Six decades of libre scientific software

Nicolas M. Thiéry

Laboratoire Interdisciplinaire des Sciences du Numérique
Université Paris-Saclay

Mathematical Software and High Performance Algebraic Computing
June 26th-30th 2023, Lyon, France

Caveats and biases

Caveats and biases

Nothing but a personal testimony

Caveats and biases

Nothing but a personal testimony

My world, my biases

- Reborn to Libre Software in 1992
- Jean Thiéry @ ALDIL 1999: quatre décennies de logiciels [...] scientifiques libres
- Computer Science for Mathematics
- Software as a **research tool** more than a **research outcome** or **research object**
- GNU/Linux, Python, SageMath, Emacs, Conda, Jupyter, Mutt, …

Another annecdote

Another annecdote

Lesson learned the hard way

Typical code for research: a thin layer of pixie dust on top of a pile of generic stuff

Lesson learned the hard way: when you fail to be FAIR

- I could not **Find** my own best friend's code! 😢
  - It needed generalization
- I could not **Access** his code:
  - It was not published
  - I did not have a Maple license
- Anyway Maple and MuPAD were not **Interoperable**
- Thereby, we could not **Reuse** each others code

Lesson learned the hard way: when you fail to be FAIR

- I could not **Find** my own best friend’s code! 😢
  - It needed generalization
- I could not **Access** his code:
  - It was not published
  - I did not have a Maple license
- Anyway Maple and MuPAD were not **Interoperable**
- Thereby, we could not **Reuse** each others code

A shame: by **sharing** we could have saved ~50% of development time

Meaning **more research** 🔬 🎖 (and more juggling 🎉)

*-Combinat: Sharing Algebraic Combinatorics software since 2000

- Apply induction from two to a community!
- By bringing libre software and best practices to research software

*-Combinat: Sharing Algebraic Combinatorics software since 2000

- Apply induction from two to a community!
- By bringing libre software and best practices to research software

Don’t get me started on this …

A brief historical perspective

1960's: primordial *structured* libre research software

- FORTRAN (Formula Translating System) gains adoption
  $\implies$ portability and simplicity (**Interoperability**👍, **Reuse** 👍)
- Punch cards → Tapes

1960's: primordial *structured* libre research software

- FORTRAN (Formula Translating System) gains adoption
  $\implies$ portability and simplicity (**Interoperability** 👍, **Reuse** 👍)
- Punch cards → Tapes

A pioneer QCPE: Quantum Chemistry Program Exchange

- Mission: index, archive and distribute programs in Quantum Chemistry, and beyond!
- A newsletter advertises new additions (**Find** 👍)
- Ships copies of the program for cost of operation fee (**Access** 👍)
- Builds a community, organize workshops (hackathons!)
- Most programs were effectively libre software: (**Reuse** 👍)
  freedom to use, scrutinize, modify, and distribute modifications
- Hundreds of programs (typically 1-2 authors, 100 lines)

1960’s: primordial *structured* libre research software

- FORTRAN (Formula Translating System) gains adoption
  $\Longrightarrow$ portability and simplicity (**Interoperability** 👍, **Reuse** 👍)
- Punch cards → Tapes

A pioneer QCPE: Quantum Chemistry Program Exchange

- Mission: index, archive and distribute programs in Quantum Chemistry, and beyond!
- A newsletter advertises new additions (**Find** 👍)
- Ships copies of the program for cost of operation fee (**Access** 👍)
- Builds a community, organize workshops (hackathons!)
- Most programs were effectively libre software: (**Reuse** 👍)
  freedom to use, scrutinize, modify, and distribute modifications
- Hundreds of programs (typically 1-2 authors, 100 lines)

- **Urgent task: collect and archive the QCPE software!!!**

1970’s: Early libraries

Practical limitation (Reuse👎)

- Sharing pattern: distribute programs
- Reuse pattern: copy and adapt

1970's: Early libraries

Practical limitation (Reuse 👎)

- Sharing pattern: distribute programs
- Reuse pattern: copy and adapt $\Longrightarrow$ **does not scale!!!**

1970's: Early libraries

Practical limitation (Reuse👎)

- Sharing pattern: distribute programs
- Reuse pattern: copy and adapt $\Longrightarrow$ **does not scale!!!**

Example: LinPack

- A collection of FORTRAN subroutines for Linear Algebra
- Modularity (**Reuse**👍)

1970’s: Early libraries

Practical limitation (Reuse 👎)

- Sharing pattern: distribute programs
- Reuse pattern: copy and adapt $\Longrightarrow$ **does not scale!!!**

Example: LinPack

- A collection of FORTRAN subroutines for Linear Algebra
- Modularity (**Reuse** 👍)

Example: BLAS (1979)

- Basic Linear Algebra Subroutines (library $\longrightarrow$ interface)
- Even more modularity (**Reuse** 👍)

1980's: Scientific computing at the fingertip of researchers

- Generalization of personal computers
  $\Longrightarrow$ A researcher can have a desktop and use it for **interactive computation**.

1980’s: Scientific computing at the fingertip of researchers

- Generalization of personal computers
  $\Longrightarrow$ A researcher can have a desktop and use it for **interactive computation**.

Example: Voyons (Jean Thiéry, CEA)

- Integrated interactive environment for statistics, modeling, simulations, visualization
- **Innovations:**
  - **Target** non specialists
  - **Coconstruct** by participating to the research
  - **Reuse** across diverse research projects: NMR spectrography, agronomy, …
  - **Open source:** complete control on the algorithms
  - **Credit by citation**
- Early forms of **Agile development** and **Research Software Engineer**

1980's: Scientific computing at the fingertip of researchers

- Generalization of personal computers

  $\Longrightarrow$ A researcher can have a desktop and use it for **interactive computation**.

Example: Voyons (Jean Thiéry, CEA)

- Integrated interactive environment for statistics, modeling, simulations, visualization
- **Innovations:**
  - **Target** non specialists
  - **Coconstruct** by participating to the research
  - **Reuse** across diverse research projects: NMR spectrography, agronomy, …
  - **Open source:** complete control on the algorithms
  - **Credit by citation**
- Early forms of **Agile development** and **Research Software Engineer**

How to scale?

- Requires reaching a critical mass
- Lack of collaboration means $\implies$ collocated team

1980’s: Scientific computing at the fingertip of researchers (continued)

- Computers on anyone’s desk
- Technology ripe for “user friendly” programming languages

1980's: Scientific computing at the fingertip of researchers (continued)

- Computers on anyone's desk
- Technology ripe for "user friendly" programming languages

$\Longrightarrow$ Potential for a **mass of users**! $\Longrightarrow$ It's worth **investing**

1980's: Scientific computing at the fingertip of researchers (continued)

- Computers on anyone's desk
- Technology ripe for "user friendly" programming languages

$\Longrightarrow$ Potential for a **mass of users**! $\Longrightarrow$ It's worth **investing**

Archetype: MatLab turns to a commercial product (also: **Maple**, **Mathematica**,)

- General purpose numerical computing environment
- Wraps numerical libraries (LinPack, …)
- In a tailored programming language
- $\Longrightarrow$ brings computing to the masses, e.g. teaching 👍
- $\Longrightarrow$ generates revenue to fund a collocated team of developers 👍
- $\Longrightarrow$ critical mass

1980's: Scientific computing at the fingertip of researchers (continued)

- Computers on anyone's desk
- Technology ripe for "user friendly" programming languages

$\Longrightarrow$ Potential for a **mass of users**! $\Longrightarrow$ It's worth **investing**

Archetype: MatLab turns to a commercial product (also: **Maple**, **Mathematica**,)

- General purpose numerical computing environment
- Wraps numerical libraries (LinPack, …)
- In a tailored programming language
- $\Longrightarrow$ brings computing to the masses, e.g. teaching 👍
- $\Longrightarrow$ generates revenue to fund a collocated team of developers 👍
- $\Longrightarrow$ critical mass

- $\Longrightarrow$ silos between developers and users 👎
- $\Longrightarrow$ silos between environments 👎

1980's: Scientific computing at the fingertip of researchers (continued)

- Computers on anyone's desk
- Technology ripe for "user friendly" programming languages

$\Longrightarrow$ Potential for a **mass of users**! $\Longrightarrow$ It's worth **investing**

Archetype: MatLab turns to a commercial product (also: **Maple**, **Mathematica**,)

- General purpose numerical computing environment
- Wraps numerical libraries (LinPack, …)
- In a tailored programming language
- $\Longrightarrow$ brings computing to the masses, e.g. teaching 👍
- $\Longrightarrow$ generates revenue to fund a collocated team of developers 👍
- $\Longrightarrow$ critical mass
- $\Longrightarrow$ silos between developers and users 👎
- $\Longrightarrow$ silos between environments 👎

“the hardware is the product” $\longrightarrow$ “the software is the product”

1980’s: A new Hope

1980’s: A new Hope

Archetype: GAP: Group, Algorithms and Programming (also: Singular, CoCoA, Macaulay, …)

- A community gets together and decides to share
- Developed by users for users
- Dedicated programming language
- Library
- Packages

1980’s: A new Hope

Archetype: GAP: Group, Algorithms and Programming (also: Singular, CoCoA, Macaulay, …)

- A community gets together and decides to share
- Developed by users for users
- Dedicated programming language
- Library
- Packages

Libre software is formalized

- A response to closing sources hurting **ethics** and **practice**
- Freedom to **use**, **scrutinize**, **modify** and **redistribute modifications**
- Remember: copyright is about balancing the needs of both authors and users

1990's: Scientific computing for the masses

- Internet for the masses: web, chat, forums, mailing lists, …

1990’s: Scientific computing for the masses

- Internet for the masses: web, chat, forums, mailing lists, …

- Systems gain momentum (**Access** 👍, **Reuse** 👍)
- Much easier to build communities and user groups
- Online archives of user contributions: (**Access** 👍, **Reuse** 👍)
  CPAN, CRAN, CTAN, …
  Maple shared library, …

Late 1990': A growing frustration

Ethical concerns

> *You can read Sylow's Theorem and its proof in Huppert's book in the library, then you can use Sylow's Theorem for the rest of your life free of charge, but for many computer algebra systems license fees have to be paid regularly …*

> *With this situation two of the most basic rules of conduct in mathematics are **violated**: In mathematics **information is passed on free of charge** and **everything is laid open for checking**.*
>
> *Joachim Neubüser (started GAP in 1986) 1995*

Late 1990’: A growing frustration (continued)

Practical concerns

- Silos by system: license, language, community
- Silos by role: developers / users
- Silos by institution and physical location

$\Longrightarrow$ **Fragments the community and the forces**

Late 1990’: A growing frustration (continued)

Practical concerns

- Silos by system: license, language, community
- Silos by role: developers / users
- Silos by institution and physical location

$\Longrightarrow$ **Fragments the community and the forces**

- Increasing institutional pressure to **valorize** research software **as commercial products**, **as closed**

$\Longrightarrow$ **Killed many cool pieces of software** 💀 (e.g. Axiom)

Late 1990’: A growing frustration (continued)

Practical concerns

- Silos by system: license, language, community
- Silos by role: developers / users
- Silos by institution and physical location

$\Longrightarrow$ **Fragments the community and the forces**

- Increasing institutional pressure to **valorize** research software **as commercial products, as closed**

$\Longrightarrow$ **Killed many cool pieces of software** 💀 (e.g. Axiom)

Reuse👎 **Sustainability**👎

2000’s: The return of libre computing

2000's: The return of libre computing

- "User friendly" general purpose programming languages: Python, Perl, …
- Software Forges (SourceForge, …, GitHub, GitLab, …) + more best practices + physical ubiquity

$\implies$ massive collaboration

2000’s: The return of libre computing

- “User friendly” general purpose programming languages: Python, Perl, …
- Software Forges (SourceForge, …, GitHub, GitLab, …) + more best practices + physical ubiquity

  $\implies$ massive collaboration

Question:

- Viable libre software development models for large systems?
- “by users for users”?

2000’s: The return of libre computing

- “User friendly” general purpose programming languages: Python, Perl, …
- Software Forges (SourceForge, …, GitHub, GitLab, …) + more best practices + physical ubiquity

  $\Longrightarrow$ massive collaboration

Question:

- Viable libre software development models for large systems?
- “by users for users”?

- The Scientific Python stack challenged Matlab
- SageMath challenged Maple and Mathematica
- R challenged S, SAS, …
- …

2010’s: libre scientific software at scale

- Social networks, cloud infrastructure, and services for the masses
- More best practices
- Open Science gets momentum and recognition by institutions
- Multiplication of devices (tablets, “smart” phones, …)

2010’s: libre scientific software at scale

- Social networks, cloud infrastructure, and services for the masses
- More best practices
- Open Science gets momentum and recognition by institutions
- Multiplication of devices (tablets, “smart” phones, …)

A massive international collaboration across academia, industry, and more

On digital commons

2010’s: libre scientific software at scale

- Social networks, cloud infrastructure, and services for the masses
- More best practices
- Open Science gets momentum and recognition by institutions
- Multiplication of devices (tablets, “smart” phones, …)

A massive international collaboration across academia, industry, and more

On digital commons

Supported by infrastructures, best practices, funding, …

- Massive modularity across systems (**Compose**, **Reuse**👍)
- Software forges (**Find**, **Access**👍) and collaborative tools (**Community**👍)
- Package management and hosting (conda, guix, pip, npm, …) (**Find**, **Access**, **Reproduce**👍)
- Archival: Software Heritage (**Find**, **Access**, **Credit**, **Legacy**👍)
- Virtual environments (**Access** 👍)
- Literate Computing (**Access**, **Reproduce**👍)
- Community building: training, workshops, hackathons (**Community**👍, **Environment**👎)

2010’s: libre scientific software at scale

- Social networks, cloud infrastructure, and services for the masses
- More best practices
- Open Science gets momentum and recognition by institutions
- Multiplication of devices (tablets, “smart” phones, …)

A massive international collaboration across academia, industry, and more

On digital commons

Supported by infrastructures, best practices, funding, …

- Massive modularity across systems (**Compose**, **Reuse**👍)
- Software forges (**Find**, **Access**👍) and collaborative tools (**Community**👍)
- Package management and hosting (conda, guix, pip, npm, …) (**Find**, **Access**, **Reproduce**👍)
- Archival: Software Heritage (**Find**, **Access**, **Credit**, **Legacy**👍)
- Virtual environments (**Access** 👍)
- Literate Computing (**Access**, **Reproduce**👍)
- Community building: training, workshops, hackathons (**Community**👍, **Environment**👎)

And the Research Software Engineer (RSE) movement!!!

2020's: The next challenges?

- From physical ubiquity to virtual ubiquity (**Environment** 👍, **Joy** 👎, **Community** ?)
- Internet anywhere, anytime
- Fluidity: local vs remote, compiled vs interpreted, gradual typing, multi-paradigms programming
- Growing digital Commons of tens of thousands of packages to compose from 👍
  Built by hundreeds of thousands of developers worldwide 👍
- Machine Learning for the masses

2020's: The next challenges?

- From physical ubiquity to virtual ubiquity (**Environment** 👍, **Joy** 👎, **Community** ?)
- Internet anywhere, anytime
- Fluidity: local vs remote, compiled vs interpreted, gradual typing, multi-paradigms programming
- Growing digital Commons of tens of thousands of packages to compose from 👍
  Built by hundreeds of thousands of developers worldwide 👍
- Machine Learning for the masses

Will that scale?

- Complexity (**Find** 👎, **Reuse** 👎)
- Potential compatibility nightmare (**Sustainability** 👎)
- Reliability? e.g. nodejs' breakages (**Sustainability** 👎)
- Silos (**Reuse** 👎, **Collaboration** 👎)
- Environmental impact???

2020's: The next challenges?

- From physical ubiquity to virtual ubiquity (**Environment** 👍, **Joy** 👎, **Community** ?)
- Internet anywhere, anytime
- Fluidity: local vs remote, compiled vs interpreted, gradual typing, multi-paradigms programming
- Growing digital Commons of tens of thousands of packages to compose from 👍
  Built by hundreeds of thousands of developers worldwide 👍
- Machine Learning for the masses

Will that scale?

- Complexity (**Find**👎, **Reuse** 👎)
- Potential compatibility nightmare (**Sustainability**👎)
- Reliability? e.g. nodejs' breakages (**Sustainability**👎)
- Silos (**Reuse** 👎, **Collaboration** 👎)
- Environmental impact???

Libre software is ubiquitous 🎉, but too often under the hood

- What about the applicative layer?
- What about fair and ethical services?
- What about fair and ethical AI models?

In particular for less tech-savvy audiences

2020's: The next challenges?

- From physical ubiquity to virtual ubiquity (**Environment** 👍, **Joy** 👎, **Community** ?)
- Internet anywhere, anytime
- Fluidity: local vs remote, compiled vs interpreted, gradual typing, multi-paradigms programming
- Growing digital Commons of tens of thousands of packages to compose from 👍
  Built by hundreeds of thousands of developers worldwide 👍
- Machine Learning for the masses

Will that scale?

- Complexity (**Find**👎, **Reuse** 👎)
- Potential compatibility nightmare (**Sustainability**👎)
- Reliability? e.g. nodejs' breakages (**Sustainability**👎)
- Silos (**Reuse** 👎, **Collaboration** 👎)
- Environmental impact???

Libre software is ubiquitous 🎉, but too often under the hood

- What about the applicative layer?
- What about fair and ethical services?
- What about fair and ethical AI models?

In particular for less tech-savvy audiences

Question: Impact of machine learning?
Copilots, natural language interaction, computations, …

Take home messages

Open Science and research software

- **A decades long joint history**; finally recognized by institutions!

Open Science and research software

- **A decades long joint history**; finally recognized by institutions!

- **Software raises very specific Open Science challenges** (it's not just another type of data) Notably:
  - Software is a social construction
  - Software is a living object ($\Longrightarrow$ ecosystems of software)
  - Software is complex, composed
  - …

A long track record of Open Science Best Practices for software

- **Findable:**
  Barriers: complexity: which function XXX of package YYY solves problem ZZZ?
  Levers: documentation, introspection, training, social networks, AI copilots …
- **Accessible:**
  Barriers: complexity, institutions, resources …
  Levers: virtual environments, public forges, package managers, repositories, archives, training, time, …
- **Interoperable:**
  Barriers: architecture, languages, systems, institutions, …
  Levers: source code, standards (e.g. webassembly), virtual environments, remote procedure calls (bind & adapt), semantic, commons, …
- **Reusable:**
  Modularity, quality, **build reproducibility**, **execution reproducibility**, **literate computing**, training, …

A long track record of Open Science Best Practices for software

- **Findable:**
  Barriers: complexity: which function XXX of package YYY solves problem ZZZ?
  Levers: documentation, introspection, training, social networks, AI copilots …
- **Accessible:**
  Barriers: complexity, institutions, resources …
  Levers: virtual environments, public forges, package managers, repositories, archives, training, time, …
- **Interoperable:**
  Barriers: architecture, languages, systems, institutions, …
  Levers: source code, standards (e.g. webassembly), virtual environments, remote procedure calls (bind & adapt), semantic, commons, …
- **Reusable:**
  Modularity, quality, **build reproducibility**, **execution reproducibility**, **literate computing**, training, …

Beyond FAIR: **Sustainable**, ARDC, … see e.g. Dicosmo's talk at Open Science Days@UGA, 2022

Open Science policies for research software

Open Science policies for research software

- Given appropriate means and training, scientists are in general sympathetic to Open Science, when not enthusiasts

Open Science policies for research software

- Given appropriate means and training, scientists are in general sympathetic to Open Science, when not enthusiasts

- Which best practices are relevant depends enormously on the piece of software

Open Science policies for research software

- Given appropriate means and training, scientists are in general sympathetic to Open Science, when not enthusiasts

- Which best practices are relevant depends enormously on the piece of software

Support and foster Open Science Best practices for Software

Don’t impose any of them

unless absolutely necessary to counter-balance other higher forces

Open Science policies for research software

- Given appropriate means and training, scientists are in general sympathetic to Open Science, when not enthusiasts

- Which best practices are relevant depends enormously on the piece of software

Support and foster Open Science Best practices for Software

Don’t impose any of them
unless absolutely necessary to counter-balance other higher forces

If in doubt, ask the Software Charter of the CoSO (Comité pour la Science Ouverte)

Research Software Engineers

- Research software development **by-users-for-users** can work very well
- However support from **Research Software Engineers** makes a huge difference:
  - train the community
  - give advice
  - tackle highly technical tasks
  - maintain
  - …

$\Longrightarrow$ A continuum between research software engineers and researchers

Research Software Engineers

- Research software development **by-users-for-users** can work very well
- However support from **Research Software Engineers** makes a huge difference:
    - train the community
    - give advice
    - tackle highly technical tasks
    - maintain
    - …

$\Longrightarrow$ A continuum between research software engineers and researchers

Recognize software development by all

Ease flexible access at all time scales to Research Software Engineers

Promote career paths for Research Software Engineers

Funding

Fund basic scientific software development and in particular *Software Maintenance*

Funding

Fund basic scientific software development and in particular *Software Maintenance*

Project based funding has its limit:

- Unpredictable
- Tension with career paths
- Huge overhead for the community

Funding

Fund basic scientific software development and in particular *Software Maintenance*

Project based funding has its limit:

- Unpredictable
- Tension with career paths
- Huge overhead for the community

Promote recurrent funding