Good practices in digital service ecodesign for software developers

Cyrille Bonamy, Cédric Boudinet, Laurent Bourgès, Karin Dassas, Laurent Lefèvre, Benjamin Ninassi, Francis Vivat

Good practices in digital service ecodesign for software developers

Authors :
Cyrille Bonamy : LEGI / CNRS
Cédric Boudinet : G2Elab / Grenoble INP
Laurent Bourgès : OSUG / CNRS
Karin Dassas : CESBIO / CNRS
Laurent Lefèvre : Inria / ENS Lyon
Benjamin Ninassi : Inria
Francis Vivat : LATMOS / CNRS

The authors are members of the CNRS EcoInfo Service Group [1.1] *which works on digital eco-responsibility.*

Summary:

This booklet is a complement to the 3 booklets of good practices related to software development proposed by the network of actors of software development within the Education and Research: DevLOG.

This project is dedicated to good practices in terms of eco-design of digital services that allow to apprehend, understand and reduce the environmental impact of digital technology.
macro
After explaining the general context in the first sheet, a second sheet ("**But Why?**") highlights the need to integrate an environmental dimension into our digital service designs, and consequently into our software developments. The third sheet ("**When?**") reminds us of the stages in the life cycle of a digital service and introduces the good practice sheets that correspond to the different stages: "**Before**", "**During**" and "**After**", bearing in mind that development is often iterative, and the boundaries between the different stages are permeable.

At the end of the booklet, you will find a specific sheet on eco-design practices for scientific computing, as well as sheets on development on mobile platforms, for the web and on hardware accelerators.

This booklet is a translation of the following French booklet :

Je code : les bonnes pratiques en éco-conception de service numérique à destination des développeurs de logiciels [1.2]

The existing DevLOG booklet (in French) :

Je code : les bonnes pratiques de développement logiciel [1.3]
Je code : les bonnes pratiques de diffusion [1.4]
Je code : quels sont mes droits et obligations ? [1.5]

A digital service is:

- information: data
- processing: algorithm, filtering, simulation
- information exchange
- user interfaces

A digital service relies on:

- software infrastructures: applications, tools, libraries, protocols
- hardware infrastructures: servers, network equipment, terminals, sensors
- people: developers, system and network administrators, project managers, researchers

The human impacts [2.1]

Through their daily lives, humans collaborating in the production of digital services will generate impacts: for example, they will use computer equipment and travel to their office or to meet. Setting up an organizational environmental policy to limit these impacts is crucial. For example, encouraging the extension of the lifespan of computers and the use of refurbished equipment will reduce the impact of manufacturing equipment. In general, try to contribute to the implementation of an environmental policy in your organization.

Example of a digital service: *the workflow of a digital simulation*.

It consists of the following components:

- pre-processing of input data
- transferring the input data to the relevant computational infrastructure (local machine, regional datacenter, GENCI [2.2], public cloud, GPU/CPU)
- calculation on the computational infrastructure
- transfer of data to the post-processing platform
- post-processing, storage and dissemination of the output data
- analysis and exploitation of the data

Many levers to reduce impacts, but a global vision is needed:

- limiting data transfers
- choice of computational and pre- and post-processing platform
- post-processing of data as close as possible to the place of creation
- limit input and/or output data
- choose external software components or redevelop them

Eco-designing a digital service means integrating environmental aspects throughout its life cycle [2.3] (be careful to define the functional unit [2.4])

Developers, but also project managers, engineers and researchers, everyone is concerned by this booklet!

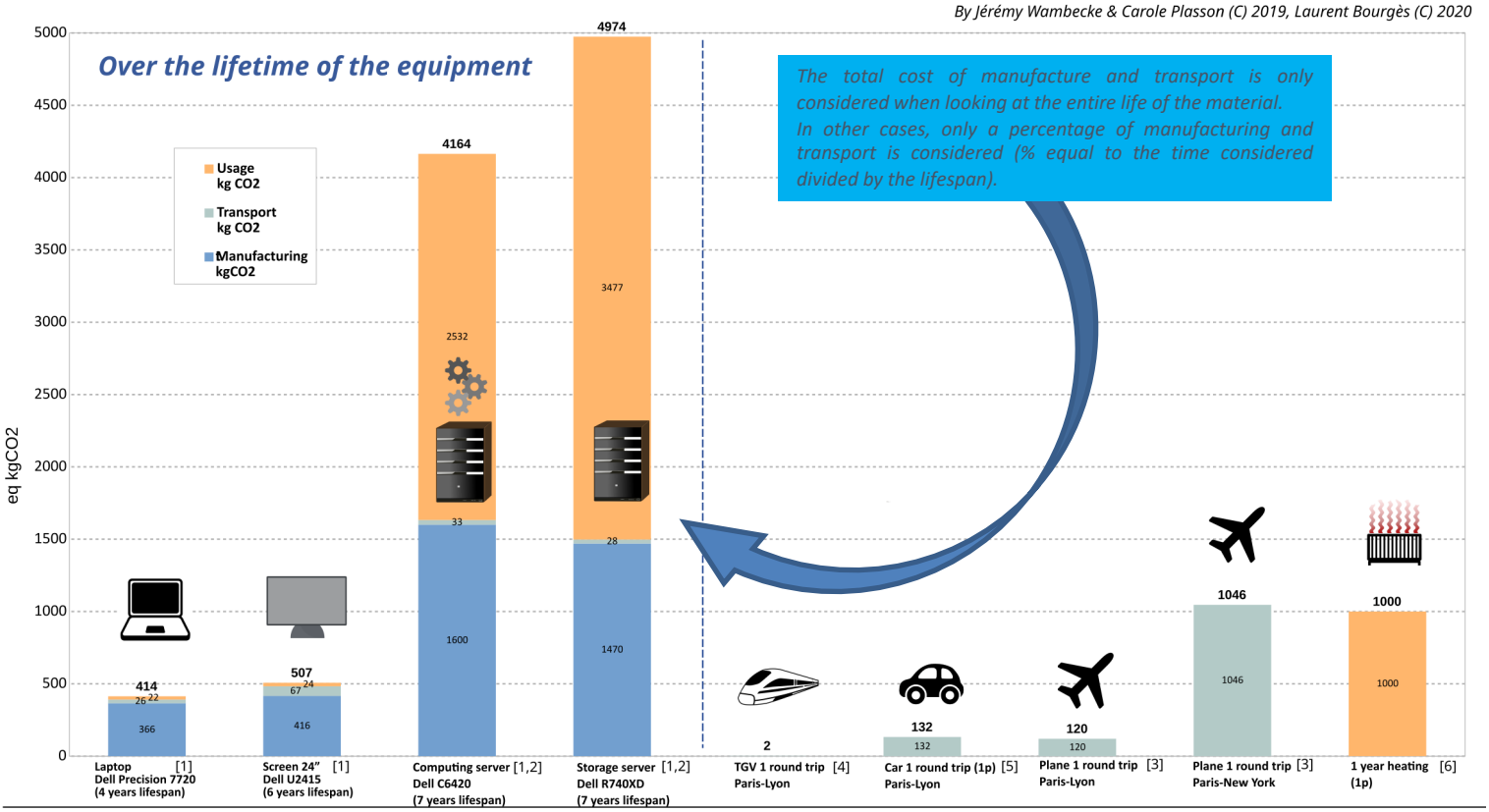
But why?

Why should we care about the environmental impact of digital technology? Simply because it is far from negligible. The increase in the contribution of digital technology to global GHG (greenhouse gas) emissions is exponential. It is time to do everything possible to stop this exponential growth.
Some references on the subject on page 18 (ARCEP [3.1], ADEME [3.2], EcoInfo [3.3], The Shift Project [3.4] , Green IT [3.5], ISIT [3.6], DINUM[3.7], Boavizta[3.8])
It should be noted that the impact of digital technology is not limited to its most publicized aspect, greenhouse gas emissions and global warming. The major impact is due to the manufacture of digital equipment and the treatment of waste: soil and groundwater pollution, reduction of water resources, and even impact on health and biodiversity.

However, the treatment of WEEE (Waste Electrical and Electronic Equipment) D3E [3.9] is outside the scope of this booklet.

CO2 emission comparison

By Jérémy Wambecke & Carole Plasson (C) 2019, Laurent Bourgès (C) 2020



[1] Dell Data (corrected usage for FR) : (https://www.dell.com/learn/us/en/uscorp1/corp-comm/environment_carbon_footprint_products)
[2] Usage from one node average consumption (Berthoud et al. 2020)= 275W (C6420), 375W (R740XD) (https://hal.archives-ouvertes.fr/hal-02549565)
[3] https://eco-calculateur.dta.aviation-civile.gouv.fr/
[4] https://ressources.data.sncf.com/explore/dataset/emission-c02-tgv/table/
[5] 473km with a car emitting 0,140 kg CO_2/km
[6] https://www.insee.fr/fr/statistiques/fichier/1281320/ip1445.pdf
Impact Factor: 0,108 kgCO_2/kWh (FR)

- the software is at the heart of the digital service and determines the hardware needed and its lifespan
- the manufacture of the hardware is not negligible from an environmental point of view
- less renewal of the hardware (which is strongly impacted by the software it operates) makes it possible to reduce the impact of the equipment manufacturing phase

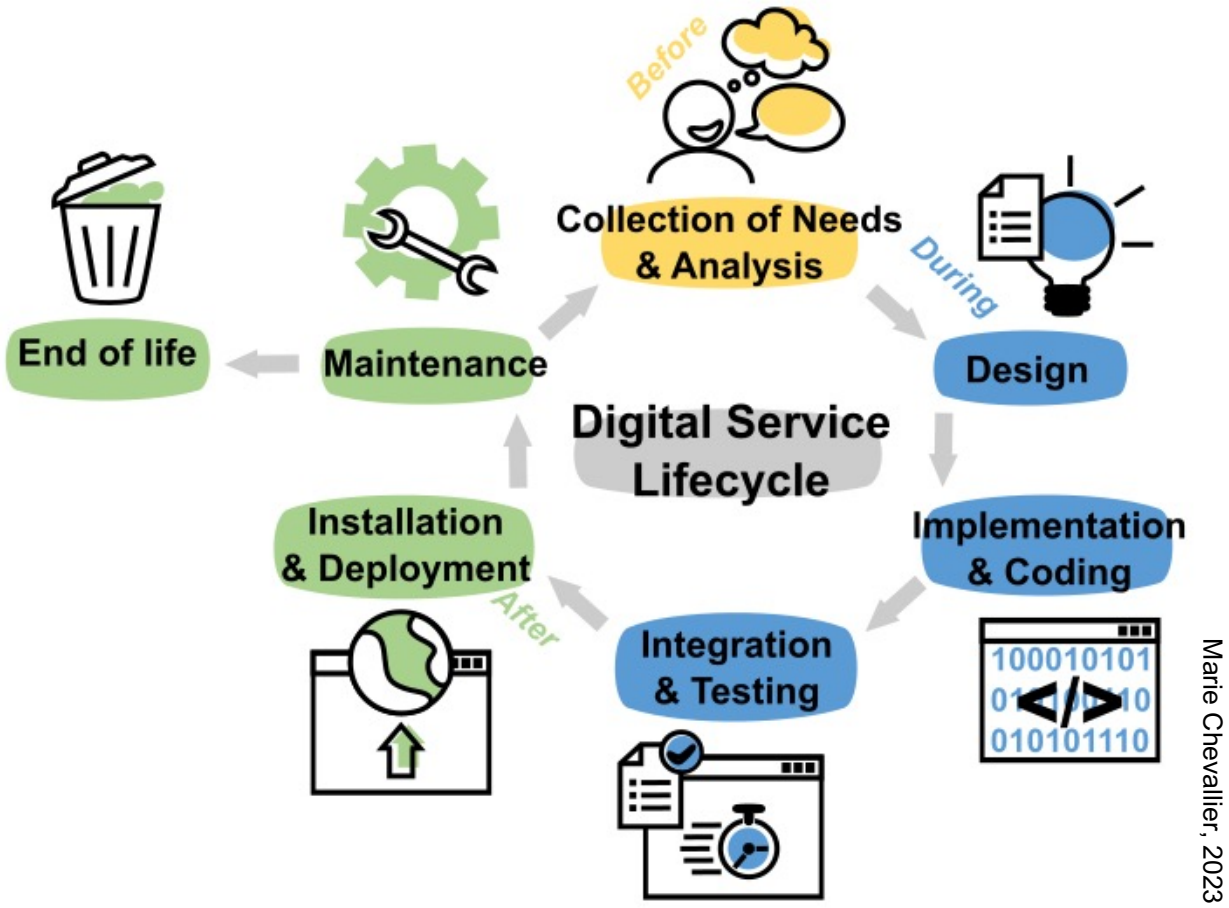
The efficiency or sobriety of the software has a strong influence on the need for computing power, memory and storage, thus reducing the volume of hardware needed for the digital service, thus reducing the impact of the manufacturing phase of the equipment and the energy consumption during operation to a lesser extent.

Example of digital impact in the research community: 5 million hours.calculation core (approx. 25 tons of CO2eq [3.10]) are equivalent to 25 round trips from Paris to New York by plane according to the French civil aviation authorities [3.11] or 11 according to Ecolab [3.12].

When?

The eco-design of a digital service consists of integrating environmental constraints into all development processes, in order to **reduce the environmental impacts of the digital service during its life cycle**. Impacts can occur at any stage of the service's life cycle: before its development (definition of needs, analysis), during its development (design, software development, testing, production) and after its development (operation, maintenance, end of life). Generally speaking, the earlier in the life cycle that environmental aspects are taken into account, the greater the effect. This booklet proposes to explore different good practices in digital service ecodesign, with a particular focus on software, and the elements to be taken into account before, during and after the software development phase.

It should be noted that the good practices proposed are not dependent on the development cycle chosen: V-cycle, W-cycle, spiral, waterfall, agile model. An iterative model can reduce the impact of a stage during a new iteration.



Taking action to reduce environmental impact is possible at every step

Follow the colour code!

Before | **During** | **After**

Rely on the main principles of digital service eco-design:

- **simplicity** : simplify the software to avoid white elephant
  - in terms of functionality: 70% of the features requested by users are never or rarely used (Standish Group, 2006)
  - in terms of user interfaces
- **frugality and sobriety**: limit the number and size of elements (images for example). For example, in a web development, avoid "fat" pages in terms of functionality and graphics.
- **suitability**: usefulness (the result must meet the user's expectation) x speed (response time for the user) x accessibility (e.g. for certain disabilities).
- **durability**: reusing all or part of a software product avoids duplicating developments; contributing for the benefit of the community.

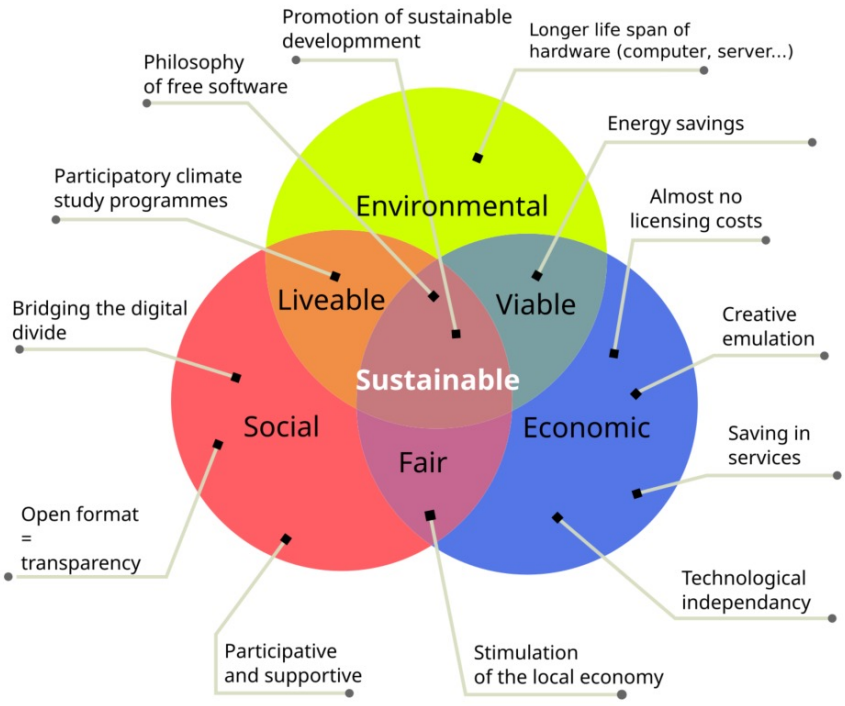
I control the number of software features: avoid obesity

Too many disparate features: What happened to the function I used to use most often? My software has become obsolete! My digital service has become a real « white elephant » !

More features than necessary : my old infrastructure is no longer sufficient, I have to change it !

Justified and useful features : my digital service is light and efficient !

I further free software: reusing components and contributing to the commons



CC-BY-SA : ll-dd.ch [5.1]

The compatibility between sustainable development and free software is illustrated in figure [5.1] with some examples

Using and producing free software allow avoiding redundant development (with useless energy consomption)

Check the **4 essential freedoms** which define the free software according to Richard Stallman : [5,2] free software !

I plan the management of the software: increasing the lifespan

A Software Management Plan (SMP) is a tool for the perpetuation of software. Information about the software, its context, its characteristics, or the organisation of its distribution is thus grouped together, updated during the life cycle, and makes it possible to modify and reuse it easily. Opidor [5.3], Presoft [5.4] (see also Data Management Plan [5.5] Slide 7)

Take the time to think about the targeted deployment platform while considering the constraints of the complete digital service. Do not forget to consider the security aspects, even before the development.

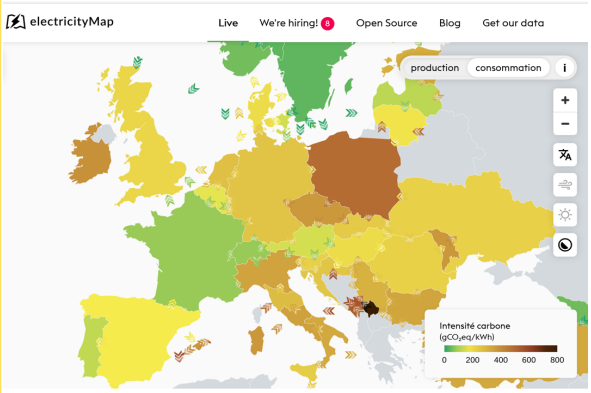
I think about the deployment of the service: adapting to the context

Depending on the characteristics (location, capacities...) of the available platforms:

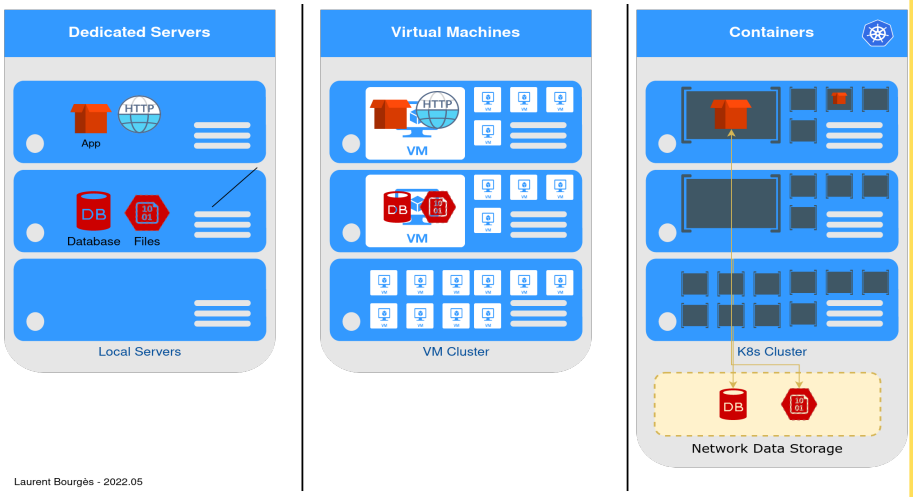
- computer, local server, embedded platform (microcontroller, Raspberry-Pi...)
- specialized platform: virtualization or specialized HPC cluster, GPU, FPGA, ARM64
- a Service Offer of site or of employer (CNRS, Universities) / public clouds

According to the constraints of the service :

- supported languages, specialized communications (Infiniband, OmniPath, SpaceWire, Serial...)
- bottlenecks: disk access, network, memory transfers
- durability, portability, security, long-term costs, maintenance, wall time
- geographical location of the uses



Electricity map / Carbone Intensity [6,1]



Laurent Bourgès - 2022.05

Some recommendations:

- I rely on shared services (authentication, storage, database...)
- I put the data on distributed and backed up storage spaces to avoid duplication.
- I favor virtualization and I reduce the size of the images (VM, container): see diagram on the right above
- I estimate as closely as possible the necessary resources (vcpu, memory, storage) to avoid oversizing
- I automate the deployment of physical and virtual machines to ensure reproducibility

I don't forget the security aspects of my service

From the unavailability of my service to the theft of my users' personal data and the installation of malicious software on my infrastructure, the consequences of a security breach can be numerous. The application and/or the associated data can become obsolete.

Security must be a concern at all stages of the life cycle of a digital service, and listing the best practices would require a separate brochure. I can start by drawing inspiration from the recommendations of the French National Agency for Information Systems Security (ANSSI-C [6.2], ANSSI-Rust [6.3], ANSSI-Java [6.4]).

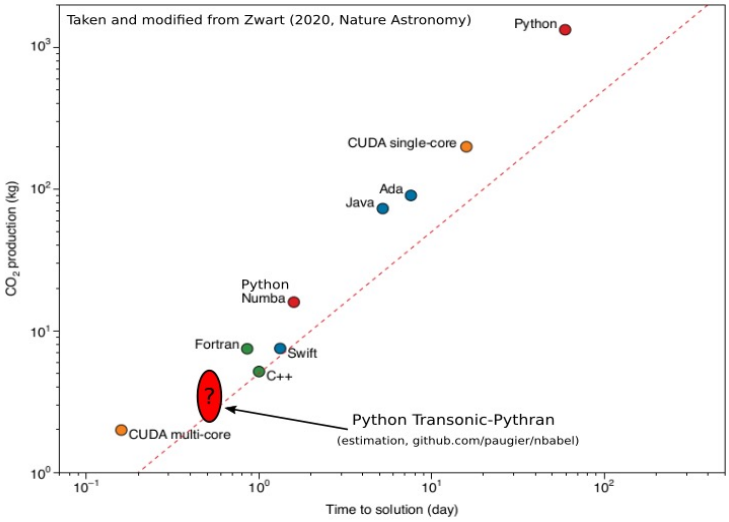
Nevertheless, I remain vigilant about the impact of the security mechanisms considered, particularly on the increase in resource consumption or the number of devices: take into account the criticality of the services so as not to over-secure unnecessarily.

Take the time to choose a language by considering energy efficiency, sustainability, simplicity and more generally the global constraints of the complete digital service.

I choose my language and/or software stack: it's all about compromise

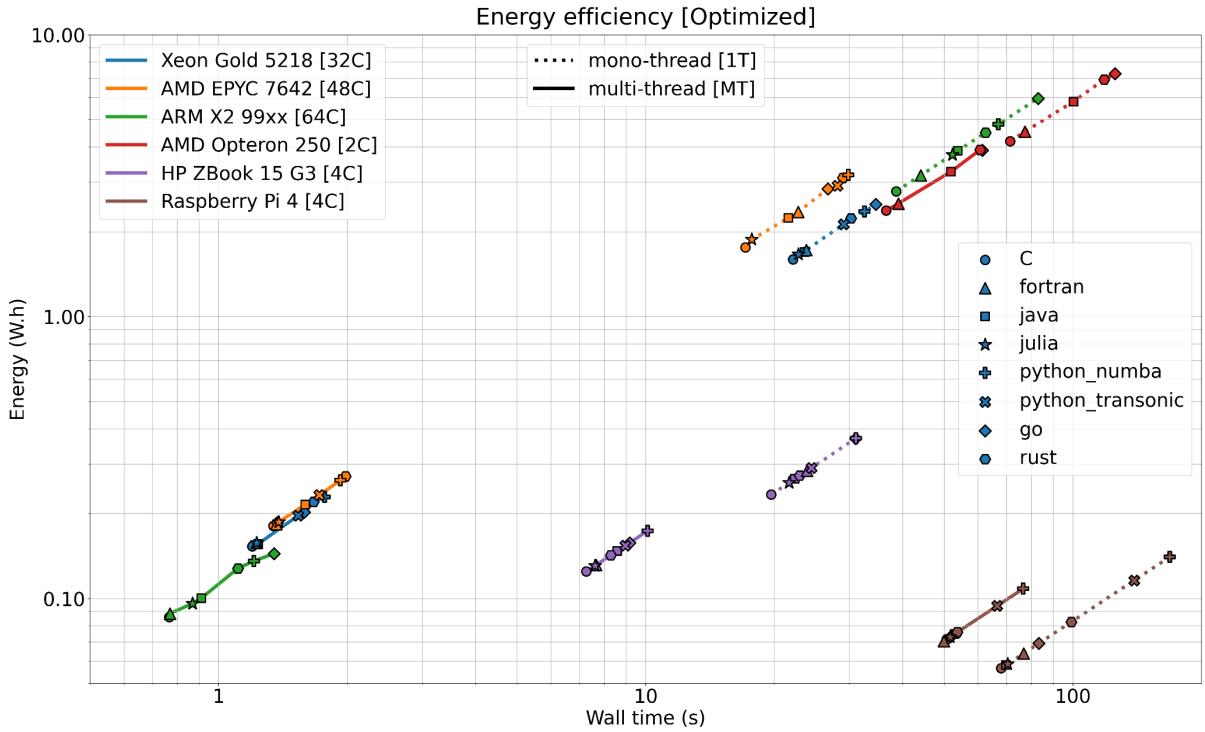
Illustrations of energy efficiency in relation to language:

- *left, Simon P. Zwart, Nature Astronomy [7.1] with a simulation N-Body, figure updated by P. Augier [7.2] et [7.3] highlighting the significance of Python compilers and human factors (code optimisation, expertise and time spent optimising, testing and measuring codes),*
- *right, Rui Pereira et al, SLE17 [7.4] [7.5] based on the Computer Langage Benchmark Game.*



Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Swift • PHP	Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
F# • Racket • Hack • Python	Lisp • Ocaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
JavaScript • Ruby	Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Dart • TypeScript • Erlang	Swift	TypeScript	Lua • JRuby • Perl
JRuby • Perl	Dart • F#	Erlang • Lua • Perl	
Lua	JavaScript	JRuby	
	Racket		
	TypeScript • Hack		
	PHP		
	Erlang		
	Lua • JRuby		
	Ruby		

The use of Python illustrates the largest gap (~ 100x) in efficiency depending on the execution environment (cpython, numba, pythran, pypy) and the optimisation of the code in this use case (performance-driven development).



In the above study (C. Bonamy, L. Bourgès, L. Lefèvre [7.6]), ARM-based machines are really efficient whatever the language considered, and without "porting" as the use of GPUs would require. We also see that two groups of languages stand out in terms of efficiency: in the lead, C, Fortran, Java and Julia, and slightly behind Python, Go and Rust

- Compiled languages (native or usually interpreted but optimised, e.g. using Numba or PyThran for Python) to be preferred for heavy, high performance or real time processing
- easily accessible (interpreted) languages to be preferred for less constrained processing, in order to facilitate maintenance, re-use and thus sustainability
- not all components need to be written in the same language

Anticipate and take into account data to reduce their environmental impact. Take into account the volume and costs of data transfer. Promote interoperability and open data.

I pay attention to the volume of data: digital sobriety

Data storage: ASCII or binary formats ??

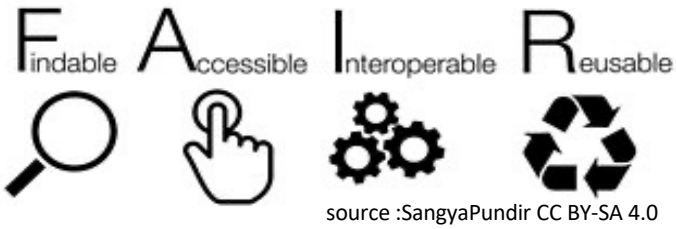
The text format (raw ASCII, XML, YAML...) generally requires more space to store the same information. However, when versioning files, this principle may be reversed (successive storage vs. differential storage). There is a risk of obsolescence when using non-open binary formats..

High Volume ?

- I favor processing as close as possible to the data (server) to avoid transferring the data sets to the users' workstations (and needing other powerful machines)
- I take into account the volume of data handled when designing processes and avoid large transfers between software layers
- I minimize the size of the software package (archive) and eliminate unnecessary dependencies.

I plan data management: sustainability, reduction of redundant developments

Following the FAIR principle reduces the environmental footprint for several reasons:



source :SangyaPundir CC BY-SA 4.0

- **Interoperability**: data should be easy to combine with other data sets, both by humans and by computer systems. Choosing to make data interoperable avoids the need to multiply data with different formats: reducing the number of data and the cost of conversion
- **Open Data / Reproductibility**: making data and source codes easily accessible (permanent DOI reference) and reproducible reduces the environmental footprint by avoiding duplication. Less storage (use of web-services), less redundant development.

Many resources exist: FAIR principle [8.1] / european directive [8.2] / European Open Science Cloud [8.3] / Research Data Alliance [8.4] or this interactive FAIR infographic [8.5] proposed by URFIST Méditerranée.

The often cited objectives of a research data management plan (DMP) [8.6] are to improve the impact of research projects and their scientific contribution, and even to meet the requirements of funders. Another objective of the plan is environmental: a properly done plan leads to FAIR data. Some examples of DMPs on the OPIDoR website [8.7], see also Datactivist's sprint PGD [8,8].

The GDPR (General Data Protection Regulation) guide [8.9] sets out a strict framework on the collection of personal data at a minimum, as well as their shortest retention period.

I am also thinking of the FAIR principles for software: FAIR4RS [8.10]

Analyze and monitor my digital service to identify the most consuming software components and functions.

I analyze my code: identify *a priori*

- Static analyses allow, among other things, to determine the difficulty of maintaining software (example tools: Sonarqube, codacy)
- dynamic analyses make it possible to quantify the use of resources (example tools: gcov, gprof, perf, valgrind)

These software analyses - part of standard good practice - are essential to reduce the impact of a digital service.

I measure performance: identify in operation

- loading time
- execution time
- power consumption
- consumption of network protocols (e.g. in the IOT)
- data size
- number of requests
- web performance
- measurement of carbon emissions

Examples of tools: Firefox web tools (trafic, queries, JavaScript), Apache JMeter (web scenarios), ecoindex (web), integrated profiler adapted to the language, CodeCarbon, Green Algorithms, KDE eco)

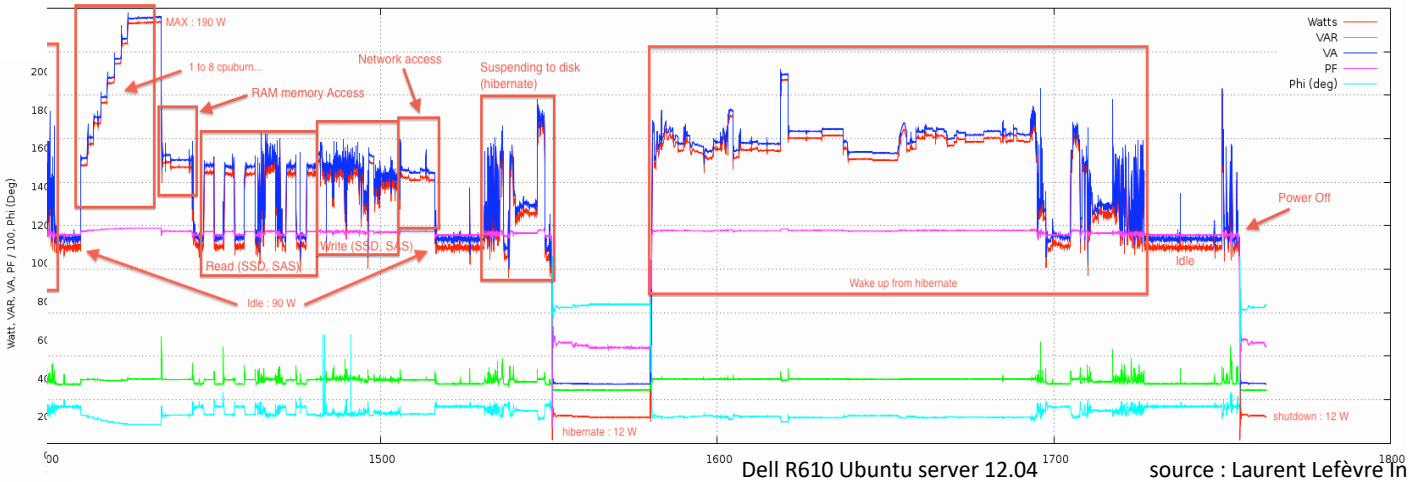
Focus: I profile the energy consumption of software

Each use of resources (processor, memory, storage space, network) in my software causes power consumption.

To measure the energy impact of software in use :
need hardware equipment (PDU, power meters) or software probes (powerAPI, warning to idle calibration).

Wattmeter Watts'up Pro - PDU Eaton

I profile the power consumption over time, to discover the different intensive phases of my software.



Dell R610 Ubuntu server 12.04

source : Laurent Lefèvre Inria

Optimizing your software, such as implementing efficient algorithms and adequate data structures, reduces execution time, which generally leads to a reduction in electricity consumption, and therefore a reduction in eqCO2 emissions. But beware of the rebound effect, and the effects on other elements of the digital service.
Similarly, you have to be vigilant when applying the usual best practices in software engineering (version management, continuous integration). They must be applied, but with caution (yes, but....)).

I optimize my code (yes but...)

Beware of the rebound effect:

Optimizing a software can lead to launching more operations or processing more data, so the ecological footprint of the service will not be reduced. The actual resource savings are negative because usage increased beyond potential savings – the rebound effect [10.1] is higher than 100% (Jevons paradox).

Optimization should simply serve to reduce the energy and resources consumption, and if possible to reach the target more quickly. Every run has an impact!
It is essential to optimize only what has the most impact (Pareto's law).

Moreover, the quality of the tests should not be neglected in the optimization process (regression tests).

I improve my code (yes but...)

Code improvement is a multifactorial problem, such as architecture, CPU, memory, storage and network use. Improving the use of any resource can have some drawbacks on the use of another resource.

For example:

- Using a cache system my speed-up some computations, but it may have some impacts on memory use, storage software complexity
- Using compression to reduce network traffic increases CPU use
- Improving security may increase energy consumption which raises with security algorithm complexity. Some algorithm are already available in the instruction set of some processors (e.g. AES), one should use it. A lack of security may generate additional costs (increased maintenance, recommissioning...) [10.2]

Version control (yes but...)

I use a version control system , but:

- I avoid (or restrict to the minimum) to manage binary packages and optional datasets
- I do not manage compilation products and output files with such a system

Continuous integration (yes but...)

- I ponder my Continuous Integration process. I opt for a minimal sized docker and I activate my CI only for needed branches . It is also possible to schedule the CI process. Thus the processing and output file are not generated for each modification
- I monitor the jobs duration, their number, artefacts size, network use
- I opt for shared forges

The eco-design of a digital service must take into account the impacts linked to deployment and production. Continuous improvement makes it possible to reduce environmental impacts.

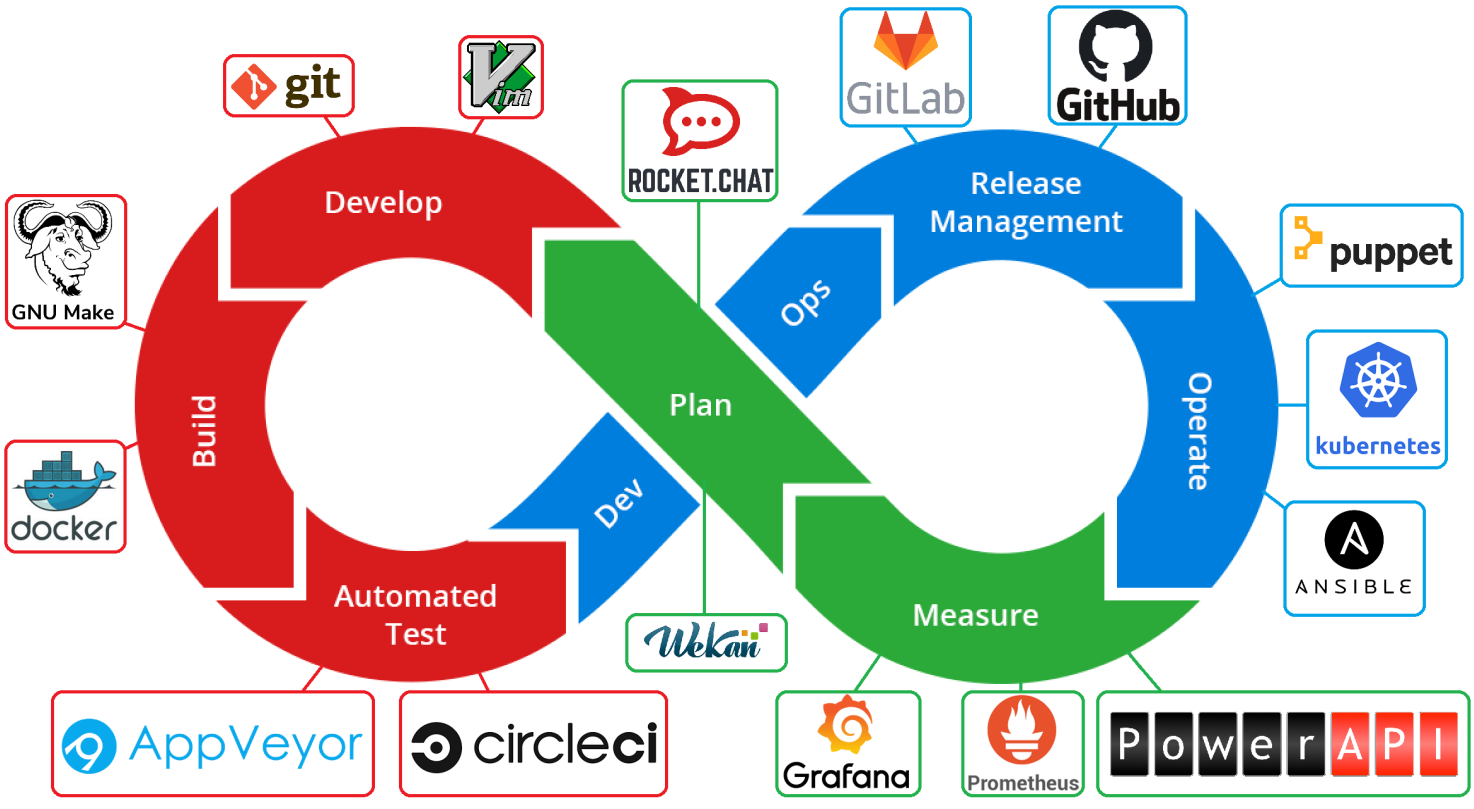
Deployment: digital sobriety

- I prefer shared hosting, with a COC label (Code Of Conduct), as close as possible to the data and users
- I favour virtualisation (fewer physical servers), minimising the memory and disk footprint (size of virtual machines, containers), except in special cases (high-performance computing)
- I pay attention to certain rebound effects: I avoid the multiplication of virtual machines, service instances

Production: continuous improvement according to use

- I use supervision (and alerts) to observe CPU peaks, resources used (disk, network), power consumption
- I modify the digital service to adapt it to the observed usage (continuous improvement)
- I reduce the frequency and volume of backups, I adopt deduplication

Examples of monitoring tools : top, vmstat, zabbix, scalasca, nagios, prometheus, grafana



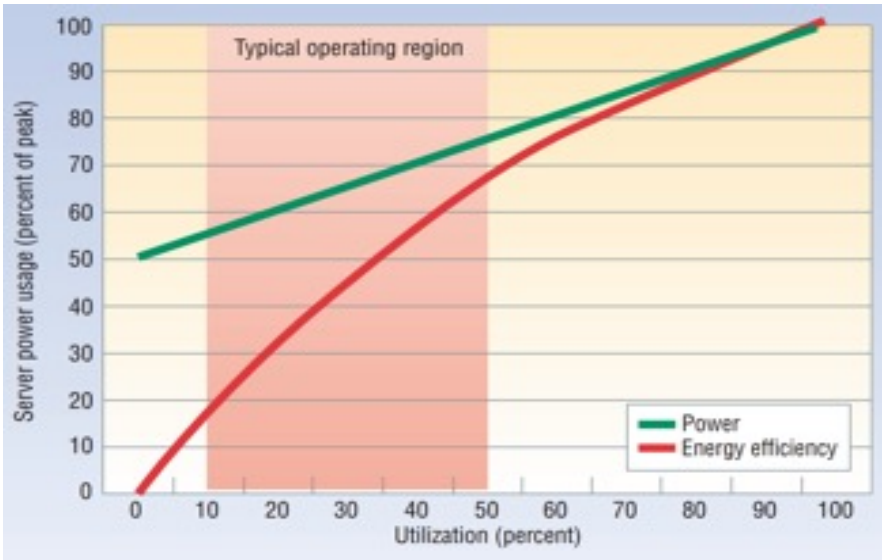
Examples of tools used for the continuous improvement of the digital service
(source : PNGEgg, adapted by C. Bonamy)

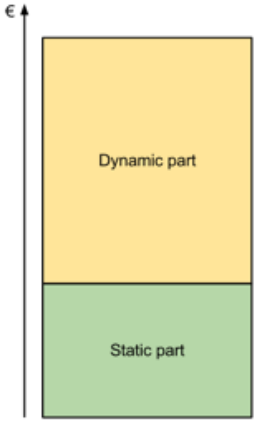
The eco-design of a digital service must take into account the impacts of the distribution, use and administration of the software, as well as the needs of the users. It must be ensured that energy efficiency and reduced environmental impacts are maintained as the software evolves.

I track down unnecessary resource consumption : avoiding energy waste

Static power consumption of machines is significant: I turn off the machines if possible

The software influences the dynamic consumption of the machines, but there is little energy proportionality (an unoccupied server can consume 50% of its power budget. The static power consumption of a server or network equipment is important





Luiz André Barroso and Urs Hölzle, "The case for Energy-Proportional Computing", IEEE Computer, 2007 [12.1]

Implementation of standby systems

- I turn off virtual machines that have a memory and CPU footprint
- I limit the number of services deployed
- I encourage the shutdown of unused hosts and resources (computing cores, storage systems)

I distribute and maintain my code: promoting sustainability and simplicity

Diffusion

- I deliver the software through a single, easily accessible web-based collaborative software platform (GitLab…)
- I prefer open source publications. All about licenses and procedure: DevLOG license brochure [12.2] and licenses on etalab [12.3]

Software updates management

- I reduce the size of software products
- I rationalize their number and frequency
- I care for backwards compatibility as much as possible
- I avoid constant addition of features
- I clearly define the updates workflow
- Long Term Support [12.4] for corrective updates

I raise awareness : improving practices

- using existing resources:
  - Ecoinfo [12.5]
  - ADEME [12.6]
  - principles green [12.7]
  - Mooc Environmental impacts of digital technologies [12.8]
  - The digital collage [12.9]
- informing the users of my digital service on the environmental impacts of their use:
  - displaying consumption monitoring information
  - making available the Life Cycle Analysis (LCA) of the digital service

After (end of life)

A digital service that is no longer used will continue to generate residual environmental impacts due to, for example, the storage of its data, its hosting, regular network access to other digital services necessary for its operation, etc. It is therefore important to deal with its end-of-life, like any other product.

I archive: promoting sustainability

Archiving

- declaration of source code to Software Heritage [13.1] to facilitate its reuse
- I save essential (= valuable or legally binding) data on cold storage: execution traces, specific data, documentation, etc.

I decommission: avoid "zombie" software and data

The users

- I warn all users and make sure that the end of life will not have undesirable side effects: retreat to more energy-intensive solutions, implementation of alternative processes with greater environmental impacts, etc.

Documentation

- I delete all the data concerning the software ... and I indicate the end of life of the software in the documentary space that was dedicated to it.

Support tools

- I update the configurations of the tools used for support and upgrades (helpdesks, version tracking tools, etc.) in order to remove any unnecessary active file and to clean up the data concerning the stopped digital service.

The software

- I remove the installation of the software on all instances where it has been installed (development/qualification/production environments, workstations if applicable).

The data

- I delete the data stored after archiving: databases and flat files.

Necessary services

- I stop and uninstall application services that were only useful for running the software (Apache, Nginx, mysql, etc.).

Virtual machines and containers

- I stop containers and virtual machines that were not shared with other services, and remove their images. Shutting them down can shut down part of the physical infrastructure.

Backups

- I delete all daily/weekly/monthly backups required for the disaster recovery plan.

Monitoring

- I delete the references to the software in the configurations of the supervision tools as well as all its execution traces.

Shared dependencies

- I check that any shared dependencies (software libraries, JS or CSS assets, etc.) are still useful to at least one other digital service, otherwise I delete them too.

Application mapping

- I delete the part concerning the software at the end of its life in the team's or institution's application mapping in order to keep the urbanisation of the information system representative of reality.

I eco-design scientific code

Before developing:

- I choose the license (if possible open to facilitate re-use and reproducibility)
- I choose the target technology and language according to the constraints (C++, Python, Fortran, GPU, classic cluster, ARM, Cloud, Web services)
- I choose the appropriate infrastructure for each phase (lab for development, regional datacenter for development and validation, national computing center for production Resources ESR [14.1])
- I think about the management of the data associated with my code; I establish a Data Management Plan
- I design the entire workflow (input data, processing, post-processing, storage) with environmental impacts in mind (cost of network transfers, etc.)
- I alert partners and other developers to the environmental impacts of digital technology (using existing entities such as EcoInfo)

At the very beginning of the development:

- I create a repository (git via GitHub for example)
- I create the files "AUTHORS", "LICENSE", "README »
- I consider open and efficient formats for the software output
- I set up continuous integration (Gitlab pipeline, CircleCI...), without unnecessary excess
- I plan the creation of the documentation and its update (Sphinx, Doxygen...)
- I consider possible crashes and plan for the possibility of restarting my calculations

During development:

- I choose the right libraries (MPI [14.2], OpenMP [14.3])
- I release the sources, share the repository, as soon as possible
- I ensure the reproducibility of my digital experiments

Afterwards, during the production phases:

Beware of the rebound effect

- I look for hot spots, using existing tools (gprof, valgrind...)
- I pay attention to the weight of the I/O (input/output)
- I analyze the scalability of my code (ideally on the end-use infrastructure) (Amdahl law [14.4])
- I get help to optimize (example : Performance Optimisation and Productivity [14.5]) while paying attention to the rebound effect (always more!)
- I make my code users aware of the environmental impacts of their calculations
- I am looking for the best compromise between wall time, accuracy of calculation and eqCO2 cost
- I think about the real interest of my calculations and I do not hesitate to redevelop if necessary
- I think about deleting data when it is no longer useful

After, end of life:

- I delete non-essential data; I keep only the minimum for reproducibility
- I release my code (through software heritage [14.6]) indicating that the software is no longer maintained

I do not think: "it's a little code, it's only for me, these good practices are not for me".

I eco-design code on GPU and/or AI

Before developing:

- I pay attention to the learning burden for Artificial Intelligence
- I do not forget to consider alternatives to AI in my choice of algorithm
- I make users aware of the cost of calculations, and in particular the overall cost, which includes the learning phase, see paper on AI power consumptionl [15.1], and on environmental assessment of projects involving AI methods [15.2]
- I choose the technology and associated constraints according to my needs (CUDA vs OpenCL: performance and simplicity vs genericity and verbosity)

During development:

- I choose the right frameworks (OpenACC [15.3], OpenCL [15.4])
- I don't redevelop what already exists and I use proven high level libraries (Tensorflow, PyTorch, scikit-learn)
- the key to accelerators is to think about standardizing (no conditions (if/else), no heterogeneity between cores) and maximizing processing on the accelerator (GPU) in order to avoid CPU to GPU memory transfers (be careful with CPU/GPU memory transfers vs kernel time)
- I make portable, high-performance code (beware of too many specificities with one type of accelerator, such as CUDA)
- I am careful about the impact of I/O, which can strongly depend on the target infrastructure (workstation vs. dedicated cluster)

Afterwards, during the production phases:

- I make users aware of the environmental cost of computing, and in particular the consolidated cost, which includes the costs associated with the manufacture of GPUs.

Some tools

Software probes:

- Experiment impact tracker [15.5] (Henderson et al.) : a python library that calculates power consumption from GPU, CPU and DRAM information.
- Carbon tracker [15.6] (Anthony et al.) : also a python library that calculates power consumption from GPU, CPU and DRAM information. In the case of AI computations, it allows to stop the training of an experiment when a fixed environmental cost is exceeded.
- Code Carbon [15.7] : a third python library based on the same principle as the previous ones.

Online estimation of energy consumption and carbon footprint associated with a digital experience:

- Green Algorithms [15.7] (Lannelongue et al.)
- ML CO2 Impact [15.8] (Lacoste et al.)

Web site eco-development

Before developing:

I make myself familiar with the Handbook of sustainable design of digital services [16.1] and I share it to the other project members, especially to the product owners and UX/UI representatives. I also invite designers to make themselves familiar with the Guide [16.2] written by ethical designers. Website eco-design is not an individual approach, it requires mobilizing the entire team involved in the project.
Indeed, the most effective gains on web development are made by :

- saving features, therefore by involving clients and final users to got to the essential
- simplifying user interfaces as much as possible, in a spirit of sobriety
- limiting the elements that generates traffic or CPU load on the terminal such as videos, plotters, datasets, graphics etc.

In general, I eliminate non-essential content or functionality from specifications. The service should be usable on a small screen, and if possible responsive. I prohibit the use of any type of Dark Pattern [16.3].
When I start development, I will take a look at the site 115 good practices [16.4].
I am concerned about security at all stages of the life cycle, and I familiarize myself with the recommendations of the ANSSI cybersecurity website [16.5].

During development:

I manage the quality based on environmental impact indicators on the terminals. To do this, I use a measuring tool if possible.(such as Greenspector [16.6]*) or I try to assess the environmental impact of my site (GreenIT Analysis [16.7], Ecometer [16.8], Fruggr [16.9]* or Greenframe [16.10]*). If possible, I add the call to this type of tools in my continuous integration chain, in addition to the development tools integrated into my browser that I use on a daily basis.

I plan the management of data as close as possible to the need, and I apply the F.A.I.R [16.11] principle.
I anticipate compatibility and maintainability over the long term. I favor long-lasting technologies maintained by active and established communities, to avoid the obsolescence of the product code. I pay attention to unnecessary dependencies of the application or frameworks used.
I limit network access and data transfers to the strict minimum: I thus make my service accessible for low-speed connections, and ideally resilient to disconnections.
I write code that is simple to understand and implement, it will also be more durable and easier to evolve. This is the basis of the KISS [16.12] principle: Keep It Simple and Stupid.

Afterwards, during the production phases:

I choose a Code of Conduct [16.13] certified host, that presents the best power usage effectiveness (PUE) and as close as possible to my final users.
I put in place a sober data backup policy and I supervise the service (CPU, IO) and the logs (maximum 1 year).
I anticipate uses: I am vigilant about the transformation of behaviors potentially induced by my application, and I adapt it in order to reduce their negative impacts.

*(Partially) commercial tools

How-to: Ecodesign for embedded and IoT devices

- put the device in Deep-Sleep (*) mode whenever possible
- Fractionnal numbers representation has a direct impact on performances and therefore on power consumption. Choose the most fitted representation (fixed or floating point) and data size allows to slow down the CPU. It is thus possible to downsize the processor.
- Smart objects generate a lot of data over internet. For the same user data exchanged different protocols add a various amount of data (ex. MQTT, HTTP, XMPP)
- Software design (interruptions vs. polling, use of circular buffers, optimization level) has a direct impact on processor load

(*) Deep-Sleep : special mode of operation that enables to slow down and reduce the power consumption. The processor can be woken-up thanks to an interrupt

How to ensure a mobile app life expectancy?

Aforementioned principles are also true for mobile application development. It is important to keep in mind that one should always better avoid a software overlay dependency on the end-user terminal. Indeed such a dependency might imply a software obsolescence.
Then, it is better to favor web based application over movile application
Digital service compatibility is a long term topic. An application should stay usable on old or exotic devices.

How to ensure resilience by securing ?

From the unavailability of my service to the theft of my users' personal data and the installation of malicious software on my infrastructure, the consequences of a security breach can be numerous. Corrective actions, when possible, are very costly in time and energy, and therefore have an impact on the environment, all the more so if the application is rendered obsolete due to a flaw.
Security must be a concern present at each step of the life cycle of a digital service, and listing good practices would require a brochure in its own right. I can start by taking inspiration from the recommendations of the National Agency for Information Systems Security (ANSSI [17.1]).

Nevertheless, one should remain vigilant on the impact of the security mechanisms implemented, in particular on the increase in the consumption of resources or the number of equipment: carefully take into account the criticality of the systems so as not to over-secure unnecessarily.

To the extreme

There are particularly energy-efficient technologies, "low-techs" taken to the extreme:

CollapseOS [17.2] : an operating system compatible with microcontrollers
Lowtech Magazine [17.3] : a website host that is only available when the solar energy production is sufficient

References

[1.1] https://ecoinfo.cnrs.fr
[1.2] Je code : les bonnes pratiques en éco-conception de service numérique à destination des développeurs de logiciels
[1.3] https://hal.archives-ouvertes.fr/hal-02083801/file/20191202_plaquette_developpement_V1.1.pdf
[1.4] https://hal.archives-ouvertes.fr/hal-02400300/file/20191202_plaquette_diffusion_V1.1.pdf
[1.5] https://hal.archives-ouvertes.fr/hal-02399517/file/20191202_plaquette_pi_licences_V1.1.pdf

[2.1] https://learninglab.gitlabpages.inria.fr/mooc-impacts-num/mooc-impacts-num-ressources/en/Partie3/FichesConcept/FC3.4.2-bonnespratiques-MoocImpactNum.html
[2.2] https://www.genci.fr
[2.3] https://www.eco-conception.fr/static/analyse-du-cycle-de-vie-acv.html
[2.4] https://www.eco-conception.fr/static/fonction-unite-fonctionnelle-acv.html

[3.1] https://www.arcep.fr/uploads/tx_gspublication/reseaux-du-futur-empreinte-carbone-numerique-juillet2019.pdf
[3.2] https://www.institutparisregion.fr/fileadmin/NewEtudes/Etude_1806/Full_report_ENERNUM_MAY_2019-eng.pdf
[3.3] https://ecoinfo.cnrs.fr/2022/07/05/carbon-neutralities-of-ict-companies/
[3.4] https://theshiftproject.org/en/article/implementing-digital-sufficiency/
[3.5] https://www.greenit.fr/empreinte-environnementale-du-numerique-mondial/
[3.6] https://isit-europe.org/
[3.7] https://ecoresponsable.numerique.gouv.fr/a-propos/
[3.8] https://boavizta.org/en
[3.9] https://ecoinfo.cnrs.fr/wp-content/uploads/2019/09/Informatique-et-développement-soutenable-LIMSI.pdf
[3.10] https://hal.archives-ouvertes.fr/hal-02549565
[3.11] https://eco-calculateur.dta.aviation-civile.gouv.fr/
[3.12] https://ecolab.ademe.fr/apps/transport

[5.1] https://ll-dd.ch/logiciels-durables/#convergence
[5.2] https://www.gnu.org/philosophy/free-sw.html.en
[5.3] https://dmp.opidor.fr
[5.4] http://www.france-grilles.fr/wp-content/uploads/2018/04/ModeleSMP_PRESOFTV3.2.pdf
[5.5] https://hal.archives-ouvertes.fr/hal-01241196

[6.1] https://app.electricitymap.org/map
[6.1] https://www.ssi.gouv.fr/entreprise/guide/regles-de-programmation-pour-le-developpement-securise-de-logiciels-en-langage-c/
[6.2] https://www.ssi.gouv.fr/guide/regles-de-programmation-pour-le-developpement-dapplications-securisees-en-rust
[6.3] https://www.ssi.gouv.fr/agence/publication/securite-et-langage-java

[7.1] https://arxiv.org/pdf/2009.11295.pdf
[7.2] https://rdcu.be/ciO0J
[7.3] https://github.com/paugier/nbabel
[7.4] https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf
[7.5] https://github.com/greensoftwarelab/Energy-Languages
[7.6] https://hal.archives-ouvertes.fr/hal-03607468

[8.1] https://www.force11.org/group/fairgroup/fairprinciples
[8.2] https://op.europa.eu/en/publication-detail/-/publication/7769a148-f1f6-11e8-9982-01aa75ed71a1/language-en
[8.3] https://www.ouvrirlascience.fr/portail-web-de-leosc/
[8.4] https://www.rd-alliance.org/fair
[8,5] https://view.genial.ly/5d64fbbd8352350fa3d22603
[8.6] https://hal.archives-ouvertes.fr/hal-01241196
[8.7] https://dmp.opidor.fr/
[8.8] http://opendatacanvas.org/sprint-pgd
[8.9] https://www.cnil.fr/fr/comprendre-le-rgpd
[8.10] https://www.rd-alliance.org/groups/fair-research-software-fair4rs-wg

[9.1] https://hal.archives-ouvertes.fr/hal-01192692/document

[10.1] https://en.wikipedia.org/wiki/Rebound_effect_(conservation)
[10.2] http://www.sti.uniurb.it/events/sfm10qapl/slides/katinka_slides.pdf

References

[12.1] https://www.barroso.org/publications/ieee_computer07.pdf
[12.2] https://hal.archives-ouvertes.fr/hal-02399517/file/20191202_plaquette_pi_licences_V1.1.pdf
[12.3] https://www.data.gouv.fr/fr/pages/legal/licences/
[12.4] https://fr.wikipedia.org/wiki/Long-term_support
[12.5] https://ecoinfo.cnrs.fr
[12.6] https://ecoresponsable.numerique.gouv.fr/publications/ressources-ademe/
[12.7] https://principles.green
[12.8] https://www.fun-mooc.fr/en/courses/environmental-impacts-of-digital-technologies/
[12.9] https://digitalcollage.org/
[13.1] https://www.softwareheritage.org/save-and-reference-research-software/

[14.1] https://www.edari.fr
[14.2] https://fr.wikipedia.org/wiki/Message_Passing_Interface
[14.3] https://fr.wikipedia.org/wiki/OpenMP
[14.4] https://fr.wikipedia.org/wiki/Loi_d'Amdahl
[14.5] https://pop-coe.eu
[14.6] https://www.softwareheritage.org/save-and-reference-research-software/

[15.1] https://ecoinfo.cnrs.fr/2021/06/12/consommation-energetique-de-lutilisation-de-lia/
[15.2] https://hal.archives-ouvertes.fr/hal-03922093
[15.3] https://en.wikipedia.org/wiki/OpenACC
[15.4] https://en.wikipedia.org/wiki/OpenCL
[15.5] https://github.com/Breakend/experiment-impact-tracker
[15.6] https://github.com/lfwa/carbontracker
[15.7] https://codecarbon.io/
[15.8] http://www.green-algorithms.org/
[15.9] https://mlco2.github.io/impact/#compute

[16.1] https://gr491.isit-europe.org/en/
[16.2] https://eco-conception.designersethiques.org/guide/en/
[16.3] https://en.wikipedia.org/wiki/Dark_pattern
[16.4] https://collectif.greenit.fr/ecoconception-web/115-bonnes-pratiques-eco-conception_web.html
[16.5] https://www.ssi.gouv.fr/guide/recommandations-pour-la-securisation-des-sites-web/
[16.6] https://greenspector.com/en/home/
[16.7] https://www.greenit.fr/2019/07/02/web-evaluez-lempreinte-dune-page-en-un-clic/
[16.8] http://www.ecometer.org/
[16.9] https://www.fruggr.io/
[16.10] https://greenframe.io/
[16.11] https://force11.org/info/the-fair-data-principles/
[16.12] https://en.wikipedia.org/wiki/KISS_principle
[16.13] https://e3p.jrc.ec.europa.eu/sites/default/files/documents/publications/jrc114148_best_practice_guide_2019_final.pdf

[17.1] https://www.ssi.gouv.fr/en/publications/
[17.2] http://collapseos.org/
[17.3] https://solar.lowtechmagazine.com/about.html