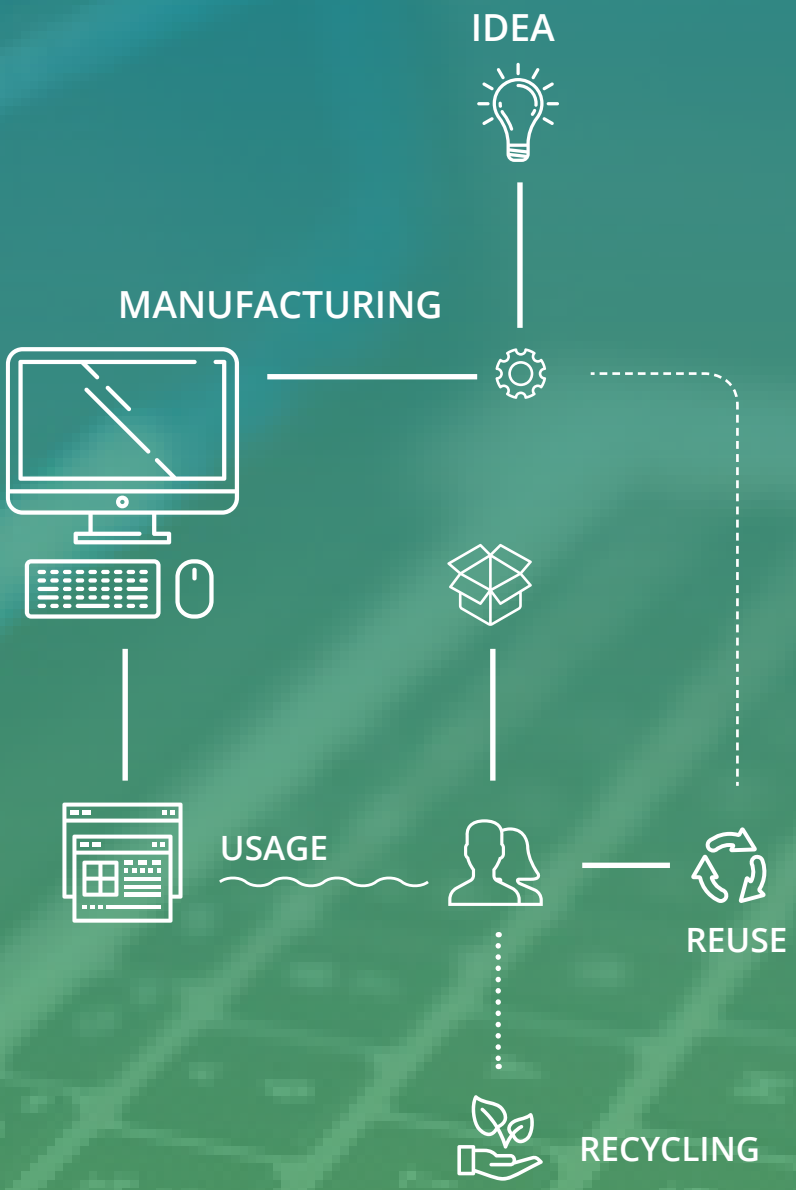
METHODOLOGICAL GUIDE



Software LCA

GREENSPECTOR

orange

ADEME

THANK YOU FOR DOWNLOADING!

Don’t miss out on our other sofware eco-design resources:

YES, I WANT TO KNOW MORE!

(+33) 9 51 44 55 79
contact@greenspector.com

GREENSPECTOR

I Why conducting a software LCA?

Eco-design consists in taking into account environmental and sanitary impacts during conception or improvement phases of a product or service. It is perceived more and more like a value creation process, in all kind of businesses and areas. This phenomenon is growing as companies get more sensitive to their share of responsibility in the future of subjects such as our planet or next generations. The other reason is firms realize the numerous benefits they can get out of such a process.

There is actually a domain eco-design is in the introduction phase: the software world, in which most methods and best-practices remain to be written. Plus, just like in any other economical areas, the benefits perceived by the different actors from the digital world are numerous and very interesting:

Cost reduction

By trying to reduce resources or raw material needed to produce a good, **eco-design also allows to decrease manufacturing costs.** This principle is applicable to software as well. Indeed, in the software production phase, lowering the number of functionalities to develop, the amount of work stations to deploy, the quantity of impressions to generate or the energy needed for the software to function, is a way to reduce pollutions generated by the activity, as well as diminish software manufacturing costs.

Software eco-design is perceived more and more like a value creation process, in all kind of

Anticipation of environmental rules

More and more norms are inflicted to companies to make products, and more generally the economy, more virtuous environment-wise. For instance, think of Electrical and Electronic Equipment related rules (EEE), RoHS, WEEE, REACH or ErP, aiming at making products less polluting. We could also mention current or future government attempts to integrate environment deterioration costs to our economy, which as of today aren't undertaken by companies (negative externalities): CO2 emission rights, carbon tax, etc. Now these rules are implemented and others around the corner, it is safe to say **companies which already considered this eco-design issue are a step ahead have a true competitive advantage** compared to other firms.

Product differentiation

« Eco-designing » is also developing a better-quality product that is at the same time **more resistant, more durable and more frugal** for the user; as these benefits go tightly with the impact reduction of the product on the environment and/or the extension of its active life cycle. The user can get the most of it. For example,

power consumption is an actual issue for a datacenter manager, who would perceive greatly a less energy consuming software, especially in an area such as this one where "Cloud operators" keep appearing on the market and better optimize their resources usage. Also, mobile platform autonomy is a key stake for smartphones and tablets' constructors and users, battery consumption it generates is a metrics to take into account.

Innovation factor

The French Ministry of Ecology, Environment and Sustainable Development declares on its website: [1] (translated to English) :

> *« Eco-design is a spur for innovation, both at the product function level and the different steps of its life cycle. Having a fresh look to optimize consumptions (materials and energy) and to reduce pollutions can sometimes lead to brand new ideas for a product's components, the way it works or the technologies it uses».*

This is true for software, but also for any other type of product.

Company's image

We are in a time consumers are more and more attentive to corporate social responsibility (CSR) efforts of companies, and being actively engaged in applying software eco-design principles for sure benefits a company's image and prestige, with positive financial impacts.

After making these observations, a group of « Green IT » experts founded the Green Code Lab which goal is to promote Software eco-design and offer tools and methods to facilitate the implementation.

As part of the collaboration between Orange and GREENSPECTOR, winner of a call for projects on software eco-design launched by ADEME, both companies brought each of their expertise together to continue working on the subject presented on this Methodological Guide to software LCA.

We offer here a methodology to conduct a software Life Cycle Assessment (LCA) that truly suits the objective of defining a methodology to diffuse widely in order to initiate future requests of software impact evaluation.

LCA is a tool you can't pass on when it comes to software eco-design. This methodology lets you assess the environmental impacts of manufactured goods, services and processes, and this in a very

Indeed, LCA is a central tool that is a key element in software eco-design. We are in a case of standardized methodology (ISO14040 and ISO14044 among others) which lets you assess the environmental impacts of manufactured goods, services and processes, and this in a very complete way. Examining pollutions generated at every single stage of the product life cycle (conception, production, usage and decline) permits to not forget any of them and figure out which stage pollutes the most (the one you should focus on at first). This effort will vary depending on the company's decisions, choices and

1 *http://www.developpement-durable.gouv.fr/L-eco-conception-c-est-quoi.html*

strategical constraints.

Having an overall vision of all stages also allows you to make sure a solution lowering the impact on the environment at a certain stage will not generate more pollution at another stage of the product life cycle (avoiding pollution and/or impact transfer).

As a consequence, the purpose of the Methodological Guide to software LCA is to offer a methodology to conduct a software LCA.

Defining a mutual methodology to this category of products is justified by the fact that software, that are often wrongly considered as intangible, hold specific features different from the «average tangible» products. This intangibility raises questions on what is the best way to conduct such an analysis on software.

We will put the emphasis on describing these specific features and presenting what we believe is the best approach to meet the objectives we would have previously set. Then, we will explain in detail how to implement this approach in the different normalized LCA stages.

Let's point out that the social aspect, which is one of the three pillars of sustainability and to which the Green Code Lab particularly pays attention, isn't discussed directly in this document (beside the indirect sanitary impacts). However, social[2] LCA methodologies exist widely, and what is mentioned here is applicable and transposable to any social and societal impact analysis.

---

2 Outil GSF de NTT NTT-Orange collaboration et Rapsodie Gross Social Feel-good Index—Social Impact Assessment for ICT Services : https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200703043.pdf

II What are the goals of a software LCA?

> *Comparing environmental impacts of several products or software solutions in order to pick the one with the lower environmental impact.*

As we will see further down, the first step in a software LCA is the defintition of goals and scope of study.

This step is essential as it will impact numerous factors choosing in what way the next steps of the study should be organized, but also the results of the study themselves. That is the reason why LCAs are said to be « goal dependent ».

When it comes to software, we can identify several goals:

Studying environmental

Studying environmental impacts of a given software (already performed): consumption of non-renewable resources, energy, pollutant emission (chemical or particles) in water, air and soils.

Determining the most impactful stages in the life

Determining the most impactful stages in a software life cycle: production/ development, usage, transportation, and decline. This particular type of study can narrow down to just software categories (email software, word processing software, CMS, web pages...)

Identifying improvement opportunities for impacts

Identifying improvement opportunities for future products, as well as ones for the reduction of impacts on the environment. This aim is particularly targeted by editors and software creator that are mindful to develop a product with a better environmental quality.

Comparing environmental impacts

Comparing environmental impacts of several products or software solutions in order to pick the one with the lower environmental impact. As a consequence, users (Information System Department, individuals, etc) or developers/integrators facing technological choices can use this tool. In the context of a compared LCA (evolution of a software or new product), only the phases that changed between the two versions of product/service will be calculated. But careful, comparing two LCAs can be tricky. In order to be trustworthy, the comparison should be executed with the same software, at the same date, with the same cut-off rules and, if possible, by the same person.

Just like any other LCA, in order to be published, a software LCA must be the subject of an independent critical review.

III Special features of software products

This part aims at presenting characteristics specific to software during a life cycle assessment (LCA).

For each issue raised by these features, we will explain what approach we recommend to assess environmental impacts.

3.1 Software: tangible or intangible?

Software is a very special type of good:

- It doesn't produce any direct tangible waste.

- It isn't connected directly to power supply, hence isn't seen as « consuming ».

- However, it does have an environmental impact represented by the consumption of resources and energy, due to hardware needs for its development and usage.

The goal of a LCA is to evaluate environmental impacts of manufactured goods, services and processes. But here, the question is: which category does software belong to?

As an initial reaction, it seems obvious that software has similarities with tangible goods, like the ones produced by the traditional industry, as they are materialized by a set of computer data (source code / executable code) that we can trade, own and use to answer a specific need.

However, it is important to make the difference between storage medium and physical interfaces of software interaction itself: software simply is a « state » of the storage medium (made of a unique and well-defined sequence of 0 and 1), a « state » of the network moving data around, a set of « states » of the screens displaying the software graphical representation, etc. So, should we consider software more as an intangible good?

To answer these questions it is key to distinct the software itself from the service it offers. This way, we can consider software as an intangible good, offering one or more specific services (features or content). As an intangible product, its environmental impacts will result from consumption of resources (human, physical, tangible...) needed for the implementation of different phases of its life cycle: manufacturing/development, operating phase, distribution, decline.

3.2 Should we isolate software from its operating environment?

It is rather obvious software doesn't function by itself, but always in an ecosystem of software it depends on, starting with OS (exploitation system), or with which it communicates and interacts. With this method, measuring impacts generated only by the studied software during usage phase is pretty tough.

Software impact never goes without the hardware and OS it works with: during a LCA, identifying environmental impacts linked to OS or hardware correctly isn't possible. However, these impacts can be retrieved thanks to comparative LCAs, which means by comparing LCA of two very specific configurations. Let's illustrate with an example: Software A on Hardware X with OS1 versus Software A on Hardware X with OS2. Or for instance, conducting sensitivity analyses would allow to asses impact deltas linked to different hardware.

IT equipment doesn't' necessarily work only for the studied software. Most of the time other applications and software are running at the same time, on the same equipment, thus, are consuming resources. As a consequence, the power consumed by the equipment cannot be associated with the studied software only.

In order to assign a software the energy it consumes, the strategy implemented as part of the Web Energy Archive (www.webenergyarchive.com) research project was to subtract the energy consumption induced by the OS and specific services such as antivirus (it is called consumption in idle mode) to the whole consumption of the equipment.

3.3 Software: what perimeter to consider?

One of the main issues we encounter when we think of environmental impact evaluation of a software is that it evolves quite a lot from one version to another (correction, features, etc) and it can have a modular architecture, or even work simultaneously on different equipments.

3.3.1 Software keeps changing

Software breaks down in a variety of versions and sub-versions with different features.

We may be tempted to say it doesn't lead to any major issue as versions are spaced out in time, but it rarely is the case.

When an official release of a well-identified version is available, software can quickly become the core subject of corrective patches or complementary modules, which may be very numerous and frequent.

It has been very common and has been the trend in the last few years.

It is important to differentiate minor evolutions of a software from major ones:

- **Major** evolutions carry new features, or even a complete application restructuration.

- **Minor** evolutions mainly carry bug corrections or addition of minor features.

As talking about a « finished » version of a software is tricky, we suggest limiting the study to the « latest stable version that is the most used ». No matter what version you study, it will have to be mentionned explicitly in the study assumptions.

The impact of corrective and/or operational versions, whether minor or major, will be taken into account only with a sensitivity analysis.

This means we model the impact of a bug correction or feature evolution by adding resources (HR, paper consumption, hardware...) during the manufacturing/development phase or in a new specific phase (maintenance).

3.3.2 Software is often modular

Software itself can be broken down into different modules we choose whether or not to install, or it can offer the possibility to install plugins and add-ons (like it is the case for most internet browser).

We cannot model this concept of modularity per se with a LCA, for a simple reason: it would be tough, almost impossible, to identify specific resources needed for the development of each and each modules.

You'll have to consider the most standardized configuration possible, then you'll be able to run sensitivity analysis in order to assess impacts of resources needed to develop specific modules (HR, hardware...).

3.4 What is the life cycle of a software?

Most life cycles of products analyzed by LCAs can be considered as composed of the following six stages:

- Product development
- Raw material extraction
- Production and packaging process
- Logistics and distribution process
- Product use
- End of life (disassembling, transportation, sorting, recycling, wastes).

Nonetheless, if this life cycle makes sense for a basic tangible product, it isn't really suited for software:

-Indeed, as an « intangible good », software doesn't require any raw material extraction directly.

The production phase doesn't work like a manufacturing process you repeat N times to produce N copies of a product: you should consider it more like a unique stage creating a version of a software theoretically reproducible and reusable endlessly.

Upstream transportation and distribution

Regarding upstream transportation (logistics), if the software is made of different modules developped in other sites, you should take into account, as much as possible, the "sending part" from others sites to the modules' aggregation site. In a first approach, these impacts could be negligible, as they are more likely to represent less than 5% of total impact.

If distribution to end-user is conducted through a download on internet, this download's environmental impact should be taken into account. If distribution is done via a tangible support (DVD, USB key...), production and transportation of these supports should be taken into calculation as well.

Software installation can be linked to the use phase.
Maintenance can be considered as production overcosts. A software's end of life seems non-existing, or at least without any impact. We will see later how wrong that statement is. We will have to integrate the program removal process and the data destruction or retrieval associated with the uninstallation process.

Just like it is stated[3], we can simplify a software life cycle by keeping only 4 stages: production, distribution to end-user, actual use and end of life/reutilization/recycling

We can simplify a software life cycle

Production Design and development process is considered as a unique stage allowing to produce the software. This phase includes the whole software design process:

- need analysis
- design,
- programming,
- test,
- stabilization,
- deployment.

- Resources associated with correctional maintenance acts (bug fix) and functional enrichments are to be included in this stage

- Software is often composed of elements such as frameworks, libraries, etc. In that case, we can consider the production of these components has a negligible impact if we look at the amount of copies (reutilizations) that are made.

Distribution to end-user : Several scenarios are possible, we'll present briefly three of them.

- Downloading: software and documentation are distributed electronically. The program issuer (download server) perimeter has to be taken into account, just like the recipient's (end user's computer), as well as the infrastructure used to send electronic files (networks, routeur etc), by taking a portion of the hardware manufacturing and energy needed to download the sotfware depending on used resources..

- Software and documentation are packaged and sent in the mail, hence the support have to be taken into account (CD-ROM, DVD, USB key, documentation), so as for packaging and mailing services associated.

- User can get the license and the user manual in a local shop or via mail and download the software. The packaging step (manufacturing & transportation) has to be taken into account, the software download too. In that particular case, impacts due to users movements can be rather high and become greater than the other impacts. Previous LCAs, conducted by the Orange Group on terminals, mobiles, modem, CD-ROM, showed the clients' moves can vary a lot from each other and be very impactful, particularly if it is done by car (several kilogramms of CO2).

The software utilization by the end-user is initiated by the installation on its hardware (initial operation) following the download (distribution) for instance and covers the whole software use stage on the user's suited hardware. Perimeter includes:

- Hardware needed or required to use the software. In this case, we consider the portion of:

- hardware manufacturing (user's equipment, network access, server access),
- the energy used when the hardware is on (user's equipment and potentially network access and server access),

which could automatically integrate the consumption of required software;
- required software integrating its own resource consumption (OS, virtual machines...). We can isolate the resource consumption of thoses mandatory software by establishing a standard value called "Idle" which is the resource consumption of hardware and its requirements, before any execution of the analyzed software; this value can be split in as many values as we want if we wish to isolate OS from browser for instance.

- The software being assessed and integrating its power consumption:

- data needed to use the software or ones created by it and stored on the application's different resources;
- power consumption associated with this data is integrated by default in the equipment

For example, if we take a Web page, the hardware and software requirements to display the page are: a computer/tablet/smartphone, an OS (Android, Windows, iOS...), a browser (Firefox, Chrome, Edge, Safari...) and potential plugins.

End of life / Reutilization / Recycling:

We assume that, at the end of life, a software is erased or uninstalled on the user-side and the editor-side. There are several things to take into account for this step: the end of life of support hardware and software-generated data.

- End of life of support hardware: cf. § 3.5.

- Data end of life: we can uninstall the software properly folllowing a procedure deleting all setting files on the client's terminal. In this phase, you should also take into consideration the software-generated data created willingly, or not, by the user. In that case, we may face different situations:

- the user does not wish to retrieve the data he created;

- the user wants to get its data back in order to use them with a similar tool and a conversion process exists, this process was included in the new tool during the design and development phase;

- the tool doesn't allow the retrieval and data conversion process for a new use, in that case we will have to estimate the conversion impact for the user in that end of life stage.

3.5 End of life: software-induced

The end of life stage of a software can be tough to apprehend and manage.

Obsolescence as such doesn't actually exist for software. Indeed, theoretically a software is endlessly

The end of life stage of a software is especially hard to apprehend in the life cycle assessment, specifically for the two following reasons:

Obsolescence as such doesn't actually exist for software. Indeed, theoretically a software is endlessly usable, as long as hardware exists to make it work. Software doesn't recognize wear and doesn't break down because it has become too old itself. As a consequence, we cannot properly predetermine a software lifetime duration, as it is linked to its components degrading throughout time. The only explanations to software obsolescence are external to the software itself:

- user's choice to delete it,
- maintenance policy of a version,
- obsolescence of hardware supporting the software,
- obsolescence of other software interacting with the software we analyze (exploitation system, database...),
- disappearance of the user's need
- etc.

A software doesn't seem like it is generating any physical waste in its end of life stage.

Whenever we decide not to use it anymore - or when we cannot use it - it is simply deleted from the terminal on which it is installed, without generating any physical waste. In the worst case scenario, there are remaining files uselessly occupying disk space.

But in reality, if we have a closer look, we can find:

- physical wastes (wastes from the design and development stage - CD + package + user guide in paper form if the software was packaged - taken into consideration in other stages of the analysis),

- and more specifically hardware-related wastes (computer, smartphone, tablet, network equipment...) generated from the use of hardware, required to make the software work.

But the question is: how does software contribute to generating wastes? Well simply with its direct or indirect impact on hardware obsolescence.

- **Replacement or software update requiring new equipments:**

For a similar user's need, if the software goes through a major update or if it is replaced with another software, plus if this operation requires additional physical resources (more powerful machines, different technologies), then we can consider older hardware as software-induced waste. This is a phenomenon of « hardware obsolescence » caused by software renewal: hence, software is reponsible for the wastes. A mature software (with no functional evolution) has no reason to spearhead wastes... But what software doesn't evolve, right? It'll be necessary to watch consumption of resources required by the new software versions.[4] .

- Side effects of uninstallation on other software:

You also need to pay attention to other software as uninstallation can make them obsolete: dependences can exist, which could lead to a cascade effect of obsolescence.

If a software participates in making an equipment obsolete it means the software update enriching the operational service is consuming more and more resources, until the support equipment isn't compatible anymore. In this case, both the software and service are responsible for wastes. A mature software (without any operational evolution) shouldn't participate in generating wastes.

- Wrong uninstallation:

An uninstallation process that is badly executed - or badly applied - can contribute to obsolescence as well. Indeed, registry keys are left out, temporary files too; if the software modifies the system, it remains a residual footprint which makes the system heavier. If an editor decides not to maintain a major software of an equipment such as PC, then the terminal will become obsolete and generate wastes. And the software will be reposible.

4 SLI : an indicator to assess software durability, Frédéric Bordage, http://www.greenit.fr/article/logiciels/sli-un-indicateur-pour-evaluer-la-durabilite-des-logiciels-4237

IV *What differentiates a software LCA from a service LCA?*

A service is provided by the use of a software, relying on a terminal's tangible resources (computer, mobile, tablet) which possibly need network resources or remote IT equipment resources (service platform). Software is a constituent element of a service..

Because of that, environmental impact assessment of a software will come out different from a service one.

A software Life Cycle Assessment only focuses on resources needed for software development, as well as the tangible resources' share needed for it to function properly.

On the other hand, a service Life Cycle Assessment includes all resources needed to make it possible (terminals, software, networks, service platforms, servers...) and will rely on these LCA elements.

As a consequence, in order to conduct a website LCA, conducting a LCA of the whole chain supporting the service is necessary: servers, network and client terminal, both for the hardware and software part (server, client, network...). The impact of the analyzed software (here one or more web pages) on different hardware components will be taken into account, particularly those forming the system and allowing the end user to access the web page(s).
When it comes to some service LCA, the software impact on some hardware could be very weak, thus not taken into consideration. For instance, it is the case for the website's network part (to be confirmed thanks to the software LCA of network elements).

LCA results of the different systems (equipment) will be used as data for the service LCA. However, it will be necessary to check that operating units of the elements forming the service are coherent (use time, data, allocations, etc).

Another term is that LCA must have been conducted with the same exact set of indicators, with the same software version, with the same accuracy, the same "cut-off" rules, and preferably conducted by the same person.

V *Functional unit of a software*

Defining the analyzed product's functional unit is a mandatory step to LCA; as comparing two products' environmental performances only makes sense if the service in the end is the exact same.

Functional unit represents a product's function quantification.

From this unit, it will be possible to compare different products' scenarios. Like any other unit, it has to be very accurate, measurable and additional. Without being so specific, the functional unit should include a functional component, a performance criteria and a set duration.

Here are a few **examples of functional units** for other types of product and industrial processes[5] :

- For **paint:** painting 1 sq m of wall with an opacity of at least 0.98, measured with an opacimeter, for 20 years.

- For a **mobile phone**: one year being operational on a 3G network..
Software works differently because services it offers are various and most of the time pretty complex. A software usually owns a whole bunch of features..

- For a given **set of features**, the amount of possible application cases associated is rather high.

A software may have different uses and user types: a majority of user utilizing only a couple of basic features, and few users using advanced ones. As an example, Word can be used as a Notepad, or as a macro model, presentation tool, printing one etc.

However, when you will be trying to define the functional unit, you will have to pay attention to services offered by the software, such as:

- Writing X pages of a document in a word processor

- Writing and sending an email of x lines to y recipients through a messaging software

- **Video player**: read x minutes of video of a certain given quality (resolution, compression level)

-**Web server:** processing x HTTP requests

-**Database**: handling x data Mo or answering y requests

In the context of a service like accessing a web server from a terminal, you will have to identify a functional unit ("consult the homepage for 40s on a laptop") and the support architecture: :

- 1 main server
- 2 CDN servers
- several network elements crossed on average
- 1 box or company switch
- 1 client terminal: laptop

Introduction to Life Cycle Assessment (LCA), External brief note : may 2005, ADEME.

VI *System boundaries and data collection*

This paragraph aims at explaining how to set the study's perimeter, to list the required data and to get them.

There are two types of life cycle assessment: attributional and consequential. The attributional LCA[6] describes physical flows required and emanated for a product or process; whereas, the consequential LCA describes how coherent environmental flows will vary, depending on decisions that are made. This second type impacts border definition.

We would use preferably the attributional LCA, which objective is to assess environmental impacts emanating from a process/service. .

6.1 Scope definition of a LCA study

When conducting a Life Cycle Assessment, and after defining the study goal and functional unit, the next step is to determine the study scope, and to do so you need to list all basic processes involved in the study of a "software" product. Describing the system with a flux diagram (input and output) mentioning the processes and their relationships is rather useful. The energy input and output should be considered and treated just like any other LCA input/output.

Scope definition is iterative; often, you'll need to go back on the scope to include or exclude processes, either because specific data isn't available or because a process must be included as it presents an impact that needs to be more precisely identified.

Skipping some steps of the life cycle, the process, input or output is possible only if they don't alter significantly the general study conclusions in the end. These decisions must be clearly stated, as well as what they involve and why they were skipped.

In terms of LCA study, we take into account different cut-off criteria to pick inputs to be included in an analysis, like: masse, energy, environmental scope etc. Similar cut-off criteria can also allow you to identify which output it is best to watch in the environment; for instance by including final waste treatment processes.

Cut-off criteria are also established in a way that it makes it possible to conduct life cycle assessments in a fair timescale. Indeed, it is counterproductive to allocate a lot of time to search data for elements which environmental impact is negligible.

6.2 Collecte des données

Data collection involves power consumption measures (development, tests, storing, hosting, usage...), the identification of the electrical mix used depending on the country, the quantity of consumables (paper, ink cartridge...), distances (deliveries, travels...), material and electronic components identification in used terminals, real physical characteristics measure of elements (masse, surface, development...), network related consumptions etc.
Primary data must be retrieved from developers. Secondary ones can be used whenever direct measures aren't conductible.

6 Finnveden, G. et al. 2009. "Recent Developments in life cycle assessment". Journal of Environmental Management vol 91 p. 1-21

The Energy Star database, available online, is a source of data appropriate to use for desktop, laptops and servers. However, if the device and settings aren't the same, it might not be totally reliable.

Regarding secondary data issued by the LCA software, the geographical zone, the creation date as well as the source of each data used in the study must be provided.

It is possible to use LCA results for machines (servers, computers...) if they were conducted with the same LCA software (impact indicator) and the same database version (or else an update is required).

6.2.1 Production phase

Software libraries

When software libraries are developed by third-parties, it is pretty complicated to get primary data related to software development. You will only be able to calculate impacts of the parts developed within the company.

Estimation techniques will be used for the size and complexity of a software; it can be the estimation of an effort in h.j for the development, or the code size (amount of code lines or Mo).

Tests and development

You have to take into account:

- The human resources in h.j needed for software development and light maintenance, revealed with the time record associated to stages such as: software design, development and tests before it gets deployed.

- Development and test related consumptions of energy (computers, servers), but also broader energy fluxes (energy, lighting, air-conditioning) of software manufacturing or production location. Computers used to develop a software are considered as capital equipment, thus they are out of scope.

- The amount of travels and distances (car, train...) required for the application development, the type of car used.

- Consumables used during the development and test processes (like paper, office supplies).

- Physical meetings and videoconferences.

In that case, a suitable method for consumptions attribution is based on the amount of h.j for each software development and on the calculation of an emission factor per employee to allocate to development and test.

6.2.2 Distribution and installation stage

Distribution of the software can be done either in the electronic way (with a download on the internet) or via a physical medium (DVD, USB).

It might also be a combination of the two: distribution by physical medium to the central system of a company, then electronic distribution to individual users in a firm. When a combination is used, you should use a weighted average.

For an electronic distribution, you should include:

- storing and hosting of software by servers (including mirror ones)

- network use (WAX & LAN) for software transfer and download

- computer/terminal use for a software download by the end-user

For a physical diffusion, you should include:

- raw material and media production (DVD or CD)

- casing and packaging

- physical documents delivered with software

- media transportation (including storing, if applicable)

Regarding the software installation, you should take into account: (the actual installation stage should be included in the usage stage, except if training sessions is needed)

- the duration of computer/terminal use by the end-user in order to download the software;

- the power consumption required for the software installation, plus the ad hoc electrical mix;

- the use of network (data volume in Mo or Go for instance) to transfer and download the software;

- For the installation, we can consider the software installation time on the terminal on which it has been downloaded;

- The initial training in h.j, the power consumption of the client's terminal/ computer and a matching electrical mix.

For company software that is rather complex (such as ERP systems), there is a significant level of activity in this stage, whereas it can only include software physical delivery (or distribution) for "out-of-the-box" software.

6.2.3 Use phase

Measurement of the energy consumption of a software on use.

In order to do that, you need to have a use scenario for the software(x hours a day), to know the electrical mix of the country of installation and to declare the type of measure tool and the methodology associated.

The energy related to the use of a software can represent most of the energy consumed by ICT hardware, which can be truly affected by the software development.

During the LCA of an ICT service, measuring the energy used by the hardware necessarily involves the energy used by the software; in that case, you don't have to assess separately the software energy consumption. Nonetheless, in the case several software are running at the same time on a same terminal, you will have to take into account only the consumption of that specific software.

6.2.4 End of life (EOL) stage

- When a software gets distributed via physical mediums, the emissions associated with the EOL of those mediums (CDrom) should be included.

- When hardware is used to make a software work, it has to be included in the event that the stopping of the software makes it obsolete.

If, in order to extend a « service », the installation of a more powerful software version (major version) is required and it involves furthermore tangible resources (machines, technologies), then we can consider older hardware as new waste.

As a consequence, it is necessary to implement the rules related to WEEE (Waste Electrical and Electronic Equipment): reutilization or collection, dismantling, sorting, recycling, landfill and transportation associated.

IT managers handling equipment in EOL stage and WEEE eco-organizations (or local waste disposal department) picked by firms will participate in information collection.

Because of a lack of data, or depending on the results previously obtained with other product and service LCAs, the following data can be considered out-of-scope:

- support services ((R&D / Capital Goods / Sales & Marketing),
- client's travel by car to go pick up its software
- secondary and third packaging,
- upstream transportation of components,
- software that uninstallation of the software targeted in the study make obsolete if they are not created – or there is no information available,
- registry keys left out and temporary files (remaining residual resources) in the case of a bad uninstallation..

In alignment with the GHG Protocol, emissions caused by the manufacturing of capital goods (buildings, machines etc) can be excluded – in that case, computers used to develop the software would be considered "capital goods".

6.2.5 Quality of data

The quality of data retrieved to form an inventory must be assessed in order to determine its relevance and reliability.

The method used relies on recommendations from the guide Product Environmental Footprint (PEF) guide (European Commission, 2012).

A data quality indicator is calculated with the following formula:

$$QD = \frac{(MC + AM + C + U + DR + T + GC + TC + Qmin.4)}{(6 + 4)}$$

with:

MC, methodology coherence
AM, acquisition method,
C, completeness,
U, uncertainty,
DR, data representativeness,
T, timeliness
GC, Geographical Correlation
TC, Technological Correlation,
Qmin, weakest quality obtained.

Data Quality rating	***Data Quality level***
≤1.6	*Excellent quality*
>1.6 à ≤2.0	*Very good quality*
>2 à ≤3.0	*Good quality*
>3 à≤ 4.0	*Acceptable quality*
>4	*Poor quality*

Only the most impactful elements will be assessed.

VII *Life cycle impact assessment*

The Life Cycle Impact Assessment (LCIA) turns flux inventory into a series of impacts very well identifiable due to software and LCA database.

Similarly to the rest of the life cycle assessment, impact evaluation is based on a functional unit.

The life cycle impact assessment uses a list of inflows as inputs (raw material, transformed material, energies ect), as well as some outflows (rejections, wastes, emissions, etc) aggregated on the whole system, which is being analyzed at every single stage. These fluxes are aggregated in impact categories to, in the end, provide category indicators.

Ultimately, it is possible to reach a unique environmental rating, even though it involves to weight impact categories between each other.

It exists several methods to conduct such an assessment.

Issue-oriented methods

The chain of cause and effect for environmental issues is rather rough. Most of the time, we can notice primary effects directly coming from analyzed activities, such as CFC emissions; and secondary effects which are basically the consequences, like stratospheric ozone depletion, leading to an increase in UV rays reaching human's level, hence developing more cataract issues and cancers.

These methods are also known as « mid-point » methods.

Damage-oriented methods

Unlike issue-oriented methods, damage-oriented ones put an emphasis on regrouping impacts according to results, as deep as possible in the chain of cause and effect. This is why these methods are also called « end-point »..

Remarks

(note issued from normation):

For numerous products and services, software-related emissions (in all stages beside use) don't represent a big share compared to the overall emissions of the system being assessed, especially as emissions due to software development are amortized on the amount of software copies that will be made. As a consequence, it won't be necessary to conduct a detailed assessment of the software life cycle.

However, if a software is being custom-made with a smaller amount of instance, it is recommended you do a prior assessment of all stages, so you can determine whether or not a detailed analysis of emission is needed.

VIII *Indicators*

It is crucial to determine impact indicator (mid-point categories) and damage categories that will be measured during an analysis. This choice actually depends on the objective set by the company, and also on the availability of exploitable data.

Here below are a few examples of impact indicators used in the software Simapro.

- **GW** (Global Warming):

This indicator evaluates the contribution to global warming caused by the emission of greenhouse gases. It is expressed in g eg CO2.

- **OD** (Ozone Layer Depletion):

This indicator evaluates the contribution to the depletion of stratospheric ozone layer by atmospheric emissions. It is expressed in g eg CFC11.

- **PO** (Photochemical Oxidation):

This indicator calculates the production of ozone in the tropospheric layer via the action of solar radiations on oxidizing gases. It is expressed in g eg C2H4.

-**AE** (Aquatic Eutrophication):

This indicator calculates the eutrophication (gain in nutrients) of oceans and lakes by affluents. It is expressed in g eg PO4.

- **AT** (Aquatic Toxicity):

This indicator evaluates toxicity of water by taking into account the authorized top-level concentrations of effluents. It is expressed in dm3.

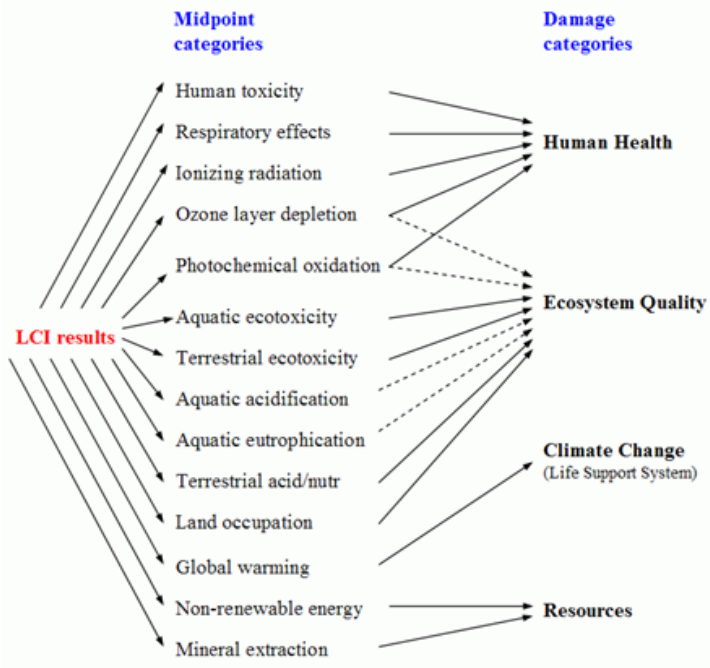


Fig. : Scheme by IMPACT 2002+, linking the life cycle inventory (LCI) and damage categories, via mid-point categories.

For instance, damage categories (End Point):
-Human health (DALY),
-Ecosystems quality (PDF.m².an),
-Climate change (kg CO2 Eq),
-Resource consumption (MJ primary energy).

All these impacts can be analyzed separately and, in the end, brought back altogether in the form of a unique impact unit such as « environmental footprint » or « planet overshoot day », as it is usually offered by LCA software.

Damage Categories	Standardization factors	Units
Human Health	0,0071	DALY/point
Quality of ecosystems	13700	PDF.m^2.an/point
Climate change	9950	kg CO_2 /point
Resources	152000	MJ/point

Table: Standardization Factors for all four damage categories for West Europe (Jolliet et al. 2003, Humbert et al. 2005)

Flow indicator (example)

- ED (Energy Depletion):
This indicator calculates the energy depletion caused by fuels (fossil, uranium for nuclear power, wood, etc.) as well as alternative sources (hydroelectricity, solar, wind energy, wave and tide etc.).

The indicator also takes into account the energy contained in materials (produced during their combustion in the end of life stage for instance). It is expressed in MJ.

- **WD** (Water Depletion):
This indicator evaluates the consumption of used water, no matter its origin or quality (potable, industrial, etc). It is expressed in dm3.

Design indicators (example)

Design indicators allow to put the emphasis partly on:

- the number of different material used
- product end-of-life indicators:

- % of reuse,
- % of recyclability,
- % of energy recovery,
- % of wastes.

For software, we identified impact indicators and fluxes:

- « Climate change (CC for ILCD) » kg eq CO2,

- « Aquatic eutrophication – Freshwater (AEF for ILCD) » kg eq P,

- « Abiotic resource depletion (ARD for ILCD) » kg eq Sb,

- « Respiratory inorganics (RI for ILCD) » kg eq PM2.5,

- « Ionizing radiation – Human health (IRHH for ILCD) » kg eq U235.

Fluxes:

- « Energy depletion » MJ

- « Water depletion » m3

IX *Interpretation of the life cycle assessment*

Contributions to impacts and fluxes are identifiable with a LCA software:

- impacts in phases allow to establish some sort of hierarchy which varies depending on the company's concerns (CO2, freshwater eutrophication...) and room for action;

- same thing inside the different stages of the life cycle themselves, it is possible to know if impacts are connected to the software production or maintenance, to travels of development teams, to materials used in production, to storing or even hardcopies printing...

Consistency check

This control aims at making sure the results we get are compatible with the scope of study initially defined. In case you are comparing two different scenarios, it is advised to demonstrate that, for each scenario, the chosen hypotheses are consistent with one another.

These differences between scenarios can come from differences in data sources, data precision, technological representation, etc. Differences that are related to time, geography, age of data and indicators also have to be taken into account.

During the checking phase, used data have to be compared to initial recommendations. Differences have to be documented and justified.

Uncertainty analysis

Its goal is to verify the uncertainty impact of main data on the model results. This is usually conducted with software tools, using, for instance, the Monte-Carlo technique.

Sensitivity analysis

The objective is to validate the reliability of final results by determining their influence on variations in hypotheses, source data and methodology.

The sensitivity check can be done to any element of the analysis: imputation, exclusion criteria, system border, chosen impact categories, standardization data, etc.

X *Limitation of LCA*

Even though life cycle assessment is a global method allowing the evaluation of environmental impacts, some limitations have to be considered.

First off, it is only about potential impacts - and not real ones. Plus, results are a lot dependent on hypotheses initially chosen (scope of study, functional unit, etc.) and also on quality of data (availability, confidentiality, complexity, etc).

Because of that, conducting such an analysis requires a level of knowledge and competences rather important in this domain. Regarding the service or software design, one of the factors limiting the LCA is the volume of data needed to conduct the study.

When developing a new service or software, the life cycle assessment requires quite a good and advanced knowledge of the software or service, thus with fixed settings, like technical choices, that will then determine the impact of that same new software or service.

There is indeed a risk with libraries we use to develop an application as it is nearly impossible to assess their impact on the production and use stages.

And finally, as the life cycle assessment focuses on environmental impact evaluation, it is rather frequent that recommendations coming from the interpretations of the study results end up conflicting with other interests, like economic or social perspectives for instance.

The goal of this part is to present the first results coming from the assessment of environmental impacts of the application being analyzed and on which we applied power consumption reduction measures.

This first software life cycle assessment relies on the recommendation series ISO14 040.

That way we considered the following steps in the life cycle;

Manufacturing
- of equipment on which the application was developed and tested
- of the analyzed application itself

Transportation
- of equipment from the manufacturing location to the Orange plant, where the application has been developed.
- of the analyzed application's different modules

The application use stage
- for convenience and simplification reasons we installed on a same terminal (Raspberry Pi) different instances of the SDS software, which allows to model different equipment supposed to communicate with each other.

End of life
- of equipment used to develop the application
- of the application.

By definition LCA is multicriteria, but for simplification and clarity purpose on this first study on software impacts, we will only focus only on the impact of CO2 emission.

11.1.1 Functional unit

For this study, we suggest taking the following functional unit :

« turn on 20 small equipment (alarm clock, coffee machine, light, electronic shutter...) 20 times a day for a whole year »

Each solicitation lasts more or less depending on the chosen software version:

- Test duration of non-optimized version lasts: 78,5s.
- Test duration of optimized version is shorter: 74,38s.

The goal of the study is to be able to measure environmental impacts (power consumption) caused by the SDS application operation on the whole 20 equipment.

On a Raspberry Pi we install 20 SDS application instances that we will solicit 20 times a day for a year (365 days).

11.1.2 Detailed life cycle stages

All calculations explained here-after can be found in an Excel sheet (.xls). The main objective of the upcoming paragraphs is to briefly present the approach that was implemented.

11.1.2.1 Manufacturing stage

Application development and production phase

The application was developed over a one year and half period by a developer, so about 300 person-days (1 person-year at Orange = 200 person-days)

Power consumption of all development computers and laptops was measured over a period of two working days (11,75 kWh), this data was then compared to an estimated calculation of the consumption of those equipment. The results that came out are very similar. These data allowed us to get the power consumption needed to develop the application: 1,7 MWh.

This amount of energy can be converted in equivalence with kg of CO2 by considering the emission factor for France (0,14927 Eq CO2 kg/kWh). Hence, to develop the application, 263kg eq CO2 have been diffused.

Along with this power consumption linked to the application development, you have to include the buildings share (lighting, heating, AC...). The Orange Labs building located in Rennes has been hosting teams since the 70's, plus that location is also composed of a company cafeteria. Consumptions in water, gas, electricity include both service buildings and the company restaurant. As it wasn't possible to extract data from the Orange Lab building only, it has been decided to refer to the consumption data provided by ADEME (Agency for the Environment and Energy Management) and CEREN (Centre for Economic Studies and Research on Energy) for an office building located in a zone with an average climate. The Orange Labs building is 14 456 m² big and usually hosts 602 people, so it represents about 30,5m² per person. For CO2 emissions that include all fluxes, the building share added to a developer is 2240 kg eqCO2, and only 1520 kg eq CO2 if we just consider heating.

Manufacturing phase of development PCs

For this stage we studied the developer's work environment:

- Development PC
UC : HP Z420 Workstation,
Screen : Samsung SyncMaster BX2240,

- Office desktop ,
Laptop DELL Latitude,
Screen Dell U2312HM,

- Netgear switch (to connect two equipment on the Orange intranet network).

The life time of a development PC was set to five years, and a laptop is four.

For simplification purposes, the developer's whole environment wasn't taken into account: settings, saving, storing...

Environmental impacts (primary power consumption) caused during the production phase of those equipment were provided by suppliers. When data wasn't available for the desired reference, we considered the average of similar equipment.

The share of the production phase of equipment with global impacts is proportional to the application development duration (300 person-days). Thus, when it comes to the development PC, the actual life time in use is 1000 days (5 years of 200 days/year).

The production stage contributes to 30% of total impacts issued by the supplier.

Regarding laptop, if its lifetime is four years, only 37% of production phase impacts will be taken into account.

Overall, the production of PC and laptop issues 394 k eq CO2.

Production phase of the Rasberry Pi used to model the five equipment

Just like development PC, for the Raspberry Pi, we only consider the production share that is proportional to the soliciting stage.

The Rasberry Pi is used 20 times a day for 19 seconds, for an overall lifetime of 7 years.

Over a five year period of application operation, the use phase/production phase ratio is 3.10-8. We can then deduct that the production phase of the Raspberry Pi equipment is negligible.

11.1.2.2 Upstream transportation

- Upstream transportation for the application:

In the context of this study, no transportation stage were to be considered for the application.

- Transportation of development equipment:

Just like we saw for the production phase, only 30% of development PC's impacts and 37% of the laptop's will be taken into account for transportation.

Information on production location and transportation means is available on equipment suppliers' websites. Development PC is transported by boat between Asia and Europe, whereas laptop is transported via airfreight. These websites also provide the weight of packaged goods (in kg).

Emission factors (boat/truck/plane) are from IEME (eq kg CO2 per kg.km of transported good).

Distances are the following:

Camion Chine	2000 km
Bateau Chine - Europe	18000 km
Avion Chine - Europe	10000 km
Camion Europe - France	500 km

In the end, transportation of PC and laptop emits 33kg eq CO2.

11.1.2.3 Use phase

Power consumption caused by the Rasperberry Pi running during a session (78,5s) for all twenty software instances is 0,43 mWh. The session is repeated 20 times a day, every day of the year. Power consumption for a year is 39 Wh, so an emission of 5,9 g eq. CO2.

11.1.2.4 End of life of IT equipment

Les équipements de développement sont supposés avoir une fin de vie de D3E standard. Les calculs donnent 7,84 kg eq C02.

11.1.2.5 Overview

Development equipment is supposed to have a standard WEEE end-of-life. Computations are 7,84 kg eq C02.

Production of PC + laptop	394,39	kg eq. CO_2
Distribution of PC + laptop	33,110	kg eq. CO_2
Code development (300 days)	263,088	kg eq. CO_2
End-of-life	7,84	kg eq. CO_2
Building with heating + other fluxes (light...)	2 240	kg eq. CO_2
Building with heating only	1 520	kg eq. CO_2
Phase of code usage (1 year)	$4.72\ 10^{-4}$	kg eq. CO_2

The table here-after gathers every lines connected to the production phase (excluding the building).

Development phase (excl.buidling)	698.428	kg eq. CO_2
Buidling with heating + other fluxes (light...)	2 240	kg eq. CO_2
Buidling with only heating	1 520	kg eq. CO_2
Code usage phase (1 year)	$4.72\ 10^{-4}$	kg eq. CO_2

After receiving GREENSPECTOR 's recommendations which aims at optimizing the application, five days were needed to implement – in other words, a total of 305 days to develop this newer version of the code.

CO2 emissions linked to the application use phase represent 4,5 g eq. CO2 (for one session). All impact figures increase as the share of the application development rises, going from 300 days to 305 days.

Here is the table of emissions for the optimized version :

Production phase	Production of PC + laptop	400,96	kg eq. CO_2
	Distribution of PC + laptop	33,662	kg eq. CO_2
	Code development (300 days)	267,473	kg eq. CO_2
	End-of-life	7,97	kg eq. CO_2
	Phase of code usage (1 year)	$4.39\ 10^{-4}$	Kg eq. CO_2
	Building with heating + other fluxes (light...)	2 280	kg eq. CO_2
	Building with heating only	1 550	kg eq. CO_2

This table here-after presents all of the lines connected to the code production phase (excluding buidling):

Development phase (excl.buidling)	710,07	kg eq. CO_2
Buidling with heating + other fluxes (light...)	2 280	kg eq. CO_2
Buidling with only heating	1 550	kg eq. CO_2
Code usage phase (1 year)	$4.39\ 10^{-4}$	kg eq. CO_2

11.1.3 Interpretation

The key points coming out of the LCA results are the following.

- The prevalence of buildings' footprint in CO2 emissions – which are about ten times more important than emissions related to software development itself. One of the first recommendations would be to stay in a low energy building or, even better, one with a positive energy surplus.

- Moreover, manufacturing of PC and laptops represent a share that is slightly higher than the one from the development phase.

- Finally, CO2 emissions associated with a 1-year-use stage are in reality rather low if we compare with the ones caused during the software development phase. The ratio is even more important if we include equipment manufacturing. Let's all remember that these use phase emissions are from an application working intermittently and for short periods (78s).

This type of ratio in-between impacts from manufacturing phase and ones from the use phase represent an optimized operation in terms of power consumption. As soon as the use phase is over, consumption goes void.

- The application considered may not be ideal for the implementation of a software eco-design operation. Indeed, energy gains only represent 7% and absolute values of CO2 emissions gain per use (0,472 & 0,439g CO2) are very weak too. Thus, calculating the number of optimized software versions that are to be deployed in order to compensate code-optimization related overheads. (5 extra developing days) brings us to way higher values which don't show much relevance to the approach.

- These 7% gains are to be put in perspective with the 40% gains Orange[7] managed to get during the complete refactoring of their application Every Where, for which all eco-design axes were implemented: architectural and functional need. In that particular case, the only way the 7% were obtained is thanks to the replacement of the most energy-consuming sequences, which is pretty satisfying knowing that only one third of recommendations were actually implemented.

- The whole point of this operation is to validate this approach and process, as well as its potential to reduce environmental impacts of software.

7 « Comparaison des couts d'implémentation entre les versions BEW V8 – V9 » Internal document to Orange – confidential.

XII *Conclusion: next steps*

This document allowed to use a methodological and theoretical process to assess environmental impacts of a software.

For a first example of software application in the connected objects world, we weren't able to neither assure the methodology portability, nor validate hypotheses chosen for the next explorations.

This methodology requires a bit more testing to improve and actually set the base for environmental impact measurement of software with a standardized LCA methodology.

Overall, these results are encouraging in terms of work and studies realized by Orange and GREENSPECTOR.

This document can be completed in the context of future analyzes done by Orange and GREENSPECTOR, as well as the Green Code Lab national community for software eco-design.

13.1 Reminder on LCA related norms

ISO norms:

– ISO 14040: LCA – Principles and framework

– ISO 14044: LCA – Requirements and guidelines

– ISO 14048: Data documentation format

– ISO 14049 : Examples of application of ISO 14041 to goal and scope definition and inventory analysis

Complementary TIC norms:

– ITU-T SG5 - Q18 L Methodology: Goods networks and services, part 1

– ETSI TS 103 199 V1.1.1 Life Cycle Assessment (LCA) of ICT equipment, networks and services; General methodology and common requirements

– GeSI /Carbon Trust - ICT guidance on WRI/WBSCD product and value chain standards

– IEC TC 111 - IEC/TR 62725, "Quantification methodology of greenhouse gas emissions (CO2e) for electrical and electronic products and systems".

Authors:

Marc VAUTIER - ORANGE
Elisabeth DECHENAUX - ORANGE
Thierry LEBOUCQ - GREENSPECTOR
Olivier PHILIPPOT – GREENSPCTOR

+33 (0) 9 51 44 55 79
contact@greenspector.com