Open Source Software

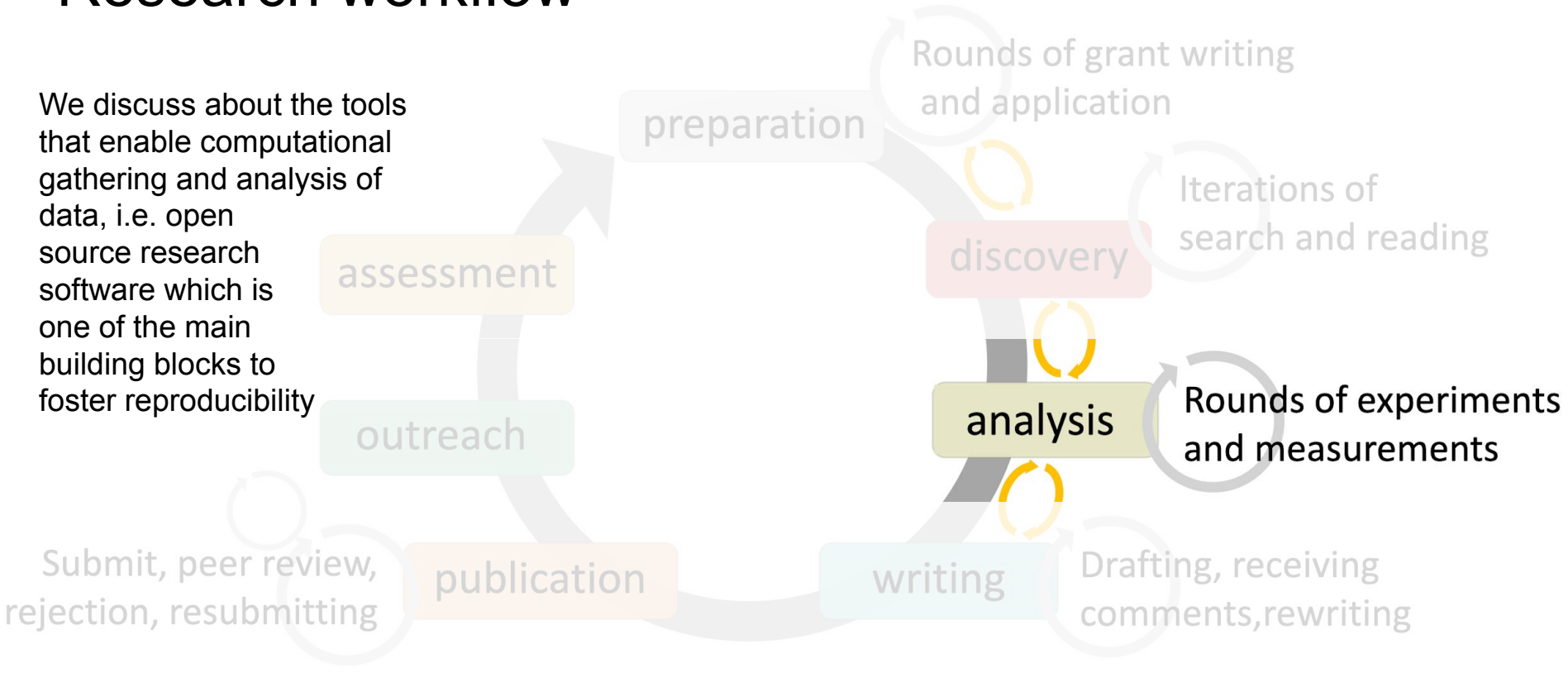
Silvio Peroni
silvio.peroni@unibo.it – https://orcid.org/0000-0003-0530-4305 – @essepuntato

Open Science (A.Y. 2020/2021)
Second Cycle Degree in Digital Humanities and Digital Knowledge
Alma Mater Studiorum - Università di Bologna

DIPARTIMENTO DI FILOLOGIA CLASSICA E ITALIANISTICA

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



Kramer, B., & Bosman, J. (2015, June 18). The good, the efficient and the open—Changing research workflows and the need to move from Open Access to Open Science. CERN Workshop on Innovations in Scholarly Communication (OAI9), University of Geneva, Geneva, Switzerland. https://www.slideshare.net/bmkramer/the-good-the-efficient-and-the-open-oai9

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

Open Source Initiative. (2007). The Open Source Definition. https://opensource.org/osd

The origins: Unix

The practice of sharing programs (between 1960-70, especially within universities) 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 (OS) by Ken Thompson at Bell Labs in 1969, which was an OS that could be run an wide range of machines, with the related creation of a community of Unix users

Shift in the 1980s: AT&T began to sell licenses for Unix, the widespread diffusion of personal computers reinforced 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 (i.e. they cannot share code anymore)

Nuvolari, A. (2005). Open source software development: Some historical perspectives. First Monday. https://doi.org/10.5210/fm.v10i10.1284

Free Software Foundation

In 1984, Richard Stallman founded the Free Software Foundation (FSF) with the aim of repurposing the open environment of computing shown in its early history – "you should think of 'free' as in 'free speech,' not as in 'free beer'"

Stallman and the FSF wanted to produce a non-proprietary operating system named GNU (a.k.a. *GNU is Not Unix*)

They recreated a non-proprietary GNU version of many components of Unix, developed in such a way that they could run on almost every version of Unix

To protect the freedomness of GNU software, Stallman introduced a particular licensing procedure called General Public License (GPL), which permitted the free distribution, modification and redistribution of a modified version of the programs it covers, but modified versions of programs licensed under the GPL had to be also licensed under the same terms – the viral clause (a.k.a. copyleft)

Nuvolari, A. (2005). Open source software development: Some historical perspectives. First Monday. https://doi.org/10.5210/fm.v10i10.1284

Linux

In 1991, Linus Torvalds was working on a free version of Unix, with the intent to include, in future versions, new features developed by others as long as they would have also been freely redistributable (GPL was used to license Linux)

Linux 1.0 was released in 1994 and 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

The semi-official recognition of its potential came in 1999, when a Microsoft internal memorandum indicated that Linux (and, in more in general terms, the diffusion of the open source as a process of software production) was a major competitive threat for the company

Nuvolari, A. (2005). Open source software development: Some historical perspectives. First Monday. https://doi.org/10.5210/fm.v10i10.1284

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

Raymond, E. S. (2000). The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary (Version 3.0). O’Reilly Media.
http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

New version control systems

Version control is a system that enables one to track changes to software over time so that you can recall specific versions later if needed

Since 2002, the Linux community began using a proprietary version control system called BitKeeper, so as to better coordinate the development effort in particular in the development of the Linux kernel

While the BitKeeper company provided the tool free-of-charge for the community, in 2005 it decided to revoke such a status

This prompted Linus Torvalds to develop their own tool based on some of the lessons they learned while using BitKeeper, and the result was the release of Git, a free and open source distributed version control system designed to handle everything from small to very large projects

Chacon, S., & Straub, B. (2014). Pro Git (Second edition). Apress. https://git-scm.com/book/en/v2

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

Cohen, J., Katz, D. S., Barker, M., Chue Hong, N., Haines, R., & Jay, C. (2021). The Four Pillars of Research Software Engineering. IEEE Software, 38(1), 97–105. https://doi.org/10.1109/MS.2020.2973362, available in OA at https://arxiv.org/abs/2002.01035

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

Prlić, A., & Procter, J. B. (2012). Ten Simple Rules for the Open Development of Scientific Software. PLoS Computational Biology, 8(12), e1002802. https://doi.org/10.1371/journal.pcbi.1002802

Open source software and open scholarship

One fundamental element that open source and open scholarship have in common is that they believe that the outputs of the wider process of organised knowledge creation, including intellectual properties that they represent, deserve to be a public good, rather than proprietary property

In addition, while software alone is insufficient for enabling reproducibility, sharing it is a minimal requirement for reproducible research

Collaboration in open source projects is essentially continuous and iterative as a progressive form of verification, where public disclosure of bugs and issues that get resolved sequentially make the process of fixing and improving the software transparent

Since scholarly research is often relying on computation, it is crucial that scholars understand how to apply collaborative practices from open source development in their research workflow

Tennant, J., Agarwal, R., Baždarić, K., Brassard, D., Crick, T., Dunleavy, D. J., Evans, T. R., Gardner, N., Gonzalez-Marquez, M., Graziotin, D., Greshake Tzovaras, B., Gunnarsson, D., Havemann, J., Hosseini, M., … Yarkoni, T. (2020). A tale of two 'opens': Intersections between Free and Open Source Software and Open Scholarship [Preprint]. SocArXiv. https://doi.org/10.31235/osf.io/2kxq8

Towards FAIR for research software

Applying the FAIR principles to research software enables transparency, reproducibility and reusability of research, and should facilitate making FAIR data

However, software is not data and, as such, some adaptation and contextualisation of the FAIR principles should be provided

Indeed, some quality aspects concerning the form of research software can be considered as covered by FAIR (e.g. the interoperability and reusability principles), but others, like those concerning the functionality of research software, go beyond what is covered by the FAIR principles

Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., van de Sandt, S., Ison, J., Martinez, P. A., McQuilton, P., Valencia, A., Harrow, J., Psomopoulos, F., Gelpi, J. Ll., Chue Hong, N., Goble, C., & Capella-Gutierrez, S. (2020). Towards FAIR principles for research software. Data Science, 3(1), 37–59. https://doi.org/10.3233/DS-190026

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

Smith, A. M., Katz, D. S., Niemeyer, K. E., & FORCE11 Software Citation Working Group. (2016). Software citation principles. PeerJ Computer Science, 2, e86. https://doi.org/10.7717/peerj-cs.86

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

Alliez, P., Cosmo, R. D., Guedj, B., Girault, A., Hacid, M.-S., Legrand, A., & Rougier, N. (2020). Attributing and Referencing (Research) Software: Best Practices and Outlook From Inria. Computing in Science & Engineering, 22(1), 39–52. https://doi.org/10.1109/MCSE.2019.2949413, available in OA at https://hal.archives-ouvertes.fr/hal-02135891

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 thought 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

Abramatic, J.-F., Di Cosmo, R., & Zacchiroli, S. (2018). Building the universal archive of source code. Communications of the ACM, 61(10), 29–31. https://doi.org/10.1145/3183558, available in OA at https://hal.archives-ouvertes.fr/hal-02157125

End

Open Source Software

Silvio Peroni
silvio.peroni@unibo.it – https://orcid.org/0000-0003-0530-4305 – @essepuntato

Open Science (A.Y. 2020/2021)
Second Cycle Degree in Digital Humanities and Digital Knowledge
Alma Mater Studiorum - Università di Bologna

DIPARTIMENTO DI FILOLOGIA CLASSICA E ITALIANISTICA