Source code[1]

Roberto Di Cosmo
INRIA, Paris, France

> "Programs must be written for people to read, and only incidentally for machines to execute." (Harold Abelson)

In a computer system, there are generally two parts: the *hardware*, which is the physical part of the system (processor, memory, disks, screen, network card, sound card, keyboard, mouse, etc.), and the *software*, which designates the set of instructions that can be stored and executed by the machine. It is the software that gives life to a computer system.

Instructions that can be directly executed by a machine (called "machine language") are often very low-level, represented by simple sequences of bits and difficult to understand by a human being. This is why software is almost never produced directly in machine language, but "written" by developers using a "programming language", which can then be automatically translated into machine language.

As an example, here is an excerpt from the executable program that prints a simple "Hello world" message:

```
4004e6: 55
4004e7: 48 89 e5
4004ea: bf 84 05 40 00
4004ef: b8 00 00 00
4004f4: e8 c7 fe ff ff
4004f9: 90
4004fa: 5d
4004fb: c3
```

Here is the program written in the C programming language, from which the executable program from which we extracted the fragment presented above was produced:

```
/* Hello World program */#include<stdio
.h>
void main()
{
printf("Hello World");
}
```

The "source code" of software is generally understood to be the program that was written by the developer and from which the executable code is obtained on a machine. Also, one might be tempted to consider as "source code" any program written in a programming language. As technology has evolved, the situation has actually become more complex: developers have sophisticated tools that can "produce" programs in one programming language (such as C) from programs written in higher-level programming languages, to the point that it is not enough to look at the language in which a program is written to know whether that program was written by a developer or generated automatically from a higher-level program.

That is why the definition of "source code" for software, found in the GPL, is "the preferred form for a developer to make a modification to a program".

Source code is a special form of knowledge: it is made to be *understood by a human being,* the developer, and can be mechanically translated into a form to be *executed* directly on a machine. The very terminology used by the computer community is telling: "programming languages" are used to "write" software. As Donald Knuth, one of the founders of computer science, wrote, "programming is the art of explaining to another human being what you want a computer to do".

Software source code is therefore a *human creation*, just like other written documents, and that is why it falls within the scope of copyright law. Software developers thus deserve the same respect as other creators, and it is essential to ensure that any changes to copyright law take into account the potential impact on software development. This was not the case in the drafting of EU Directive 2019/790, the first draft of which seriously endangered the collaborative development of open source software, and which required a significant effort to make the necessary corrections.

As software source code becomes more and more complex, it is regularly modified by groups of developers who collaborate to make it evolve: to understand it, it has become essential to have access to its development history.

The software source code is thus incorporating *an important part* of our scientific, technical and industrial heritage, and thus constitutes a valuable heritage, as already argued by Len Shustek in an excellent article in 2006.

This is one of the missions of Software Heritage, an initiative launched in 2015 with the support of INRIA, in partnership with UNESCO, to collect, organize, preserve and make easily accessible all the source code publicly available on the planet, regardless of where and how it was developed or distributed. The goal is to build a common infrastructure that will allow a multiplicity of applications: of course, to preserve the source code in the long term against the risks of destruction, but also to

enable large-scale studies on the code and the current development processes, in order to improve them and thus prepare a better future.

At a time when it is clear that software has become an essential component of all human activity, unrestricted access to publicly available software source codes, as well as qualified information on their evolution, is becoming an issue of digital sovereignty for all nations. The unique infrastructure that Software Heritage is building, as well as its universal approach, is an essential element to meet this challenge of digital sovereignty, while preserving the commons dimension of the archive.

Bibliography

Abelson, H. (1984). *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge.

Di Cosmo, R. (2019). Saving software development from the European copyright reform. *Dicosmo.org* [Online]. Available at: https://www.dicosmo.org/MyOpinions/index.php?post/2019/04/17/Saving-software-development-from-the-European-copyright-reform.

Di Cosmo, R., Nora, D. (1998). *Le Hold-up planétaire : la face cachée de Microsoft*. Calman-Lévy, Paris.

Shustek, L.J. (2006). What Should We Collect to Preserve the History of Software?. *IEEE Annals of the History of Computing*, 28(4), 110–112 [Online]. Available at: https://doi.org/10.1109/MAHC.2006.78.

Software Heritage. Site [Online]. Available at: https://www.softwareheritage.org.