

Open Source Software

Silvio Peroni

silvio.peroni@unibo.it – <https://orcid.org/0000-0003-0530-4305> – [@essepuntato@scholar.social](https://www.elsevier.com/author/urls/uri=@essepuntato@scholar.social)

[Open Science \(A.Y. 2024/2025\)](#)

[Second Cycle Degree in Digital Humanities and Digital Knowledge](#)

[Alma Mater Studiorum - Università di Bologna](#)

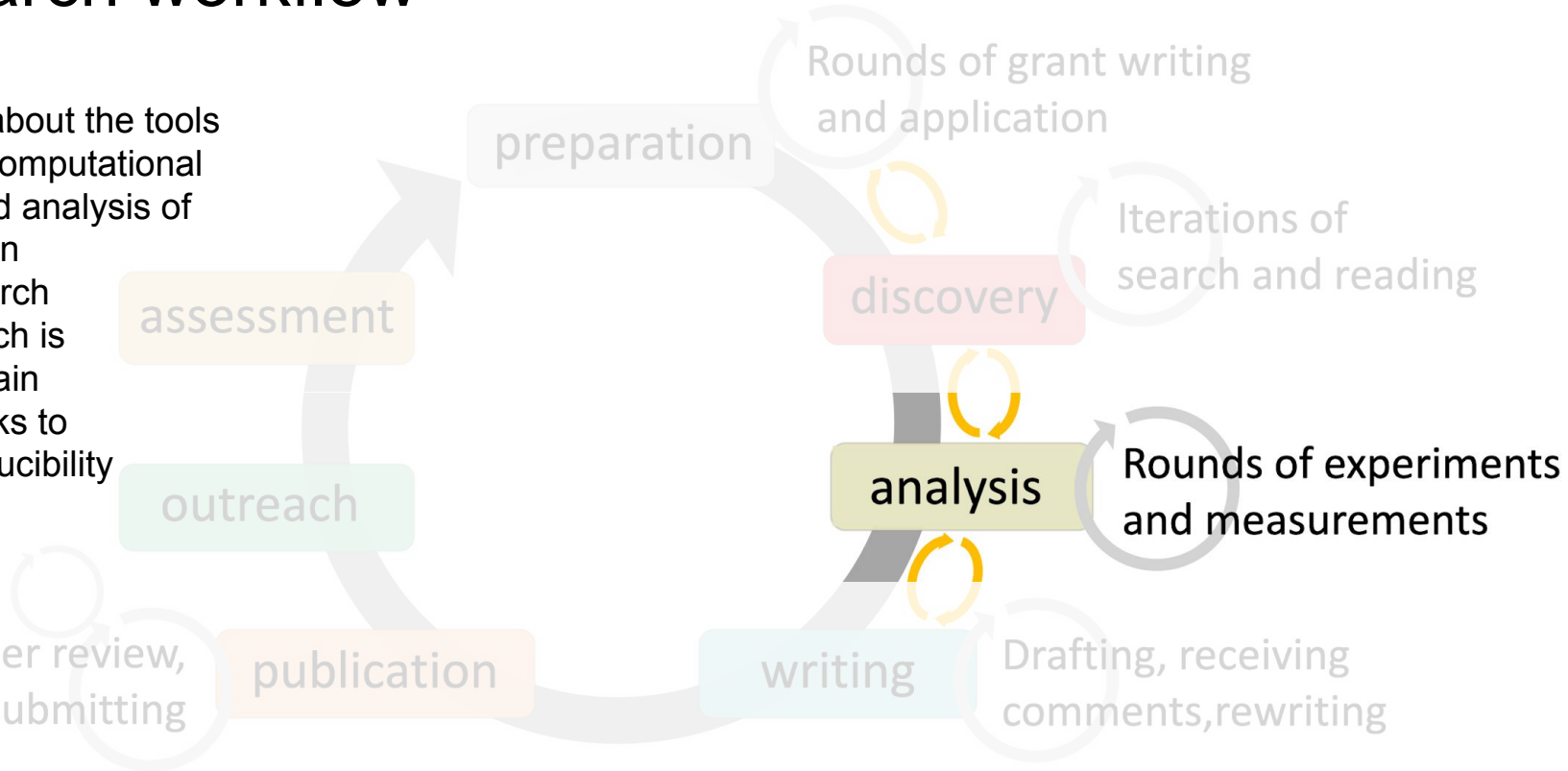


Discussion

Borisova, E., Abu Ahmad, R., Garcia-Castro, L., Usbeck, R., & Rehm, G. (2024).
Surveying the FAIRness of Annotation Tools: Difficult to find, difficult to reuse.
Proceedings of The 18th Linguistic Annotation Workshop (LAW-XVIII), 29–45.
<https://aclanthology.org/2024.law-1.4/>

Research workflow

We discuss about the tools that enable computational gathering and analysis of data, i.e. open source research software which is one of the main building blocks to foster reproducibility



Open source: a definition

Open source does not just mean access to the source code, but also:

1. **Free Redistribution** – not restrict from selling the software as a component of an aggregate distribution
2. **Source Code** – allow distribution in source code as well as compiled form
3. **Derived Works** – allow modifications and derived works, to be distributed under the same license
4. **Integrity of The Author's Source Code** – derived works carry a different name from the original software
5. **No Discrimination Against Persons or Groups** – not discriminate against any person or group of persons
6. **No Discrimination Against Fields of Endeavor** – not restrict anyone from making use of the program
7. **Distribution of License** – the rights apply to all to whom the program is redistributed
8. **Not Be Specific to a Product** – if the program is extracted from a distribution, all parties to whom the program is redistributed have the same rights as those that are granted in conjunction with the original distribution
9. **Not Restrict Other Software** – no restrictions on other software distributed along with the licensed software
10. **Be Technology-Neutral** – no predicate on any individual technology or style of interface

History of open source software

1960-70: The practice of sharing programs resulted in a push towards the creation of **portable software** — programs that could be ported to different computer platforms → a major step: development of the **Unix operating system**

1980s: AT&T began to sell licenses for Unix, reinforcing the drive towards the increased commercialization of software products – many programmers moved away from universities and research labs to private software firms, where they were bound by non-disclosure agreements. Richard Stallman founded the [Free Software Foundation \(FSF\)](#) with the aim of producing a non-proprietary operating system named [GNU](#) (a.k.a. *GNU is Not Unix*) and introduced a particular licensing procedure called [General Public License \(GPL\)](#) (with **viral clause**, a.k.a. copyleft)

1990s: Linus Torvalds created Linux, that **could compete successfully** in stability and reliability with commercial versions of Unix → in the following years, the community of developers working around Linux grew exponentially and got a semi-official recognition of its potential when a Microsoft internal memorandum indicated that Linux as **a major competitive threat** for the company

From the Cathedral to the Bazaar

In 1999, Eric S. Raymond wrote “The Cathedral and the Bazaar”, where he introduced two archetypical modes of software development:

- **cathedral mode**, typical of commercial software – software is developed from a unified a priori project that prescribes all the functions and the features to be incorporated in the final product, and the work is centrally coordinated and supervised in order to assure the integration of various components
- **bazaar mode**, typical of open source software – where software emerges from an unstructured evolutionary process, starting from a minimal code released by someone that is extended by groups of programmers with minimal coordination

The **owner** of the project, i.e. the person in charge of the project responsible to release versions of it, checks alterations to the software proposed by the community and integrate the most valuable ones in future releases in a way to assure that the overall project will evolve in a coherent way

Importance of research software

Software has been a part of the research environment for many decades, and the people who write, maintain and manage this research software are increasingly seen as critically important members of research teams

Good open source research software can make the difference between **valid, sustainable, reproducible** research outputs and short-lived, potentially unreliable or erroneous outputs

Areas that should providing comprehensive support for research software (4 pillars):

- **Development** – maintainability, sustainability and robustness are core aspects of quality software
- **Community** – communication, learning from others, and keeping up with the latest developments
- **Training** – ensuring initial and ongoing skills development for people building research software
- **Policy** – institutional and national policies that recognise the importance of research software

Some rules for open development of software

1. **Do not Reinvent the Wheel**: reuse existing software when possible
2. **Code Well**: study others' code and learn by practicing
3. **Be Your Own User**: address important questions and be useful to others
4. **Be Transparent**: allow many eyes to evaluate the code and fix any issues
5. **Be Simple**: create online documentation, sample data files, and test cases
6. **Do not Be a Perfectionist**: release early, release often
7. **Nurture and Grow Your Community**: acknowledge the tools you are using and the contributions of each person to your project
8. **Promote Your Project**: appearance matters, websites help advertisement
9. **Find Sponsors**: some level of funding is essential in the long-term
10. **Science Counts**: allow leadership to be shared and passed to other members

Recognising software as a citable contribution

Being a specific output of research, software should be considered a first-class citizen of the research endeavour and should be cited accordingly

Citations enable the specific software used in a research product to **be found**, provide **credit** to the scholars who developed it, and enable the **reproducibility** of the research – even if **it is not sufficient** to reference the software for having full reproducibility since other aspects – such as configurations, platform issues, documentation – are also needed

Important aspect for software: identifying who deserves to be credited when reusing a software is a difficult task since people may be involved in several distinct roles in a research software development

Shades of authorship in research software

Coding: difficult to clearly identify, since it can be manifested in several ways (e.g. line of code written, fixing code, or removing portions of code by factoring the project and increasing its modularity and genericity)

Testing and debugging: an essential role when developing software that may require setting up a large database of relevant use cases and devising a rigorous testing protocol

Algorithm design: inventing the underlying algorithm that is then implemented in a code

Software architecture design: essential for maintenance, modularity, efficiency and evolution of the software

Documentation: essential to ease (re)usability and to support long term maintenance and evolution

Long-term preservation of software

Software source code is a precious, unique form of knowledge which is **part of our cultural heritage** and is spread around a variety of platforms and infrastructures and that is often tracked through version control systems which enables one to study its evolution

However, as any form of knowledge, software can be **deleted, corrupted, misplaced, or shut down**, and this can affect thousands of publicly available software projects at once – while being a threat to reproducibility when it has been used to perform a research

To this end, it is crucial to set up appropriate infrastructures to preserve it in the long-term: [Software Heritage](#) is a universal software archive that collects and preserves software in source code form

Mandatory text to read for the next lecture

Hyde, A. (2025). PRC vs The Cathedral: How PRC Might Change Publishing (Version 1).

https://www.robotscooking.com/content/files/2025/01/PRC-vs-The-Cathedral_V1.pdf

End

Open Source Software

Silvio Peroni

silvio.peroni@unibo.it – <https://orcid.org/0000-0003-0530-4305> – [@essepuntato@scholar.social](https://twitter.com/essepuntato)

[Open Science \(A.Y. 2024/2025\)](#)

[Second Cycle Degree in Digital Humanities and Digital Knowledge](#)

[Alma Mater Studiorum - Università di Bologna](#)

