Fedora is the basis of the commercial distribution Red Hat, whose core business is to select and test stable versions of Fedora and to guarantee compatibility with recent hardware as well as with major software products like Oracle.

Now, I will try to convince you that academic software should be open source.

Collective responsability and efficiency

Open source software has no master that decides of its evolution alone. For instance, if the authors of an open source software component decide to change its theoretical paradigm, users can decide to keep using the current version if it fits their needs. For instance, if your favourite open source calculus system has changed the + disjunction symbol into the U one, you can patch it to preserve the previous behavior in order not to mess all your existing programs. On the contrary, you can create dissident branches according to your own goals. With open source, you do not depend on a particular person or group.

This freedom is important, because when there is a bug or a missing feature in private software, users that are not computer scientists have to find tricks to handle the problem, most often with script languages like Perl and Python. In some cases, users can waste amazing amounts of time splitting a big data file into many small ones because of an arbitrary hard-coded file size limit. With open source software, such problems can easily be fixed up, because it is easier for non-programmers to find someone that can fix a small bug for them than to try and discover by themselves the inner mysteries of UTF8 encoding, file permissions, missing libraries, etc.

Moreover, academic scientists that produce software are not always programming experts. As a consequence, many academic software systems are black boxes that can hardly be reused or integrated in user-friendly graphical front-ends. Releasing an academic software as open source allows competent people to clean the code, for instance in order to build a library with a well-shaped interface. This aspect can be significant in an industrial partnership context.

Authentication

The fact that free software has a collective history can be felt by some authors as a double risk. First, authors can be afraid of being anonymous contributors that will not be acknowledged and cited. In fact, free software protects authors. Contributions can be signed with comments in source code. Moreover, free software licenses forbid to modify source code anonymously, so that you can know the author of each part of the software.

The second fear regards original software developers that want to control the evolution of their product. However, there is an easy solution to authenticate an “official” version of free software: giving it a name that you can protect by legal means. It is common practice in the open source world, in particular for Linux distributions. For instance, Ubuntu is derived from Debian[3], but both distributions are identified by their names that are trademarks and cannot be used freely.

Compatibility

Most non free software is distributed in binary form. Such binaries are specific to a given architecture and a given system (except for some languages like Java). So, you have no choice, you must adapt your computer to the software you want. For instance, in many labs, you may find a computer with a very specific system version, because some software is not compatible with newer versions. On the contrary, as free software products come with their sources, it is possible to recompile them for your own computer. By the way, it is a good reason not to code with system-restricted languages such as .NET, because compilers do not exist for all systems.

Peer-reviewing

Scientific papers are to be peer-reviewed in order to guarantee their quality. One goal of this review process is to verify that announced results are consistent with experimentation. However, in full rigor, you cannot review a paper about a software system if it is not testable. Note that the software need not be open source for this purpose, it just has to be runnable; but as we will see in the next section, open source is a big plus.

Study and improvement

An important guideline in Science is to improve current practices and results, when possible. With private software, how can you improve it if you don't know how it is made? Developers that have already tried to reproduce software just by reading the paper that goes with it know that it is not easy and sometimes not even possible. As you can hardly guess how software is made, your only option is to code your own idea and compare the results, but the comparison with a black box is not very useful unless there is a significant performance gap.

In fact, there are two main ways to improve software: improve the algorithms, improve the implementation. Improving an algorithm is often the easiest way, because its theorical cost (complexity) is emphasized in the scientific paper that describes it. You can claim to have designed an algorithm with a better complexity, without even implementing it. However, complexity depends on implementation also. For instance, when an algorithm says "if x belongs to the set C...", you may use a hashtable, a binary search tree, etc. So, if you don't know how the algorithm is coded, you cannot know if you could improve the software with a different implementation. It is the same for some low-level optimizations. You may highly improve performances, just by switching instructions or giving a buffer a size that is smart for the system. Such tricks may not be considered as Science proper, but if you tell physicists that their 7-days calculus could be achieved in only 6, they may be interested. Open source software is good for Science because scientists can clearly see how it works.

People don't like to change their tools

Some academic software is not used as much as it should. When existing software fits users' needs, there is no reason for switching to a new one, except if its interesting new features represent a major gain. If not, the new features will not be used, which will be a loss for the community. With open source software, it is not necessary to recode whole existing systems just in order to add something: you can concentrate your efforts on new features, and make them visible by adding them to systems that are already used.

Software lifetime

Another reason for academic software to die young is that it is often written by young hands that exit the world of research once their PhD is obtained. In most cases, such software is a prototype that only its author can run and modify, and when the author disappears, the software dies. Here, open source could offer an accessibility guarantee. If software is interesting, it will remain possible to explore and improve it, even if it has become an orphan.

Protecting good ideas by secrecy can be counterproductive

The history of cryptography has shown that secret is in data, not in programs. There is no program that cannot be rewritten if one takes the time to do it. Even if an algorithm is not public, the program that implements it is a sequence of instructions that can always be reverse-engineered. Protecting software with secrecy is only a temporary solution. In fact, if a private software is interesting enough, sooner or later it will be rewritten as open source. However, it may be a waste of time for programmers that recode it, and the major risk concerns the original authors. By locking their software, they encourage the birth of competing software. Once an open source clone exists, there will be a competition that original authors are not sure to win. On the contrary, if people release their work as free software from the begining, there won't be competition but inheritance, which is more interesting for academic acknowledgment.

Cumulative work

In an idyllic world, all academic software should be constructive and enrich the Science toolbox. The free software philosophy perfectly fits this perspective since open source software is easier to reuse than a paper describing a method, because you don't need to code it again to reuse it.

When a specialist provides free software that deals with his/her domain of expertise, people that do not have such expertise can benefit from it. It is more constructive for them than to badly study problems that have been solved for years. This is important, because science becomes highly specialized, and even in a small field, you cannot be an expert for all. In general, contributors with various skills or competences enrich open source software in diverse ways: spell checking code, translating comments, refactoring code, etc.

Many reasons for a good practice

Cumulative work, knowledge sharing and progress are ideals that science has in common with free software philosophy. Sharing source code is a constructive practice that could make the scientific community progress faster and improve its practices. An experimental result that no one can verify should not be eligible to scientific publication. We should apply the same rule to software: you should not publish a paper about a software system if it is not fully accessible.

[1] Free Software Foundation: www.fsf.org

[2] Gnu project: www.gnu.org

[3] http://www.ubuntu.com/community/ubuntustory/debian