Université de Namur
Faculty of Computer Science
Academic Year 2017–2018

Literature Survey on Software Eco-Design : Definitions, Impacts and Effects among software life cycle phases

Antoine Sacré

UNIVERSITÉ DE NAMUR

Internship mentor:	Laurent Lefevre
Supervisor:	______________________ (Signed for Release Approval - Study Rules art. 40) Michaël Petit
Co-supervisor:	Pierre-Antoine Rappe

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Abstract

Software has an effect on the environment through the influence it has on the hardware supporting it. This influence has not received the attention it deserves and it is still difficult today to estimate the responsibility of software in the impact of IT on the environment. There are two main visions when it comes to reducing the environmental impact of IT. The environmental effect sought during software use can be positive, in this case we speak of "IT for Green", or the environmental effect sought during the life of the software can be a reduction of its negative impacts, in this second case we speak of "Green for IT". This latter vision requires appropriate development methods to estimate and reduce negative environmental effects. This thesis aims to provide a literature survey on means for reducing the negative environmental effects of software, through a technical view of software development. To achieve this, we defined the concept of software eco-design and collected research works in similar areas to gather information relevant to this type of design. The results should provide a view of the practices used in this area and also identify issues and research gaps to help researchers guide their future research.

Keywords: *Software Eco-Design, Literature survey, Software environmental impact*

Résumé

Le logiciel a un effet sur l'environnement par l'influence qu'il a sur le matériel qui le supporte. Cette influence n'a pas reçu l'attention qu'elle mérite et il est encore difficile aujourd'hui d'estimer la responsabilité des logiciels dans l'impact de l'informatique sur l'environnement. Deux grandes visions existent lorsqu'il est question de la diminution de l'impact environnemental de l'informatique. L'effet environnemental recherché durant l'utilisation du logiciel peut être positif, dans ce cas nous parlons de "IT for Green", ou bien, l'effet environnemental recherché durant la durée de vie du logiciel peut être une diminution de ses impacts négatifs, dans ce deuxième cas, nous parlons de "Green for IT". Cette dernière vision requiert des méthodes de développement appropriées afin d'estimer et de réduire les effets négatifs sur l'environnement. Ce mémoire a pour but de fournir une revue de la littérature sur les moyens visant à réduire les effets négatifs du cycle de vie du logiciel, par le biais d'une vue technique de la conception des logiciels. Pour ce faire, nous avons défini le concept d'éco-conception logicielle et nous avons rassemblés des travaux de recherche dans des domaines similaires afin de recueillir des informations pertinentes pour cette approche de la conception. Les résultats devraient fournir un aperu des pratiques utilisées dans ce domaine et aussi identifier les problèmes et les lacunes de la recherche afin d'aider les chercheurs à orienter leurs recherches futures.

Mots-clés : *Éco-conception logicielle, Revue de la littérature, Impact environnemental du logiciel*

Acknowledgements

This thesis is the result of work that began in September 2017 as part of my internship at the École Normale Supérieure (ENS) of Lyon in France, for which I would like to thank the team I was part of for their sympathy and the discussions that were always very interesting in the canteen. My internship was carried out for the EcoInfo group of researchers for whom I would like to thank Françoise Berthoud, Jean-Marc Pierson and especially Laurent Lefevre who supervised me at the ENS.

I would also like to thank Pr. Petit for his availability when I needed it and for the proofreading efforts and his constructive remarks during my internship, the finalization of the associated scientific article and during the completion of my thesis.

Finally, I thank my family and Juju for her patience during the three months of internship and her unfailing support during the writing of this thesis.

Contents

Chapter 1

Introduction

Both hardware and software levels of ICT have an environmental impact. ICT production processes are complex, expensive and require material resources that require energy to be extracted to produce the material. The use of ICT would also have effects on the environment both through changes in behaviour of their users and by replacing processes with computerized processes. According to the Climate Group [118], energy consumption and the material resources needed to support this sector are increasing from year to year and are expected to continue to increase in the coming years. Thus, the improvement in hardware energy efficiency from year to year will probably not be sufficient to compensate this increase. This is not only due to the increase in the use of computers but also to the increase in resources and energy consumptions of software itself.

During the past years, the environmental impact of ICT has been studied more and more. Efforts to limit the growth in environmental impacts are concentrated on the hardware side of computing (setting up labels, laws, conducting LCA studies, etc). The software aspect, on the other hand, has received less attention.

Developing and using software is a multi-phase process that requires energy, material and economic resources, these elements being an integral part of the environmental impact of the software. Each phase of this process has its inputs and outputs. These outputs influence the realization of the rest of the process. For example, the technical characteristics of the equipment required to operate the software or the number and complexity of functionalities influence the amounts of resources that are used in the rest of the process. Dealing with the input resource quantities of each phase and the influences between phases makes it possible to consider an increase in efficiency (the ratio between the quantity of input in relation to the quantity of output) for each of these phases, in order to reduce the environmental impact throughout the process.

Several authors have studied the environmental impact of several types of software. Among these, F. Jiang, et al. [82] studied mobile applications and I. Manotas et al. [60] have, among others, studied embedded software and data centers. This thesis is, besides, not limited to a particular type of software, as long as softwares considered in research works are those that can be developed by the general phases of the software development cycle from the classical waterfall model (requirements engineering, design, development and maintenance).

Even if impacts are not classified in this thesis, it is nevertheless interesting to recall that impact classifications exist. Several authors have attempted to classify the different types of environmental impacts of ICT. Among these, F. Berkhout and J. Hertin [119], who, in synthesizing the literature on the environmental impacts of ICTs, came to this classification:

- First order impacts: direct effects of production and use of of ICT (they consider that there are positive direct effects if ICTs is used in an environmental protection context)
- Second order impacts: environmental impacts due to the effects of ICT on behaviour, economy, industrial processes...
- Third order impacts: indirect impacts on the environment due to feedback processes due to the large-scale production and use of computers.

L. M. Hilty and B. Aebischer [120], based on the work of previous authors, while incorporating some developments, such as the consideration that there is no direct positive effect in first order impacts. Finally, the classification used by Mr. Börjesson Rivera et al. [121] differs primarily from the previously cited authors in that second-order and third-order effects are combined into a single second-order effect group. The authors justify this by the often vague and unnecessary separation between these two levels. The different classifications that can be used are conceptually quite similar.

There is a design approach that aims to reduce environmental impacts throughout the life of the product developed: the eco-design approach. In the context of eco-design, the increase in efficiency is achieved with two constraints that are: the software developed is always fulfilling of customer needs and is still economically viable. This approach is currently used, among other things, for computer hardware, but there is no reason, given the current situation, why eco-design should not be used on software. Likewise, in the scientific literature, more attention was devoted to hardware eco-design than software eco-design and this is also the case for practice oriented literature. Thus, at this time, the notion of software eco-design is still unclear and uneasy to grasp or understand.

Given this, this thesis is about the software eco-design of any kind of software aiming to reduce negative impacts regardless of their level and during all phases of the software life cycle. Concretely, this thesis has four objectives.

The first is to give a clear definition of the concept of software eco-design based on current knowledge. This definition will then guide the rest of the work.

The second is to present current knowledge on the calculation and reduction of negative environmental impacts using software eco-design. Since software eco-design aims to reduce negative environmental impacts in any software life cycle, the objective is to carry out a literature review structured according to the phases of this life cycle, with a design perspective.

In order to develop the knowledge regarding the eco-design of software, the third objective of this thesis is to identify the effects of increasing the efficiency of one phase on the other phases, in terms of environmental impact. Indeed, the consequences of increasing the efficiency of one phase could be an increase, or a decrease, of the resource consumption of another phase and, potentially, of the entire software life cycle. This challenge must be studied in order to guarantee or accentuate the positive effects of eco-design on the entire software life cycle.

The last objective is to identify research challenges. Given the work done to achieve the previous objectives and given the usefulness such an objective brings to advancing research in the field, this objective is relevant.

The thesis is structured as follows: the research method used for the literature survey is explained in Chapter 2. In Chapter 3, motivations for using software eco-design are explored. Then, the concept of eco-design is clarified. Finally, the concept of software eco-design that will be used throughout this thesis, is defined. Chapter 4 displays the state of the art. Works about knowledge synthesis on software eco-design, and related fields, are reviewed. In Chapter 5, the literature review as such is carried out. It is structured according to phases of the software development life cycle and research works are presented one phase after the other. At the end of this chapter, a visual representation of effects among phases that may influence the global environmental impact of targeted phases is presented. Following on from Chapter 5, Chapter 6 contains the research challenges identified during the literature review. Finally, chapter 7 draws a conclusion to the thesis.

Chapter 2

Research method

The research method used in this thesis is based on the guide for conducting a systematic literature review of information systems by C. Okoli and K. Schabram [122]. The main advantage of systematic literature reviews is that they allow not only a researcher to have a clear vision of a field before starting a research, but also any researcher interested in a field to obtain a serious knowledge base. Since this thesis is only inspired by systematic literature reviews, it does not claim to be qualified as such.

2.1 Databases used for research

Several databases of scientific research works have been used to carry out this thesis. In the preliminary search phase, Google Scholar, IEEE Xplore and ACM digital library were mainly used. The search engines of these databases were employed, allowing to avoid an imbalance for one or another publisher (Google scholar), but also computer sciences-based databases to complement the results (IEEE and ACM digital library). During the research work selection phase, three other databases were used when too few results emerged from other databases (Science Direct, SpringerLink and Wiley Interscience).

2.2 Note taking and reference management

Note-taking when reading the research works was inspired by the SQ3R method[1], which helps avoid accidental plagiarism by questioning the information provided by a passage and by trying to recite it, as if to explain to someone what has just been read. This allows not to use the words of authors.

Reference management was done using Zotero, to record, sort and annotate research works systematically. In addition, this tool allows searches in all the text bodies of research works, making it possible to quickly find information.

[1]Robinson, Francis Pleasant. (1970). Effective study (4th ed.). New York: Harper & Row.

2.3 Research works selection

For the research work search, the method used is inspired by the one described by C. Wohlin [123], giving good practices for the use of the snowballing method in systematic literature reviews.

The first step was to find the keywords to use when searching. In order to find these keywords, a preliminary search was carried out to quickly probe the state of the literature. The keywords first used were very general: "Eco-designed software" or "Eco-design of software" to find a maximum number of research works on the subject. Later, keywords were extended to "Green" and "Sustainable" because those were identified as much more common than words such as "Eco-design". Consequently, the query used for the preliminary research had the following form:

- (Sustainable OR Green OR Eco-efficient) AND software AND (development OR engineering)

Then, when more specific topics had to be sought, typically for research works dealing with a particular phase of the life cycle, the following query was used in the titles, abstracts and bodies to find research works:

- "(Green OR Sustainable OR Environmental OR (Energy OR Power) Efficiency OR Eco-efficient) AND <specific subject of research> "

After identifying keywords, results of the three search engines have been browsed and the relevant research works were selected through a systematic course of titles and abstracts. The priority was given to exclude research works whose research topic was not related to ecological concerns. The research works obtained through this method were then used as bases for backward snowballing (searching for other potentially relevant research works in the research work references), until a sufficient number of relevant research works have been selected. Forward snowballing was not used systematically but was used when too few research works were found with the previous method.

A sufficient number of research works was a number allowing each specific subject of research to not exceed about ten research works, after a basic reading of the research works (this number was being able to vary from one section of the thesis to another). As this work was done by only one person, it was not possible to browse more. If too many research works were found in a theme, this theme was divided into sub-themes or was restricted to keep only the most relevant ones. If too few research works were found, the preliminary search was extended to the following search engine: Science Direct, SpringerLink and Wiley Interscience. Finally, if too few research works were found after this additional research, the subject of the research in question was considered as not being addressed yet.

2.4 Research works classification

Through the reading of research works, it was possible to construct a consistent classification in relation with the objectives of this thesis and the information found in research works.

In order to compare the selected research works, a subset of criteria used by B. Kitchenham and P. Brereton [7] and K. Petersen et al. [115] to rank research works were used. Only a subset of the quality criteria defined by B. Kitchenham and P. Brereton [7] were used because the goal was not, like these authors, to create a precise measure of quality but rather a basic measure to have an idea of the research progress in specific topics.

- Is the research work based on a research? If so, what is the nature of this research?
    - Literature review : analysis of the state of the art in a field or a specific topic
    - Case study : "An empirical enquiry that investigates a contemporary phenomenon within its real-life context" R. K. Yin [73]
    - Problem identification and solution: "A solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution is shown by a small example or a good line of argumentation." K. Petersen et al. [115]
    - Validation Research : "Techniques investigated are novel and have not yet been implemented in practice. Techniques used are for example experiments, i.e., work done in the lab." K. Petersen et al. [115]
    - Evaluation research : "Techniques are implemented in practice and an evaluation of the technique is conducted. That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes to identify problems in industry" K. Petersen et al. [115]
- If the scientific paper is not based on a research, what is its nature?
    - Commentary or discussion paper based on expert opinion : "These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should been done. They do not rely on related work and research methodologies" K. Petersen et al. [115]
- In what phases of the life cycle is the research work situated?
    - Requirement engineering process
    - Design
    - Development
    - Maintenance
    - Operation
    - End of life
- What is the relationship of the research work to negative environmental impacts?

– Identifying the nature or categories of environmental impacts of software
– Defining a negative impact estimation model
– Proposing solution to reduce a negative impact
– Highlighting challenges in implementation of solutions

The difficulty here is not to be influenced by the authors' words when a research work is not exactly of the nature defined by the author. Moreover, if a research work has several parts and only one is relevant to this research, the classification is based on this part only (for example, if a research work addresses subjects other than environmental impacts and only carries out a case study on environmental impacts, this one is considered as a case study).

2.5 Data extraction

The most important ideas have been extracted from each research work (based on the understanding of the research work and field). Then these ideas were grouped into the corresponding categories, in the form of narrative synthesis.

Finally, the challenges were collected, and trends and research gaps were identified in the literature to establish the final results.

2.6 Limitations of the research method

Given this methodology, several biases are possible. The search is carried out by a single person; hence the searched terms as well as research works considered relevant can be biased. The research results which differed too much from the author's expectations may have been considered, involuntarily, as irrelevant.

Since this thesis is not a systematic review of the literature, only a part of potential relevant research works was studied and not all research works responding to a search. Sorting was done by refining the research, which may have hidden a relevant part of it.

During the research work selection stage, called the practical screen step, in the article C. Okoli and K. Schabram [122], the reader should ideally be able to justify, for each research work considered irrelevant to the subject of the study, the reasons for this choice. However, the work carried out to produce this thesis had to be based on a compromise between the quality of the results and the time available. Consequences of this compromise are that it has not been possible to be so methodical in research.

In addition, the inclusion criteria, which must be clear and defined in advance to ensure repeatability of the research, have evolved over time and a researcher conducting the same research may have trouble finding exactly the same set of research works.

Chapter 3

Software eco-design

The primary objective of this chapter is to give a clear definition of the concept of software eco-design, based on current knowledge. However, this objective is broken down here into several parts. The first is to try to explain the motivations behind the use of the software eco-design approach. The second is to define what software eco-design is. Finally, other research topics relevant for software eco-design are covered.

3.1 Reasons to use software eco-design

The following section explains why it is fundamentally interesting to integrate the eco-design concept into current software development methods. For this, the conceptual origin behind the problem of the search for sustainability, through society, ICT and finally software is describe.

3.1.1 Impact of today's society on the environment

Today's society has an environmental impact, mainly visible through climate change. Even if it is still possible to hear that it is necessary to "save the planet", it is important to remember that it is not the earth that is in danger, nothing will prevent it from turning on itself, but it is rather our descendants who are in danger.

J. Rockström et al. [28] have attempted to establish the limits that humans should not transgress at the global level, otherwise the environmental impacts would become too important at the level of the planetary system and therefore for today's society. The limits not to be transgressed are divided by planetary process, of which, according to the authors, two are not yet calculated and three are already exceeded (see Figure 3.1). Each global process would influence the others and would all be impacted by human activity. Authors argue that although they do not know the mechanisms behind these processes and although they do not know the duration or the amount of effort needed to reduce negative impacts, it is necessary to act to reduce these impacts.

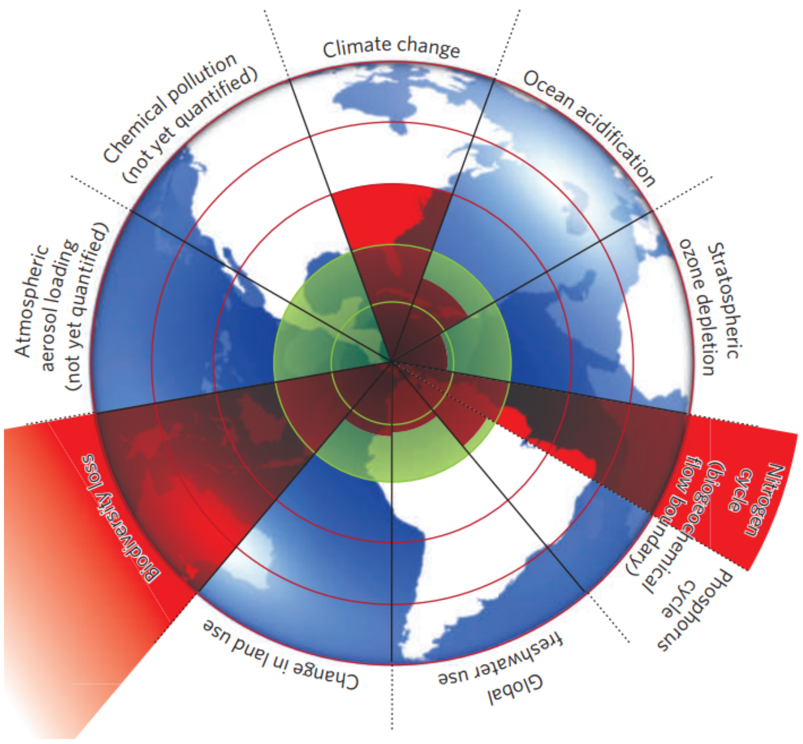

Climate change
Ocean acidification
Stratospheric ozone depletion
Nitrogen cycle
(biogeochemical flow boundary)
Phosphorus cycle
Global freshwater use
Change in land use
Biodiversity loss
Atmospheric aerosol loading (not yet quantified)
Chemical pollution (not yet quantified)

Figure 3.1: Planet boundaries. "The inner green shading represents the proposed safe operating space for nine planetary systems. The red wedges represent an estimate of the current position for each variable. The boundaries in three systems (rate of biodiversity loss, climate change and human interference with the nitrogen cycle), have already been exceeded" extract from J. Rockström et al. [28]

3.1.2 Impact of ICT on the environment

In this section, some sources of impacts, the ICT is responsible for, are covered. These sources are the resources needed to produce the computer equipment and the energy needed to operate it, but also less tangible sources such as increase in dematerialization and use of computers in general.

Many metals are needed to produce electronic components and their extraction diminishes the reserves and increases the efforts to extract fresh resources, thus increasing the energy consumption for their extraction. S. Behrend et al. [30], characterized the rare metals on the basis of several criteria, such as their prices, the quantities of the reserves and the locations of these reserves. Their results were that, for now, the available reserves of metal resources decrease while the electronic waste increases and this condition is not sustainable in the long term.

According to A. Andrae et al. [33] and I. Baudry et al. [34], electronic product production uses complex processes, expensive in water, energy and raw materials of various kinds. However, the environmental analysis of the production of microelectronic equipment is in fact very difficult to realize. According to J. Bonvoisin et al. [35], there are several reasons for this difficulty. First of all, production processes use unconventional substances whose energy impact for production has not been precisely measured. Then, the production of an hardware part is dispersed among a large number of suppliers, each of whom would have to be subject to an environmental analysis (almost impossible because of the scarcity of data due to nature of the market, which is very competitive). Finally, according to H. Schütz et al. [36], impact estimation methods should not concentrate solely on the geographical location of the use phase because an important part of the environmental damage is actually in countries producing and recycling waste.

Energy is required to operate all equipments directly related to the ICT sector. According to A. Noureddine et al. [38], ”According to W. Vereecken et al. [37] ICT consumes up to 7% of global power consumption (or 168 GW) in 2008. This number is predicted to grow and double to 433 GW in 2020 or more than 14.5% of worldwide power consumption”. Since energy is not produced in a 100% renewable way, it contributes to the depletion of resources, this time energy resources. Finally, the industry supporting IT infrastructure could be the most polluting industry, according to S. Erkman et al. [39] and F. Flipo et al. [40].

According to J. Bonvoisin et al. [35], quoting E.D. Williams et al. [43], dematerialization would generate a ”secondary materialization”, based on the fact that dematerialization would be only a displacement of material. Services that are supposed to be replaced by IT services would often still be used despite the appearance of their IT counterpart. Even if some IT services have no ”physical” counterparts, i.e. they do not replace anything and act more like new services that did not existed before and they still need an IT infrastructure to support them.

The ICT still has positive impacts on the environment, according to J. Bonvoisin et al. [35], which carried out a state of the environmental analysis of the ICT as part of his thesis. First, computers have played an important role for many years at the macroeconomic level because they allow better productivity, performance and optimized use of resources. According to V. C. Coroama et al. [41], a point in favor of the ICT sector is that it would consume less energy than it can save (it could even allow, according to The Climate Group [118] to save 15% of CO2 emissions in 2020). This substitution effect, in transport, is illustrated by Weber et al. [42] by "moving bits instead of atoms".

To conclude, even if the negative impact of computing is not negligible, it should not be underestimated that without ICT, the negative impacts of our economy could have been worse. The ICT has changed our society since its inception, it is omnipresent in everyday life and in companies, according to A. Cefriel et al. [45]. ICT has a role to play, but it can not be seen as naturally helping to reduce negative environmental impacts. However, its design and reasoned use has the potential for it. Thus, according to B. Penzenstadler et al. [18], ICT's influence on other industries is such that the efforts undertaken will be accentuated directly, indirectly or by rebounding them.

3.1.3 Impact of software on the environment

Softwares have an environmental impact throughout their life cycle. To limit this impact, it is important to estimate its extent but also to be able to calculate it.

Throughout its lifetime, software will be responsible for a series of effects on the environment. According to M. Mahaux et al. [47], who studied the impact of software and not ICT in general, impacts of each phase of the software life cycle are composed of the direct impact (the consumption of equipment used to design the software, the physical travels of the developers, etc.) and the indirect impact of "investment" in sustainability. This "investment" would be planned during the design, with the aim that the designed software reduces its negative impact in the other stages of its life cycle.

Estimating the overall environmental impacts for which software is responsible is difficult. Even if it is possible to give an order of magnitude of the environmental impact of the IT sector, these remain very broad and depend greatly on the assumptions of the researchers. Attempting to measure the software impact at a macroscopic level requires making assumptions: is the consumption of the software equal to the consumption of equipments by subtracting the consumption of these at idle ? Should we consider the evolution of the consumption of cooling equipment during the execution of a program? Is it necessary to consider the consumption linked to the maintenance stages? When does consumption related to software begin and end? Etc. The list can become very long and several answers are possible for each of them.

Estimating the environmental impacts of each software is easier. The key to allowing a sufficiently precise calculation of the software impact, so that it can be used, is to proceed in a targeted way to the software so that the study is possible within reasonable time and cost and to proceed in a systematic manner so that evaluations are comparable to each other.

3.1.4 Relevance of using software eco-design

Given that the environmental impacts of software are made during its life cycle and that it would be possible to make an "investment" in sustainability in each phase of the development cycle, thus reducing the environmental impact of subsequent phases of the development cycle, it appears necessary to take environmental constraints into account as early as possible in the design of the software.

The software eco-design approach makes it possible to respond to this need by starting an innovation dynamics in software design in order to reduce its negative environmental impacts from the very beginning of the development process.

3.2 Definition of eco-design

Software eco-design is a relevant approach to be undertaken in order to limit the negative effects of software on the environment. However, it does not have a reference definition. In order for this thesis to nevertheless be based on a clear definition, we will construct from those currently found in the literature. First of all, it is interesting to start from the definition of eco-design in general. Being the subject of more research, eco-design will serve as a basis for establishing a definition of software eco-design.

According to R. Karlsson and C. Luttropp [74] who carried out an overview of the subject area of eco-design, as eco-design is primarily a design, it takes on its main features. Eco-design thus not only has, as an objective, a meeting of need, but also a provocation of desire by the customer for the product or service designed. However, the prefix "Eco-" adds an ecological dimension, a respect for nature in the broad sense, which would translate into a search for sustainable consumption and production. According to the authors, the combination of the two terms means that eco-design would aim to support sustainable development, by integrating itself into it and whose results also participate in it, while aiming for "human satisfaction" [74].

In order to construct a definition, the view expressed by a practitioner were also used. According to C. Charbuillet [63] Eco-design is the "Integration of environmental constraints in the design of products and services according to a global and multi-criteria approach"[1]. According to the author, a global and multi-criteria approach means that a decision to improve the impact of one environmental criterion in one phase of the life cycle would not impair the impact of at least one other criterion or one criterion in another phase of the life cycle. This eco-design approach would be used preventively and meets four dimensions at once: customer expectations, technical feasibility, cost control and reduction of negative impacts on the environment. Finally, the author cites five key principles. These are:

- Product evaluation (identification and analysis of impacts),

[1]own translation

- Product improvement (through the implementation of a decision to limit the identified impacts)
- Inclusion in the company's strategy
- Team awareness
- Communication

Other authors, such as P. Knight and J. O. Jenkins [116], who have studied the lack of adoption of eco-design, consider that eco-design is equivalent to Design for Environment and define eco-design as "the systematic integration of environmental considerations into product and process design (...) The process aims to minimise the costs and "adverse environmental impacts of products throughout their entire life cycles" [117] ".

3.2.1 Analogy between product life cycle and software life cycle

The latter definition was established with manufactured products in mind as the subject of the eco-design. In order to evaluate if this definition would be also applicable on software, a cycle comparison has been made, to determine if each phase of the software life cycle can be related to each phase of the life cycle of a manufactured product.

Each phase is consider uniformly, without taking into account the potential variations of different development environments.

First of all, the requirements and design engineering phase of softwares can be seen as equivalent to the design phase of manufactured products. Indeed, their objectives are to define all the properties of the future product developed in accordance with customer and industrial requirements.

Then, if the extraction of resources in the context of the development of manufactured products is intended to make the raw resources available to the development process, then, the organization of all the elements necessary to start software development is equivalent to this step. In particular, the creation and training of development teams or the search for external companies to start software development can be considered as resource extraction steps.

Thirdly, the development and test phases can be seen as equivalent to the production phase. Indeed, the production phase is the phase from which resources are transformed, based on the specifications defined during the design phase, into a final product.

Then, the distribution phase of a software product can be assimilated to the transport phase. Indeed, the mode of distribution can be seen as a mode of transport of the product, from the developer to the final consumer. The types of software distributions are close to the type of software itself, with generic products being sold to a large number of customers through download platforms, and personalized products being ordered directly by the customer himself to the software editor.

The penultimate phase is the use phase. This phase exists both in the case of a manufactured product and in the case of software.

The last phase is the end of life phase. Like the previous one, these two phases exist in both cases, manufactured product and software. The difference here is the cause of the end of life. Unlike manufactured products, software products do not wear out, but deteriorate during the maintenance phase. The software is thus maintained until economic reasons push to stop this maintenance or the software becomes useless (the need disappears) or is supplanted by a better alternative product.

In conclusion, the life cycle of a software is very similar to the life cycle of a manufactured product.

3.3 Definition of software eco-design

The objective of this section is to present a definition of the software eco-design approach. For this, scientific references defining this term have been searched. With the exception of the thesis written by Bonvoisin et al. [35] and the article from J. Pinto et al. [62], our search has been unsuccessful. An important information that emerges from this search is that the terms "software eco-design" seem to be used mostly by French speaking authors, but it is difficult to draw any conclusions from the few articles dealing with the subject. Only two articles were found, including research on software eco-design, whose authors are not French speaking : the article written by S. Kawamoto et al. [61] in 2005 and the article written by J. Pinto et al. [62] in 2006. However, despite their age, they have been quoted only once, suggesting that the subject, even if it exists in the literature, is not really popular. This aspect is all the more surprising since the term "Eco-design" is used internationally in the hardware part of the ICT.

According to J. Bonvoisin et al. [35]:

> "The integration of environmental considerations in the design is usually referred by the term "eco-design". (...) Eco-design can be defined as the maximization of the relationship between socially useful function given by a system and its environmental impact, this definition based on the functional dimension of the eco-design. To be fully effective, an eco-design approach must be global and multi-criteria. On the one hand, it must integrate the entire life cycle of the system, that is, all the human activities necessary to achieve the desired function. On the other hand, it must consider all the effects of these activities on the environment (for example: toxicity, greenhouse gas, eutrophisation, depletion of natural resources). Finally, consideration of the environmental impact in design must follow an integrated design approach, where the environment is considered as one of the decision criteria, along with cost, quality, etc. (...) Eco-design can be based on methods and tools that can be (...) considered in two dimensions: those that make it possible to analyse the environmental impacts of a system - and thus to answer the question "which problems to tackle? and those that allow design teams to improve the environmental performance of a system under

design - and answer the "how-to" question"[2]

According to J. Pinto et al. [62]:

> "Eco-design will be considered like the design of a software which has the objective to obtain the smallest environmental impact when software is in its operation environment."

The definition proposed in this thesis has been constructed based on the previous definitions, both concerning eco-design in general and eco-design applied to software:

> Software eco-design is the integration of environmental constraints in all software development processes, in order to reduce the environmental impact of the software during its life cycle, this integration being individually adapted to the organization developing the software, stakeholders and the type of software developed.

The implementation of the eco-design must be carried out according to a global and multicriterional approach with the aim that a reduction of a negative impact of an environmental criterion does not increase the negative impact of another criterion or the same criterion elsewhere in the development process. The eco-design process of the software is composed of several parts. First, business planning and opportunities analysis; then the technical part of development; and finally, governance during and after development.

In this thesis, the focus has been put on the technical part of creating the software. This part is composed of several processes, established in stages, forming the software development cycle : requirement analysis, design, development, maintenance, operation and the end of life of the software (illustrated in figure 3.2). Concretely, five elements are to be integrated into the software development processes:

- Environmental assessment of the future software and his development
- Decision making decreasing the negative impact of the software and his development
- Inclusion in the organization's strategy
- Team awareness
- Transparency through communication to stakeholders

3.4 Other research topics relevant for software eco-design

As explained in the chapter 2, eco-design was not the only keyword used to find research on integrating environmental constraints into the software design

[2]own translation

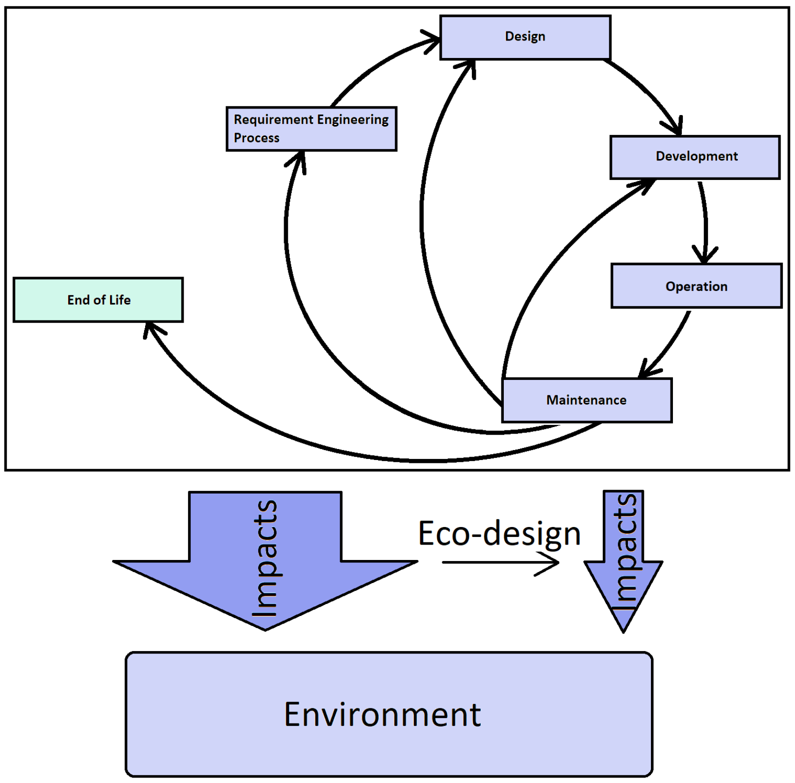


Figure 3.2: Eco-design aims to reduce the negative environmental impact of software development processes

process. Indeed, other searches, containing the words "Green" and "Sustainable" were carried out. The main motivation for using these concepts is that the terms "sustainable software" and "green Software", followed by "development" or "engineering" are far more used than "eco-efficient Software" or "eco-designed software" followed by "development" or "engineering". In order to understand how eco-design relates to these terms, the following section contains a brief explanation of these terms. In the first place, the definition of the terms "Sustainable software" and "Green software" proposed in the literature are reviewed, then their relationship to software eco-design is discussed.

3.4.1 Sustainable software

Several definitions of sustainable software were found, both related to sustainable development. A commonly used definition for sustainable development is the one from the Report of the World Commission on Environment and Development [2]:

> "Sustainable development is development that meets the needs of the present without compromising the ability of future generations to meet their own needs."

Three main dimensions are associated with sustainable development: society, economy and environment.

A first definition of sustainable software is from M. Dick et al. [6] :

> "(Green or Sustainable Software) is software, whose direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which has a positive effect on sustainable development."

A second definition is from L. Hilty et al. [12]

> "software whose development, deployment, and usage results in minimal direct and indirect negative impacts or even positive impacts on the economy, society, human beings, and the environment. A prerequisite for sustainable software is a sustainable development process, which refers to the consideration of environmental and other impacts during the software life cycle and the pursuit of the goals of sustainable development."

According to B. Penzenstadler et al. [13], software sustainability is divided according to four points of view :

- sustainability in the conceptual development process aspect,
- sustainability in the actual production of the software,
- sustainability in the system usage aspect and
- sustainability in the system maintenance aspect

In terms of sustainable software development, this definition, from C. Calero et al. [67], was found:

> ”Sustainable software development refers to a mode of software development in which resource use aims to meet product software needs while ensuring the sustainability of natural systems and the environment.”

Finally, sustainable software engineering is, from M. Dick et al. [14] :

> ”the art of defining and developing software products in a way so that the negative and positive impacts on sustainability that result and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and optimized.”

3.4.2 Green software

M. Dastbaz et al. [15] devote a chapter of their book to the legislation in the Green IT, a general observation of which is that there is no legislative act at any international level concerning Green IT, mainly because there is no clear and precise definition of this field. Nevertheless, a general definition they introduced is :

> ”The term green IT refers to various aspects of the protection of the environment, the most important principles being those of sustainability and conservation of natural resources.”

3.4.3 Relation between green, sustainable software and software eco-design

If the terms sustainable and green are compared, it can be seen that they have a well-known common base. Indeed, the definition established by the World Convention on Environment and Development in 1987 applies as well for the sustainable and green processes. One of the problems that has arisen from this definition has been the dual labeling of software that must both be designed in a sustainable way but also must be used in order to help sustainable development in a broad sense. The term ”green software” has gradually become a refuge to qualify all the softwares found under the original definition, whether they can be qualified more precisely or not. A common use of the terms green software engineering, or development, is in fact a joint use with the terms sustainable engineering, or development ([3][5][6][16]). E. Capra et al. [17], used the word ”green” to talk about energy efficient. They do it intentionally to have the best chances to find relevant paper during their research. Even though their approach is relevant, it nevertheless shows that the term ”green” is used in the scientific community, consciously or not, as a general term for not unnecessarily restricting the vision during research. By comparison, the word ”sustainable” would have been too restrictive, even if it shares the same basis as ”green”.

Even if the terms green software and sustainable are often used together, the environmental impacts to which they relate are not equivalent. J. Taina et al. [11], consider green software from a three levels perspective, based on the extent of the calculated impact of the software, and indicates that even though these three levels are applicable to green software engineering, not all of them are sufficient to be applicable to other fields such as software engineering for the planet.

K. Sierszecki et al. [16], argue that to limit the growing increase in the complexity of the term "green software", the terms "greening ICT or greening through ICT" have been introduced. However, this division can also be carried out for sustainability and we consider that this is more general version of "Sustainability for Software Engineering" and "Software engineering for Sustainability".

If the terms eco-design, sustainable and green are compared, it can be seen that eco-design is often substituted in the literature by concepts of green and sustainability. Indeed, eco-design is the integration of environmental constraints into the design process. These environmental constraints are mainly addressed in the computer science literature under terms such as ecologic, sustainable or green software development. Authors, such as R. Karlsson and C. Luttropp [74] even define eco-design as a way of integrating sustainability into design processes.

Thus, even if sustainability must respect three dimensions, most of articles dealing with sustainability for the software only do so through the environmental dimension. In fact, the energy efficiency was even the only feature dealt with until the beginning of this decade. The economic dimension has only been studied recently, which leads to a disproportion in the existing studies, according to P. Lago et al. [19]. A possible justification for this imbalance is that other aspects of sustainability are already included in existing practices, according to B. Penzenstadler et al. [18]. Another possibility is the nature of the energy dimension, which is seen as a global metric that allows the whole system to be controlled because all the components need energy, according to O. Philippot et al. [20]. The next possibility is that energy efficiency is much more tangible than the sociological impact for example, because the results of an improvement in software energy efficiency are directly visible and then appear more interesting to develop. A last possibility is the consideration that the environmental dimension influences others and that its development will have a positive effect on the system and its stakeholders that will launch a positive spiral of change in the other dimensions.

However, from a strictly theoretical point of view, the terms differ. Indeed, eco-design is not as restrictive, concerning the dimensions to integrate, as taking into account the sustainability in the software development. Indeed, eco-design does not require any improvement on the social and economic level. The approach taken by eco-design is therefore more optimistic, considering that the reduction of negative impact on the environment will have positive effects in the other dimensions. Sustainability for ICT is not equivalent to a more stringent version of software eco-design. Indeed, the implementation of eco-design must be done according to a global and multi-criterial approach. Even if the consideration of sustainability in development processes is in line with a multi-criterial approach, this consideration does not have to be a global approach, contrary to the eco-design approach.

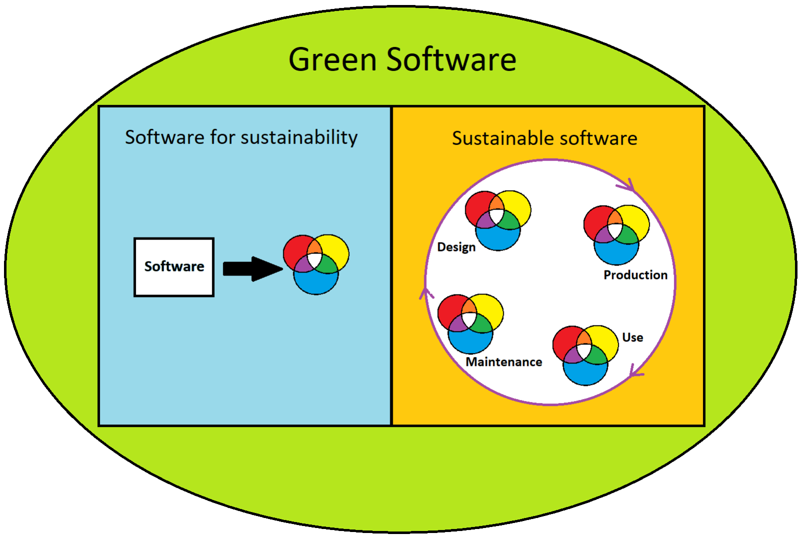


Figure 3.3: Our vision of the relationship between green software, sustainable software and software for sustainability

3.5 Conclusion

In conclusion there is no clear definition, no consensus, on the meaning of software eco-design, green software engineering, or development, and sustainable software engineering, or development.

Regarding the relationship between green and sustainable, "green software" doesn't only mean "energy efficient software" or "environmental friendly software" it is larger than that. Moreover, even if sustainability and green are not the same today, they share a common basis. "Green software" has become a bit of a buzz word that has made it very complex to pin down as it is a bit tote. For these reasons, green is not consider as a part of sustainability in this thesis, because green is often put at the same levels as sustainability when an author must mention this area. Green is, in this thesis, considered as a kind of cloud of idea containing the sustainable domain, as you can see in figure 3.3. This corresponds to the general idea that a green software is not a sustainable software, because it lacks the integration of the social dimension in its design, but that a sustainable software can be considered as a green software.

Regarding the relationship between eco-design and, green and sustainability, in practice, sustainability is mainly found in the literature under the single dimension of respect for the environment. This makes it possible to consider most of the research works on sustainability, and green, as relevant for eco-design.

Chapter 4

Review of existing literature surveys on software eco-design

This chapter analyze the previous scientific papers aimed at gathering current knowledge on software eco-design. The purpose is to present scientific papers aimed at gathering current knowledge, like literature surveys, concerning software eco-design.

Unfortunately, no scientific paper has been found on this subject yet. Since no scientific paper on the state of knowledge on software eco-design could be found, the following section presents several scientific papers on related topics. The selected literature review collected research works published between 1989 and 2016. A scientific paper on the practitioner's perspective in the field of green software engineering has also been included in order to get a view of what is happening in the industry.

The research works were browsed for several purposes. The first objective is to list and compare the structures used to organize their results, in order to use the most relevant structure possible in this thesis. The second objective is to list hot spots and cold spots in topics covered by current research that these works identified, in order to have keys to identify where the majority of the research is headed and where the breaches could be found in the field of software eco-design. Hot spots and cold spots of software eco-design are then refined in the next chapter where we perform our own literature review.

The research works are presented in a chronological order, with, for each research work a presentation of the structure used, hot and cold spots, and the authors' conclusions. Finally, a conclusion closes this chapter after the individual presentation of the various research works.

4.1 Software Engineering for Sustainability: topics and trends

Two surveys were conducted by the same authors on the same topic. The first survey was conducted by B. Penzenstadler et al. [13] and included research works published between 1991 and January 2012. The authors of this research work have conducted a systematic review of the literature on sustainability in software engineering. The research works considered by the authors relate to the development, the production and the utilization phases. However, the research works studied have in common that they analyze at least one area of application of sustainability.

The first structure chosen by the authors to organize their results is composed of four categories:

- Software design and implementation
- Software maintenance
- Software production (from the point of view of the resources needed to support it)
- Software usage

This structure was, in fact, not followed because too few valid research works were found at first. In order to not stop the research there, the authors have expanded their field of research and thus increased the number of valid research works. Their field of research having been modified, the structure used by the authors to classify the papers had to be modified and followed the application domains of research works which had this final form :

- Systems / Knowledge
- Disciplines
- Education
- Application / Implementation
- Technologies / Methods

Regarding hot spots and cold spots, according to the authors, the various research works focus on specific solutions and too few general studies on the field were conducted. In the same vein, the only reference framework that the authors found was focused on web applications. The last result the authors found is that all existing case studies are applications of methods by the authors of these methods themselves and are limited to specific application domains.

The second survey, written by B. Penzenstadler et al. [57] is an extended version, in terms of analysis, of the previous survey, having studied research works between 1989 and 2013. The reason for the conduct of this new version is that, at the time of writing the first research work, the number of research on sustainability in software engineering was so low that the authors had to extend their research very broadly, while two years later, the number of published research was such that a systematic literature review focusing on the original field of study was possible.

The structure used by the authors is composed of ten domains that replace the application domains used in the previous research work. The following data have been extracted from the research work, where, for each domain, the number of occurrences of research works has been added:

- Software Engineering & Life cycle (22)
- Systems Engineering & ICT (12)
- Services, Mobile & Cloud (10)
- Metropolitan Areas & Housing (9)
- Ultra large scale system Green Computing (7)
- Energy Efficiency (5)
- Business & Economics (5)
- Nature & Agriculture (5)
- Software Engineering Education (5)
- Mechanics & Manufacturing (3)

The main lesson formulated by the authors is that there is an imbalance in terms of the number of academic research (80% of valid papers) compared to industrial research (20% of valid papers).

Concerning hot spots, the authors used a method of topic modeling on all the valid research works that they gathered in order to find the most popular topics:

- Software Engineering Process
- Software Design
- Software Quality
- Software Requirements
- Models and Methods

Finally, despite the larger number of studies, the authors could not determine the most used approaches in research on sustainability in software engineering or in research on sustainability in the industry.

The results of authors were that the attention paid by the scientific community to software engineering for sustainability seems new but growing. The large number of fields of application and the nature of the research work (little number of experience or evaluation) show, according to the authors, that the research is not yet mature.

4.2 Metrics related to software "greenness": description and classification

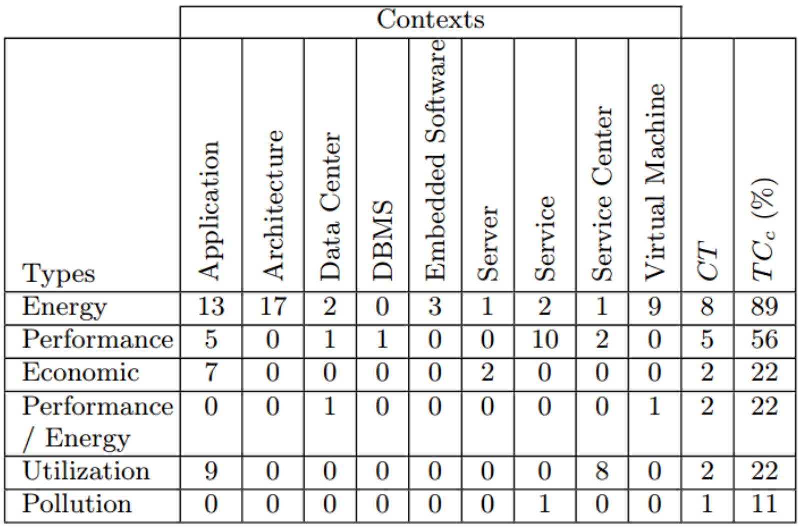
The third survey concerns software metrics, in the context of green software development, written by P. Lago et al. [19] and having studied research works published between 2001 and early 2012. This research work is representative of the amalgam between green and energy consumption, the metrics considered being all related to energy consumption.

Regarding the structure used by the research work, the authors used the hardware components (data center, server, resources, virtual machine) and software (services, applications, software architectures, database manager) to organize the valid research works. The types of metrics are classified regardless of the phase of the software development cycle in which they are located (although these phases are sometimes mentioned by the authors).

Concerning hot spots and cold spots identified by the authors, metrics aiming at the use of resources and performance appeared around 2010 in the literature and have therefore been the subject of less attention than those aiming at the consumption of the software, at the time of the realization of their survey. Their emergence is explained by the authors by the fact that a decrease in energy consumption could go hand in hand with a decrease in performance, which justifies the need for a balancing between these metrics. The software metrics in the development and maintenance stages are the most represented during the decade 2001-2012. The software architecture in the software design stage is the second measurement context in terms of the number of metrics, although, all of these metrics are based on a single research work. The third and fourth most commonly used contexts are services, in the execution and maintenance development stages, and service executions in the data centers, with metrics growing together with interest in service-oriented software development, starting in 2008. Following the same trend, metrics applied in the context of virtual machines have also been developed since 2008. Finally, measurement contexts such as servers and embedded software have received so little attention that it was not possible for the authors to determine trend in the studies. Metrics to measure or evaluate resource use and costs of software design, life cycle phases, or green practices have not been studied in contexts such as data Centers, virtual machines or DBMS. The detailed results are visible in figure 4.1. One last important piece of information is that out of the 96 metrics, 33 are used for estimates (17 of which come from the same research work), that is, they are used upstream of the software development process, typically in the phase of elicitation of the requirements. 67 other metrics are used to perform a measurement, thus downstream of the development process. Finally, 4 metrics are common to both categories.

4.3 Green Software Development: evolution of primary studies

The fourth survey is about green software development in Collaborative Knowledge Management Environment, written by R. Abdullah et al. [58] and studied



	Contexts										
Types	Application	Architecture	Data Center	DBMS	Embedded Software	Server	Service	Service Center	Virtual Machine	*CT*	TC_c (%)
Energy	13	17	2	0	3	1	2	1	9	8	89
Performance	5	0	1	1	0	0	10	2	0	5	56
Economic	7	0	0	0	0	2	0	0	0	2	22
Performance / Energy	0	0	1	0	0	0	0	0	1	2	22
Utilization	9	0	0	0	0	0	0	8	0	2	22
Pollution	0	0	0	0	0	0	1	0	0	1	11

Figure 4.1: Relation between metrics types and contexts - extract from [19]

research works published between 2010 and September 2014. The authors identified 37 papers related to Green Software Development (i.e. containing at least the terms "green software development" in the title or body of the research work).

Regarding the structure used to classify these different research works, the authors used nine categories. These categories can be grouped into two groups, one grouping the categories of research works dealing with the subject in a "horizontal" manner, without distinguishing between the types of software and the other grouping the categories of research works treating the subject in a "vertical" manner, in focusing on execution context of the software. Horizontal categories:

- Software Development Life Cycle (11)
- Software Metrics category (9)
- Environmental Management for Software Company (2)
- Software Definition Approach (1)

Vertical categories:

- Software for End-User and System (5)
- Software for Mobile Platform (4)
- Software for Cloud Environment (2)
- Software for Legacy Systems (2)
- Software for Digital Media Platform (1)

Regarding to hot spots and cold spots, the authors' results are that the majority of the research works studied focus on the software development cycle as well as the effectiveness of green software development. The reasons given by the authors to explain this majority are that the respect of the requirements of the stakeholders would be the priority during the software development. However, it is during these phases of development that these requirements are introduced into the software. The metrics for their part, make it possible to check the quality of the software, i.e. software compliance in relation to these requirements. The authors explain that the most used quality metric is energy efficiency by the fact that this measure would be the one with the most direct link with the decrease in the energy consumption, which would henceforth be the most sought reduction of impact.

The authors conclude that the researchers must target different areas because the present ones are limited.

4.4 Green and sustainable software: software engineering aspects

The fifth survey was written by Marimuthu C. et al. [59]. The authors of this research work studied research works published between 2010 and May 2016. The purpose of this survey is to provide an update of the results of the other literature reviews (including B. Penzenstadler et al. [13] and B. Penzenstadler et al. [57]) as they focused on research works published before 2014. The authors decided to carry out a study on the green and sustainable software at the same time, and not separately as for other surveys. Another difference with other surveys is that they used a snowballing procedure to get a view of the most important research works and detach themselves from search engine results. Finally, according to the criteria defined by the authors, only research works dealing with ways to improve the environmental impact of software through software engineering were studied.

Regarding the structure used, the authors identified two categories of research, research aimed at limiting the environmental impacts of the software development life cycle phases, including the use and end-of-life phases of the software, and research aimed at producing software with limited impact on the environment. The authors therefore oppose research aimed at development processes to those aimed at the developed product. However, they found that the distribution of research works is not fair between these two categories. Indeed, the authors ranked 9% of the items found in the first category and 91% in the second category. The topics of the research were classified according to several categories, the authors having followed the structure used by the Software Engineering Body of Knowledge:

- Software requirements (18)
- Software design (2)
- Software construction (4)
- Software testing (17)

- Software maintenance (12)
- Software engineering process, when a new process is proposed by a research work (7)
- Software quality (8)
- Computation foundations (6)
- Engineering foundations (8)

Respectively, chapters 1, 2, 3, 4, 5, 8, 10, 13 and 15 of the SWEBOK.

Regarding hot spots and cold spots, the results of the authors were as follows: the number of research works increased from 2012, with nevertheless a fall in 2015. The review of the literature was carried out in 2016 so the authors could not determine whether this fall was temporary or whether it would continue. The most common type of research work is the "validation research". The most covered topics are the testing phases and the requirements phase. On the other hand, the least common type of research work is "experience paper", while no "opinion paper" was found by the authors. These results show that the field of research was not yet mature because several important topics were still not been covered by several types of research (this is visible in figure 4.2 where it can be seen, for example, that software design has not been the object of experience paper or evaluation research). We can also see in the figure 4.3 that the type of contribution are mostly academic, with industry appearing to lag behind in this area. Finally, out of 14 tool papers found by the authors, 12 relate to the test phase. This means that research is lacking to allow industry developers to use tools in other phases of the development cycle, especially during the design and construction phases.

The authors concludes the same way as the others literature reviews, researches are too focused on certain topics and researchers must work on the least explored topics.

4.5 Green software engineering: practitioner's perspective

Finally, the last research works studied in this section is a study of industry practices, written by I. Manotas et al. [60] in the field of green software engineering, realized with the aim of obtaining a view of what is being done today in the industry. The authors conducted interviews and surveys with industry developers to obtain qualitative data and validate them with a representative number of participants. An important remark concerning this research work is that certain elements must be taken into account when interpreting the results. In the first place, all the practitioners targeted to answer the survey came from ABB, Google, IBM and Microsoft, which may have led to a community bias in the survey. Then, even if the theme was "green software engineering", the authors have limited themselves to energy management as a criterion of selection of practitioners to respond to interviews and surveys, and have avoided the practitioners handling any other environmental considerations.

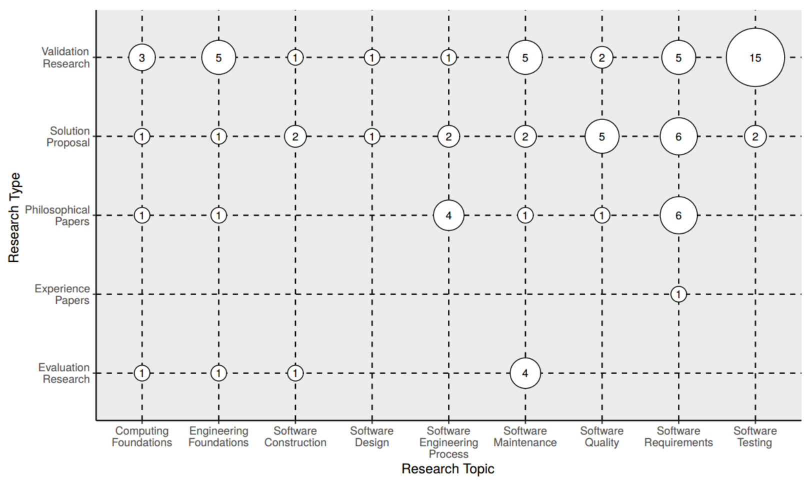


Figure 4.2: Research type by research topic - extract from C. Marimuthu et al. [59]

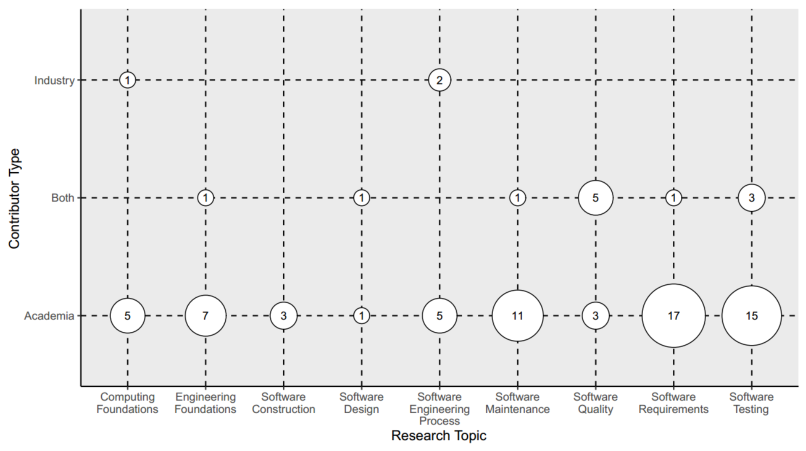


Figure 4.3: Research contributor type by research topic - extract from C. Marimuthu et al. [59]

The structure used by the authors to organize the results of the research work is based on the phases of the life cycle software development (requirements, design, construction, maintenance), with additional attention on activities of finding and fixing energy issues, which is not present in the other surveys studied.

Hot and cold spots are seen here according to the maturity of the stages of the green software engineering. First of all, for the requirement phase, the platform in which energy requirements are most often elicited is the mobile platform. Some of the reactions of the practitioner are representative: "the goal is to accomplish something without making the user annoyed about battery drain", "turn-by-turn guided navigation should not drain more battery than a car can charge". In the case of embedded software and data centers, practitioners say they rarely take energy consumption into account when working on these types of platforms. An explanation of this result for data centers is summarized by one of the interviewees: "Our main concern is market share and that means user experience is a priority. We can be more efficient to try to cut costs, but we do not charge by energy. So we tend to focus on other things like performance or reliability". Regarding the construction phase, developers are at 80% aware that their code has an influence on the energy consumed but only 30% of them feel they have a good intuition about the best practices to apply. However, some of them think they can not handle the problem in many cases: "Dependencies on libraries (...) that are inherently inefficient can make a difference in life", "Problems do not always reside in the app code. The hardware often does not support polling, idle, or other modern commands to minimize energy use". Regarding the maintenance, the main result is that developers are for most of them, almost completely unaware of potential problems of energy efficiency during this phase.

The authors' results are that the inclusion of the green in the software engineering is more often approaching common sense with the user experience than a form of true ecological consideration. However, according to the authors, the vast majority of developers would be willing to compromise with other non-functional requirements and use some tools to preserve energy consumption.

4.6 Conclusion and positioning of our work with respect to existing ones

In conclusion, regarding the structures observed in the literature reviews used to classify research works, collected in similar fields to a certain extent with our field of study, the classification methods are depending on the type of research works, the subject treated or the research method. Concerning the subjects treated, it is possible to classify them according to two groups, the research aiming at the processes of developments in the broad sense and the research for the software products with the aim of reducing its negative environmental impact. In the research group on broader development processes, it is possible to organize the research according to several parts:

- Software Development Life Cycle phases :
  - Requirements
  - Design

  - Construction / Implementation
  - Testing
  - Production
  - Maintenance / Research and correction of energy consumption issues
  - Software End of life

Regarding hot spots and cold spots in the literature, it is difficult to make a decision given the fact that surveys do not have the same areas of research. However, one result common to all literature reviews is that there is not enough research in general and the researches that exist are not sufficiently diversified concerning research topics. In addition, issues such as lack of industry involvement in research and lack of validation of theories are present. These two issues are all the more critical as the industry is asking for tools and methods to manage the energy consumption of software developed, at least in the design phase. Finally, the last two issues mentioned are the existing amalgam between reduction of negative environmental impacts and energy efficiency, as well as the lack of training and awareness of developers in the area of development and software reduction of environmental negative impacts.

The positioning of this thesis in relation to the research works summarized in this chapter is as follows. First, in terms of scope, unlike the previously studied surveys, this thesis does not limit its search to "sustainability development" or "green development", but considers both visions at the same time. The dimension of "eco-efficiency development" and "environmental development" have also been added. As with research works of B. Penzenstadler et al. [57], R. Abdullah et al. [58] and Marimuthu C. et al. [59], the scope of this thesis addresses all stages of the software life cycle from the perspective of software creators and no selection is made on the types of platforms or software covered by the articles. Then, in terms of objectives, like all other research works of this chapter, the primary goal is to gather the information and present it in such a way as to quickly obtain a picture of the state of literature. However, unlike the other research works, this thesis focuses mainly on environmental impacts: their identification, methods to estimate them and solutions to reduce them.

The research works summarized in this chapter are used in the following manner in this thesis. First of all, the conclusions on their structure and on the hot and colds spots are used as help for the realization of the survey. Then, research works collected by B. Penzenstadler et al. [57] and P. Lago et al. [19] are used as a basis for searching for relevant research works for this thesis (unfortunately research works collected by Marimuthu C. et al. [59] were not browsed due to time constraints).

Chapter 5

Survey on Software Development Life Cycles phases

5.1 Introduction

The purpose of this section is to present in a structured way data extracted from research works, collected trough the process described in chapter 2. Throughout this section, two structures are used: a presentation structure and an analysis structure.

5.1.1 Presentation structure

The reporting structure adopted here is based on the phases of the software development life cycle (Requirement engineering process, Design, Development, Maintenance), completed with additional phases: operation of the software and end of the life of the software. These phases cover the entire life of the software, from the first sketches of its design to its withdrawal. This structure was chosen because it corresponds to the characteristics of eco-design: global and multi-criteria. It also corresponds to what has been done previously in the other surveys.

5.1.2 Analysis structure

The analysis structure adopted is the same for all phases and is composed of several things. First, we have attempted to associate each research work with one or more *environmental impacts* it addresses. Next, we identified, among the research works, those who develop *calculation models* for these impacts. Then, we sought *solutions* that the authors propose in order to reduce negative environmental impacts. Finally, we drew *conclusions* about the observations we could make from this information.

5.1.3 Software life cycle phases

The different phases that will be presented in the rest of this survey are the following.

Requirement engineering The first phase considered is the requirement engineering process. According to P. Bourque et al. [124] who wrote a guide to the SWEBOK, requirements engineering "is widely used in the field to denote the systematic handling of requirements" and "Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem". This phase is important because it is during this one that the bases of the developed software are defined. Regardless of the type of development process used (Prototyping, Spiral, Waterfall, Agile,...) there's always a moment when "it starts".

Design The second phase considered is the design process. According to P. Bourque et al. [124] "software design is the software engineering life cycle activity in which software requirements are analyzed in order to produce a description of the softwares internal structure that will serve as the basis for its construction". During this phase, several abstract software elements are defined. Non-functional requirements defined during the requirement phase are used to design the software structure, architecture and user interface. Abstract description are also defined during this process. Finally, algorithmic and data structure choices are made.

Development The third phase is the development phase, also called construction phase. According to P. Bourque et al. [124] "software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging". To help developers in this task, they use development environment (IDE) as well as API and libraries to facilitate and decrease the production time and effort.

Maintenance The maintenance (or support) phase is composed of a series of evolutions of the software, once the production of this one is finished. According to P. Bourque et al. [124] software maintenance is "the totality of activities required to provide cost-effective support to software. Activities are performed during the predelivery stage as well as during the postdelivery stage". This support can be caused by the detection of a bug or changes in requirements, or the environment. The end of the maintenance phase comes when it is planned or when the maintenance becomes too expensive or too risky.

Operation The operation phase is the phase during which the software is distributed and used. According to ISO/IEC 15288 [125], "The purpose of the Operation Process is to use the system in order to deliver its services. It assigns personnel to operate the system, and monitors the system services and operator-system performance". This means that it is during this phase that the software will permit the organization to earn money and will cause energy consumption through the use of resources (CPU, RAM etc.). Moreover, it is during this phase that errors are reported. The operation phase starts when the software is released and ends when the software is no longer used. The use of the software starts at the same time as this phase but may exceed the end of the distribution and even the end-of-life phase of the software. Similarly, the distribution of the software can be much shorter than the operation phase, for example in the case of custom softwares. Except for these general elements, it is difficult to give a general definition of the operation phase because it differs according to the software developed.

End of life The last phase considered is the end of life of the software. According to ISO/IEC 15288 [125], "The purpose of the Disposal Process is to end the existence of a system. This process deactivates, disassembles and removes the system and its waste products, consigning them to a final condition and returning the environment to its original or an acceptable condition". We identified three possible points of view of the beginning of the end of life phase. The first point of view is that of the designers, who considered that the software life is finished when the development of the software is finished. The second point of view is that of the maintainers, who considered that the software life is finished when the maintenance of the software is finished. Finally, the third point of view is that users, who considered that the software life is finished when they no longer use the software. This third view has already been addressed in the previous phase. When a software has reached these three stages, the end of life phase begins. After this phase, the life cycle is over.

5.1.4 Structure of this chapter

Before presenting works relative to the different phases of the life cycle, we first present the works treating the life cycle of the software as a whole and not a particular phase (see section 5.2). This presentation allows us to present these research works in one place. However, we also present these research works in the different phases, when they are relevant. Thus, the reader who goes through a single phase can find all the information concerning this phase in a single place.

5.2 Survey of Research works targeting more than one software life cycle phase

This section aims to present research works that do not target a single phase but several phases of the software life cycle. Indeed, several authors have dealt with environmental impacts at the level of the software life cycle as a whole and not only at the level of a particular phase.

We have decided to group these research works in a separate section in order to present them in their wholeness. The research works are presented in a chronological order.

In 2006, J. Pinto et al [62] have made a conceptual map in order to develop, in the future, a model of software quality. Concretely, they carried out a preliminary research concerning the development of a quality model based on

ISO/IEC 9126 that can be used for software eco-design. The authors' hypothesis is that the use of quality models relating to eco-design in general, like ISO 14040-14043 on software, is not excluded. The authors then estimate that if these types of model incorporate environmental constraints, then they could be used to assess the direct and indirect environmental impacts of the software. Regarding the quality factors to be used among the existing quality factors, the authors propose that environmental impacts be considered as a combination of reliability and efficiency.

In 2010, J. Taina [52] evaluated the CO2 emissions of several phases of the life cycle of a software (requirement, development, beta testing, delivery, use and maintenance). In order to evaluate the carbon footprint of software life cycle phases, the author used direct estimates (for the carbon footprint of a piece of paper for example) and indirect estimates, based on energy consumption and estimation of CO2 emissions per kWh in Finland. Then, for each phase, assumptions about the progress of the phases were made, such as the number of meetings and the number of people present at these meetings, the use of paper or the number of features. Employee commuting has also been taken into account. Two original phases are taken into account in this model: the beta testing phase and the delivery phase.

The beta testing phase was evaluated by considering, among other things, the number and duration of the test cycle, the number of beta testers and the duration of use of the software. It is interesting to note that the results of the carbon emission model of this phase are significantly higher than the estimates of the other phases.

For the delivery phase, the author considered that the software would be sold in dematerialized version on an online platform. This phase was therefore evaluated in four stages. First, an estimate of the upload and download of the software between the publisher and the different sellers. Then an estimate of the sales process and finally an estimate of the borrowing of the download by the user.

In 2011, S. Shenoy et al. [69] suggested practices that reduce the environmental impact of, the software development processes and the software products. Two important remarks can be made about this article. First of all, the authors work in the industry for Siemens Information Systems Limited. Secondly, no experiments or empirical results have been presented to prove the usefulness of the good practices mentioned in this paper.

The authors established their model through the following software development phases: requirement, design, implementation and retirement, but did not systematically present solutions for reducing the environmental impacts of the software development processes and the software products.

First, software-level solutions are proposed in the requirements and implementation phases.

During the requirement phase, the authors highlight a first potential paradox of environmentally friendly software design: the software developed should be optimized for the supporting hardware running the software but also be usable on the maximum number of supports, legacy or future, in order to maximize the service life of the equipment. The authors also argue that the features of the developed system should include concepts that limit energy consumption, such as not unnecessarily using bright colors that consume more power or switch to low power mode when the software is not used.

During the implementation phase, the authors argue that it would be better to adapt the company's development processes so that they can integrate the sustainability constraints. The authors also recommend limiting as much as possible the coupling between the software developed and a type of hardware, in order not to link the lifetime of the software to the one of the hardware. Therefore, it would be better to avoid API dependent on a particular type of hardware. Finally, the authors support the use of the pair programming as well as the use of tools for programming and code generation to minimize programming errors.

Then, the solutions at the level of the development processes were presented in the design and end of life phases.

During the design phase, the authors argue that the design should be as simple as possible, to reduce the work time and time of explanation to the stakeholders. The design elements should be reused to the maximum, to also reduce the time of conception. The use of plug-in architectures is therefore, plebiscited by the authors. Finally, changes in design should be avoided to the maximum as these can lead to significant re-design efforts.

During the end of life phase, the authors argue that it would be interesting to keep the source code and all the information that could be useful in future development.

In 2011, S. Naumann et al. [5] have developed a conceptual model that aims to assist in the development, choice and use of sustainable software. This model is much more cited than the other models discussed in this section (181 citations, against 61 for the model S. S. Mahmoud et al. [70] and 46 for the model of S. Shenoy et al. [69] [1]) and is often presented as a reference in the field. The model is divided into four parts. The first part aims to allow an evaluation of the impacts of the software on the environment. With this in mind, the authors defined the causes of impact in each phase of the software life cycle. The second part concerns the relevant metrics for measuring impacts. The third part aims to define an organizational framework for sustainable software development. Finally, the model ends with a part listing the relevant tools for the stakeholders to achieve sustainability.

In 2013, S. S. Mahmoud et al [70] have proposed a sustainable software design model. The authors' attention is here on the modeling of a new development process directly integrating the constraints related to the development of sustainable software as well as the software tools to meet these constraints.

The limit of the scope of the article is that the authors have focused exclusively on the development phases of the software and have not approached the software operation phase.

The article has the advantage of taking into account all phases of the generic software engineering process, but yet omits the use and disposal phases. In addition, the authors consider the test phase as an important phase in integrating sustainability into software development. Another point is that the proposed software development model is a mix between traditional development and agile

[1]in August 2018

methods.

Finally, an element that separates this developed model from the other software development models is the addition of a green analysis phase which aims to evaluate the software's ability to respect the environmental constraints that have been set for it. The proposed model being inspired by Agile methods, this stage is performed at the end of each iteration. In the words of the authors "This stage acts like a testing stage but for energy efficiency" but they do not justify the need for an additional stage, when verification in the testing phase could be enough.

In 2014, Eva Kern et al. [53] presented a method for calculating the CO2 emissions of the software development and applied this method to a real project. In order to evaluate CO2 emissions, they focus on several things. First, they focus on the company infrastructure's CO2 emissions per year based on office space, whose sources are air conditioning and automatic equipment such as elevators and lighting. To estimate these emissions, they used the average data for these emissions in Germany. They also estimated CO2 emissions from the company's IT infrastructure, this time using the power consumption data for the manufacturers and converting them with the average CO2 emission for electricity generation in Germany. Missing information on equipment consumption is roughly estimated. Finally, CO2 emissions per year are multiplied by the number of years of software development.

Next, emissions related to employee commuting in the design, development and maintenance phases are estimated. For this they use the average number of working days per year per person. Using typical distances between home and work and using the appropriate transport, they estimate average CO2 emissions per year per person. From this, they estimate CO2 emissions from transport during software development by multiplying the estimate by the number of months men estimated for software development. This same method is also used to estimate the commuting during the other phases of the life cycle.

Concerning the emissions of the operation phase, the authors mentioned some interesting parameters that can influence them: the type of software, the estimated length of use, the need to buy new equipment or not, the amount of users as well as their type. However, given the very limited amount of information the authors have about these parameters, they decided to neglect the emissions from the operation phase. In return, they estimated the number of servers (and personal computer) that should be necessary during the operation phase to exceed the CO2 emissions of the development and maintenance phases for five years.

5.3 Survey of software eco-design phase by phase

In this section, the survey results are presented phase by phase as explained at the beginning of this chapter. For each phase, the analysis is identical and follows the structure also explained at the beginning of this chapter. Specifically, for each phase, the environmental impacts identified by the literature, existing models to estimate them and solutions to reduce them are detailed. Each phase ends with a conclusion.

5.3.1 Requirement engineering

5.3.1.1 Impacts

The impacts identified by authors who dealt with this first phase can be classified into three types of impacts. This classification is used to structure the presentation of research works within the following subsections. Below is a brief explanation of these.

Requirement engineering processes execution on its own carbon footprint J. Taina [52] has identified that the carbon footprint of the requirement engineering phase relates to the electrical consumption and heating needs of the buildings, the electrical consumption of the infrastructure needed for the requirement process, as well as the commuting to work of the stakeholders.

Requirements, and business, choices on software and business sustainability J. Cabot et al. [55] and M. Mahaux et al. [47] considers that the software's functionalities would have an impact on the ecological footprint of the business process in which the software is used. The integration of software into the user's business processes should therefore be taken into account during the requirement analysis, as well as the future environmental impact of the software product itself. C. Sahin et al. [26], C. Atkinson et al. [24] and M. Mahaux and C. Canon [71] considers that excess in functionality and quality levels are responsible of excess in software's energy consumption. B. Penzenstadler et al. [18] considers that quality attributes have an impact on software and business sustainability.

Fitness for purpose quality on software lifetime M. Mahaux and C. Canon [71] consider that the requirements engineering phase have an effect on the software lifetime by inferring on the quality of the software and therefore on the satisfaction of users, who will potentially have less desire to get rid of software if it meets their need.

5.3.1.2 Calculation models

A Requirement engineering processes execution on its own carbon footprint

A.1 Calculation of the carbon footprint of the requirements engineering process

The carbon footprint has been measured by J. Taina [52], who evaluated the CO2 emissions of several phases of the software life cycle (requirement, development, beta testing, delivery, use and maintenance) of a software. In order to evaluate the carbon footprint of software life cycle phases, the author used direct estimates (for the carbon footprint of a piece of paper for example) and indirect estimates, based on energy consumption and estimation of CO2 emissions per kWh in Finland. Then, for each phase, assumptions about the progress of the phases were made, such as the number of meetings and the number of people

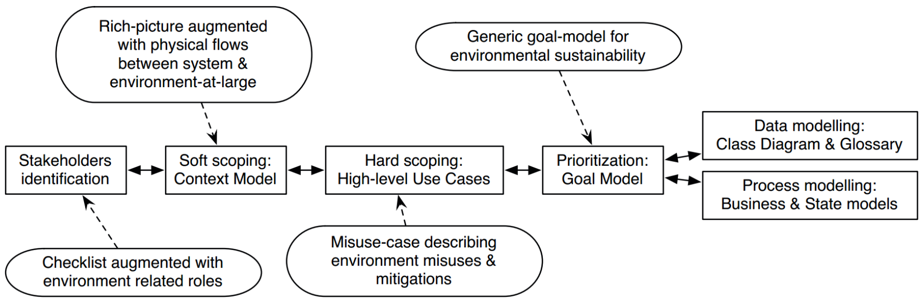

Rich-picture augmented with physical flows between system & environment-at-large
Generic goal-model for environmental sustainability
Stakeholders identification
Soft scoping: Context Model
Hard scoping: High-level Use Cases
Prioritization: Goal Model
Data modelling: Class Diagram & Glossary
Process modelling: Business & State models
Checklist augmented with environment related roles
Misuse-case describing environment misuses & mitigations

Figure 5.1: Representation of the flow of the various activities carried out - extract from M. Mahaux et al. [47]

present at these meetings, the use of paper or the number of features. Employee commuting has also been taken into account.

5.3.1.3 Solutions

A Requirements, and business, choices on software and business sustainability

Solutions identified in research works dealing with this impact are the following:

1. Adapt the work-flow used to elicit requirements, to incorporate sustainability constraints
2. Build a database containing the relationship between decisions made on requirements and their environmental impacts
3. Use GreenSLAs (Green Service-Level Agreements) to formally express non-functional sustainability needs
4. Design software that can be customized for the customer
5. Add sustainability as a quality attribute in software engineering standards

A.1 Adapt the work-flow used to elicit requirements, to incorporate sustainability constraints

The experience report of M. Mahaux et al. [47] is often cited as a reference when talking about sustainability requirements because it was the first, and the only one to our knowledge, to explore the issue in depth. The article is not intended to define a general method but rather to explore.

In their article, the authors adapted a work-flow used to elicit requirements by incorporating environmental considerations. Figure 5.1 represents the flow of the various activities and their added environmental dimension. It is important to note the presence of double arrows in this diagram meaning that the process should not be carried out in a sequential but iterative manner, with possible ”backtracking”.

The different activities carried out are as follows.

The first activity relates to stakeholders identification, where the authors have taken into consideration actors opposed to taking sustainability into account, with the aim of obtaining results closer to reality. The authors found this addition useful and easy to set up for teams wishing to take sustainability into account.

The second activity relates to defining the soft scope (the system and its environment). For this purpose, a rich picture context model was realized, this one made it possible to identify all the elements other than the software having a potential impact on the sustainability. The environmental constraints appear in this model via bold arrows additions. According to the authors, this modification of classical context models was sufficient for users of the model to define the scope and context while taking into account environmental constraints. Nevertheless, the authors highlighted a limitation of the bold arrows, which could only explicitly represent half of the environmental constraints they had identified, the other half had to be elicited by discussions with stakeholders.

In the third activity, they defined the hard scope (the high level solution). For this purpose, they used the rich picture model to create a set of use case diagrams. However, these use cases made by the authors did not incorporate any environmental constraints of the rich picture model. In order to integrate environmental constraints, the authors carried out a misuse case analysis to identify the sources of environmental impacts. Nevertheless, according to the authors, the misuse case analysis method has identified almost the same constraints present in the rich picture model.

The fourth activity relates to a goal analysis. In this analysis, they have integrated respect for environmental constraints as goals to be achieved. In this perspective, they did not use a different method than the traditional goal analysis methods.

Finally, the latest data and process modeling activities were performed without taking into account environmental aspects, as they did not seem relevant for the authors to do so.

J. Cabot et al. [55] consider that softwares should integrate, from the requirements analysis phase, all information concerning the software but also the business, making it possible to limit the ecological footprint of the operating phase. The authors defined and tested on one case a preliminary method using only the I* language in order to take into account sustainability constraints. The originality of the solution is to take into account the constraints of the business in which the software will be used and not only the software itself. The result the authors obtained is that this approach effectively allows decisions to be made that have the potential effect of limiting the ecological footprint of the software product and its environment of use.

A.2 Build a database containing the relationship between decisions made on requirements and their environmental impacts

Another solution would be to build a database containing the relation between requirements decisions and their environmental impacts. Such databases do not exist yet even if it is theoretically possible to build them. C. Sahin et al. [26], propose a general way of characterizing the high-level changes to be made by a engineer so that his software consumes less energy. After identifying the consumption of different code patterns, they propose to link these patterns to the Use Cases and Class Diagrams from which they are derived. Finally, they propose to link these models with the properties of the design (reactivity, security...). Some drawbacks are that this method has only been tested at the level of the consumption-code mapping and not at the consumption-Design Properties level. Thus, the consumption-code mapping was done by using extremely accurate hardware equipment to measure consumption, which allowed the authors to use pieces of code, whose execution are easily detectable, in order to separate the other parts of the code to be studied. This limits the possibility of measuring the consumption of a software by someone who does not have this equipment, which limits the use of this method to measure easily and systematically the consumption of a software. Fortunately, research is progressing and it is now possible to measure, with a software tool, the consumption of a program written in Java, according to H. Acar et al. [27]. We could question the validity of the results on the basis of the material used for the measurements, but that does not prevent these results from being generalizable.

A.3 Use GreenSLAs (Green Service-Level Agreements) to formally express non-functional sustainability needs

As regards to the difficulty of having different stakeholders objectives to converge, C. Atkinson et al. [24], propose the use of GreenSLA (Green Service-Level Agreements) to formally express the non-functional needs related to sustainability. This would have the effect of improving the communication between the parties on the objectives of respect for the environment. The initial idea of adding a green dimension to Service-Level Agreements is that usually the specifications are defined so that the software have the best performance possible, while they do not necessarily need such a performance. The GreenSLA should be able to reduce negative impacts (in terms of energy consumption during the use of the software), by formulating specifications closer to the real needs of users, with the minimum of excess in functionality and quality levels. There are two main problems with the proposed SLA formalism. The first one is that the constraints formulated are very general and there is no indication that the methods that will be put in place to respect them will not worsen the negative impacts elsewhere in the development process. The second problem is that only programmers really understand the SLA formalism. Still, they are designed to ensure that all stakeholders agree on the software requirements. One response to this limitation is the use of UML models to communicate with stakeholders of which the incorporation of environmental aspects consists of the addition of ecological constraints in certain classes of diagrams.

A.4 Design software that can be customized for the customer

M. Mahaux and C. Canon [71] consider that the software must be designed to limit excess functionality to a minimum. The authors consider that it would therefore be interesting to design software that can be customized for the customer, allowing only the necessary functionalities to be chosen from the requirement engineering phase, in order to try to limit the environmental impact of the entire rest of the software life cycle. However, in this research work, the authors remain at the intuition stage and do not develop this subject any further.

A.5 Add sustainability as a quality attribute in software engineering standards

An other interesting proposal is to draw inspiration from what has been done when dealing with security and safety requirements in the past. This is the point of view developed by B. Penzenstadler et al. [18]. Indeed, in the not-so-distant past, just before the 2000s, safety and security were not considered as major non-functional requirements and it was only after several years of development that they were considered as such. Safety and security became increasingly important as they satisfied a need and were therefore accepted by the community before being added to the software engineering standards.

B Fitness for purpose quality on software lifetime

B.1 Do everything possible to obtain the best fitness for purpose quality

M. Mahaux and C. Canon [71] consider that the requirements engineering phase should seek to increase the lifespan of software so that the impact of their development process (including the end-of-life phase) would be relatively less compared to the entire software life cycle. They justify this by the importance of the requirements engineering process in designing software that best meets the needs of the customer and therefore whose duration of use, the lifespan of the software, is the longest. The authors here hypothesized that a longer development time implies more effort and therefore more environmental impact. However, in this research work, the authors remain at the intuition stage and do not develop this subject any further.

5.3.1.4 Conclusion

Structure coverage

The table 5.1 class for each identified impact, the authors having proposed a calculation model or a solution for this impact. Based on this table, our observations on the state of the literature are as follows:

First, we can classify the environmental impacts studied in the literature for the requirement engineering phase into three categories:

- Requirement engineering processes execution on its own carbon footprint
- Requirements, and business, choices on software and business sustainability
- Fitness for purpose quality on software lifetime

Secondly, we can observe that no article offers both an estimation model and one or more solutions for a given environmental impact. Within the framework

Impacts	**Calculation models**	**Solutions identified**
Requirement engineering processes execution on its own carbon footprint	J. Taina [52] (Problem identification and solution)	
Requirements, and business, choices on software and business sustainability		J. Cabot et al. [55] (Case reports) M. Mahaux et al. [47] (Case reports) C. Sahin et al. [26] (Problem identification and solution) C. Atkinson et al. [24] (Problem identification and solution) M. Mahaux and C. Canon [71] (Commentary) B. Penzenstadler et al. [18] (Commentary)
Fitness for purpose quality on software lifetime		M. Mahaux and C. Canon [71] (Commentary)

Table 5.1: Table of research works by impacts identified and by contribution

of our research, we have not collected any research works allowing, for a given environmental impact, to have an estimate and a solution allowing the reduction of this impact.

Finally, we can observe, by following the classification defined in Chapter 2, that most of the research works are not based on research and mostly are in the form of commentary. All three research-based research works ([24] [26] [52]) are in the form of problem identification and solution. We have not found any validation or evaluation research.

Impacts

Concerning the impacts studied, we found no mention of potential effect of the development of the software on the hardware (already existing or not) or other software in its environment. In other terms, if the software developed is strongly linked to an infrastructure, the consequences of its development and operation could be to have to get rid of an existing infrastructure and to produce a new one and this should at least be identified and estimated. Similarly, replacing existing software could also have environmental impacts if the old software had different resource requirements, both in terms of infrastructure and in informational term.

Calculation models

Concerning impact calculation models, since no software component has yet been developed in this phase, these seemed limited to the impacts of the work done in the phase. However, it might be possible to create models to estimate the environmental impact of decisions made during this phase, if more information about the concrete environmental impact of these decisions on subsequent phases of the software life cycle were available. These estimation models could be based on bottom-up mapping as defended by C. Sahin et al. [26].

Solutions

Regarding solutions, no research explores the potentially positive impact mentioned by M. Mahaux and C. Canon [71], which is that requirements engineering, when properly performed, would increase the software's life expectancy. Moreover, this type of research would then provide leads to validate or not the hypothesis that a longer lifespan of software is beneficial to the environment.

Some authors have identified challenges for implementing solutions to reduce the environmental impact.

One of these solutions would be to limit excess during elicitation of requirements process. Nonetheless, according O. Shmueli et al. [72] who conducted a literature review on research on excessive software development practices and classified the excesses according to three objects (resources, plan and needs), this area of research is almost not addressed in the literature. In addition, still according to O. Shmueli et al. [72], no major research was conducted to find out the impacts of these excesses, their causes as well as the means to deal with them. Nevertheless, the authors were able to identify, by browsing existing research, that stakeholders, just like managers, have a role to play in the causes of these excesses.

Finally, we have not found any research estimating the results or the disadvantages of the solutions presented. However, an interesting result of pooling these studies is that we can see that they all offer solutions that can potentially work together. We can therefore imagine that future research could study the effectiveness of using the different solutions together.

5.3.2 Design

5.3.2.1 Impacts

The authors who dealt with this second phase identified three types of impacts, all of which targeting the energy consumption during the operating phase.

Software modularity on energy efficiency of software execution S. Bhattacharya et al. [103] and S. Bhattacharya et al. [102] identified that the main methods allowing the modularity of software under development do not allow a perfect modularity and inadvertently introduce overloads at runtime. Authors take as an example the case of creating multiple copies of a processed document when using a document exchange gateway, generating higher resource consumption than would normally be expected.

High level design decisions on energy consumption of software C. Sahin et al. [78] identified that high-level design decisions to use design patterns or not have an effect on energy consumption of software. K. S. Vallerio et al. [75] identified that different GUI features have different effect on energy consumption of software. S. Pasricha et al. [81] identified that power management algorithm used on mobile phones caused a higher energy consumption compared to a power management algorithm developed by the authors. Finally, F. Jiang, et al. [82] identified that the type of input modality on mobile phone application used by users had an effect on energy consumption.

Users consumption habits and service providers optimization techniques of IT service on the environmental impact of the supplier data center C. Atkinson and T. Schulze [83] identified that the environmental impact of data centers, providing software as services, is influenced by usage patterns, and optimization and balancing techniques employed by IT service providers.

5.3.2.2 Calculation models

No calculation model were found.

5.3.2.3 Solutions

A Software modularity on energy efficiency of software execution

The solutions identified in research works dealing with this impact are the following:

1. Reach the level of zero bloats

2. Analyze the link between software bloat reduction and energy consumption decrease

A.1 Reach the level of zero bloats

Concerning the implementation of solutions to limit negative environmental impacts, S. Bhattacharya et al. [103] present the problem of bloat softwares and the challenges surrounding the resolution of these. According to the authors, software runtime bloats are software programs which cause "runtime resource consumption disproportionate to the actual function being delivered".

The causes of these runtime bloats are multiple. One reason is the 30-year growth in the amount of resources available to run the software, reducing the need to control resource consumption. A second reason is the industry's need for rapid software development and, as a result, the use of a development support framework with flexibility and flexibility and interoperability increases the complexity of software. A third reason is the need, today, for some companies, to develop generic software whose exploitation uses only part of the functionalities at the same time.

The authors believe that it is possible to limit the size of software bloats and according to them, it is possible to qualify a software as green if it reaches the level of zero bloats. However, the authors believe that it is theoretically not always possible to reach this level because general features will sometimes be necessary. Another challenge is that it is very difficult to model the necessary functionalities and the extra features for a given software. To counter this problem, authors recommend a bottom-up approach, looking for the source of the bloats (most existing work giving priority to the bottom-up approach). Once the types of bloats are detected, another challenge is to evaluate the amount of resources they consume. The authors state that on this point, heuristics are necessary. The potential gains in energy consumption are potentially large according to the authors (their experiences have allowed them to halve software consumption). Moreover, if we consider that the increase of the resources required for software operation also affects the lifetime and the production of the hardware supporting the software, we can estimate that the interest for the decrease of the bloats is very important for software eco-design.

A.2 Analyze the link between software bloat reduction and energy consumption decrease

S. Bhattacharya et al. [102] have studied the relationship between decreased runtime bloat and decreased energy consumption. The authors were able to observe several things by taking into account the proportionality of the energy consumption according to the use of the CPU as well as the CPU part (cores, cache or level of hyper-threading) acting as bottleneck for the performance. They were able to observe that the effect on energy consumption depends on the CPU part on which the bloat is acting, by measuring the general performance and the performance at peak. Their observations are, firstly that if a particular CPU part is the bottleneck, the peak power increases proportionately to the bottleneck CPU part. Then, if the bottleneck is resolved, energy gains are only visible if the other resources of the machine used by the software do not have an energy consumption proportional to their use, otherwise their energy consumption will rise.

These results show that the correlation between runtime bloat decrease and energy gain is not as strong as it seems.

Unfortunately, neither we nor the authors of the previous article have found other research presenting empirical results in this area, in order to validate the intuitions and justify the investment of the software development organizations.

B High level design decisions on energy consumption of software

The solutions identified in research works dealing with this impact are the following:

1. Choose design pattern that can limit energy consumption
2. Choose user interface design that can limit energy consumption
3. Develop software aware of interaction's pattern of users
4. Choose interaction modes that can limit energy consumption

B.1 Choose design pattern that can limit energy consumption

C. Sahin et al. [78] tried to establish the link between energy impact and design patterns used to resolve commonly encountered problems. To establish this link, they compared the energy consumption of an implementation without a design pattern and an implementation based on a design pattern. The results observed were that depending on the patterns, there was either no significant change in consumption (for example for the bridge pattern), a significant change, positive or negative, or a very significant change (for example for the decorator pattern, whose use caused an increase in consumption by 700%). These results could not be explained by the authors solely on the basis of their measurement and they called for the development of more empirical research and resource analysis impacted by design patterns, in order to provide a solid background to explain future results.

B.2 Choose user interface design that can limit energy consumption

K. S. Vallerio et al. [75] have shown that user interface design choices can influence the energy consumption of the system. However, this study was conducted before the explosion of the use of smart-phones and good practices presented by the authors are now probably used in the sector.

The authors also conducted a preliminary study on user preferences. The results were that most of the users have preferred interfaces that allow them to perform their task quickly than good-looking interfaces.

B.3 Develop software aware of interaction's pattern of users

S. Pasricha et al. [81] shows that it is potentially possible to achieve significant environmental gains by developing software aware of interaction's pattern of users. The authors have developed a framework to adapt the CPU frequency and the brightness of the screen of a smart-phone according to the type of interaction pattern (vocal command, direct interaction, use of the keyboard, etc) performed by the user. The authors of this article have achieved several results.

First, on the assumption that quality of service is different for different users, they used a Bayesian method to classify the applications according to the different interaction patterns specific to each user.

Then they used energy management algorithms to modulate the energy consumption of the smart-phone according to the interaction pattern realized by the user.

Finally, they confronted their results with a real use and obtained encouraging results: 29% reduction of the consumption compared to the use of a traditional operating system resource manager, without effect on the quality of service for each user.

B.4 Choose interaction modes that can limit energy consumption

In terms of user interface design, F. Jiang, et al. [82] think that the interaction mode chosen for communication between the user and the application can significantly influence the energy consumption and should therefore be chosen knowingly. The authors evaluated differences in energy consumption according to text entry modes on mobile applications. They compared three interaction modes and their results show energy consumption differences of around 50% depending on the interaction mode and the length of the text. These results are encouraging because they show that counterintuitive results (Speech-To-Text appears to be more energy efficient when entering long texts) can appear when real measurements are made.

C Users consumption habits and service providers optimization techniques of IT service on the environmental impact of the supplier data center

C.1 Integrate environmental constraints into software design

The following two solutions shows what could be done to take into account environmental impacts in modeling processes. C. Atkinson and T. Schulze [83] have attempted to show in their article a possible application of two simple techniques for integrating environmental constraints into software design. In the example used by the authors, whose subject is the modeling of a shopping

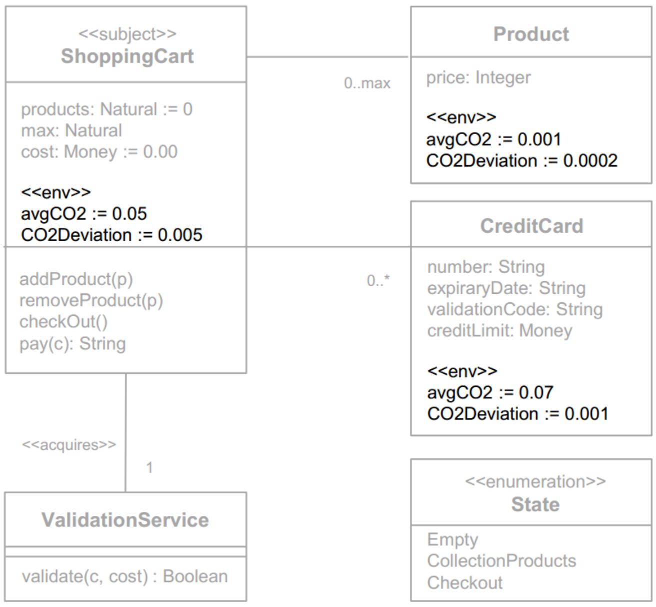
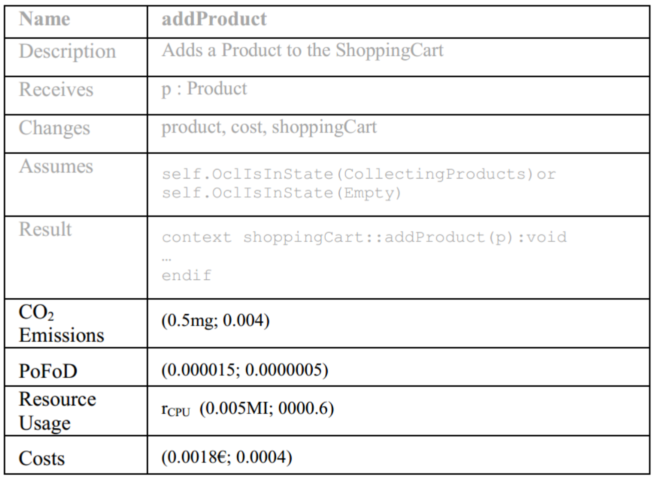


Figure 5.2: Structural view - extract from C. Atkinson and T. Schulze [83]

cart, the specifications are formulated using a method developed by these same authors: KobrA. KobrA is based on UML and the OCL constraint language to define, in an understandable way for the stakeholders, a specification model. This method is based on three views to model the specifications: structural, operational and behavioral.

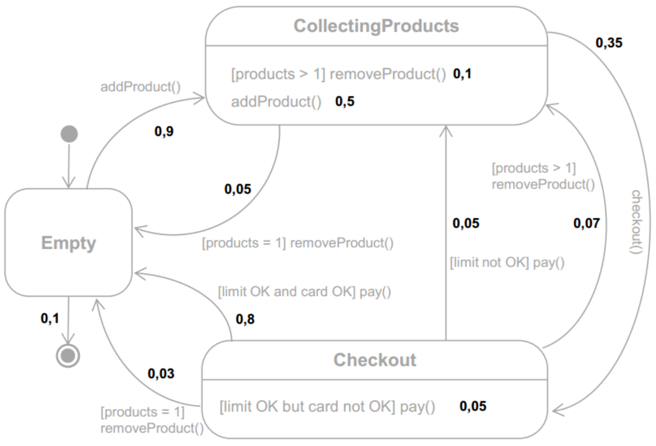
The first solution involves linking an environmental impact assessment with elements of the specifications themselves in order to estimate the environmental impact of the software or part of it. Specifically, in the structural view (Figure 5.2), attributes corresponding to the CO2 emission for creating an instance for each class was added. An important limitation is that the authors do not specify precisely how it would be possible to calculate these CO2 emissions and just say it should be possible for an expert to estimate these emissions on the basis of his experience. In the operational view (figure 5.3), an evaluation of the environmental impacts, carried out by an expert, of the call of a method is added for each method. Finally, in the behavioral view (Figure 5.4), usage probabilities are added to characterize the behavior of users. The information added on these three views is then combined to calculate the probabilities of the environmental impact values. Thus, from this model, specifications can be modified and their impact on the software's energy consumption can be observed.

The authors also produced a variation of the first solution, in the form of SLAs. This variation consists in considering the figures provided by experts (in consultation with the customer and the service provider) no longer as estimates of actual software emissions, but as emission constraints that cannot be exceeded. An important consequence is that it is necessary to estimate emissions in order to verify whether they correspond to the constraints set.



Name	addProduct
Description	Adds a Product to the ShoppingCart
Receives	p : Product
Changes	product, cost, shoppingCart
Assumes	`self.OclIsInState(CollectingProducts)or self.OclIsInState(Empty)`
Result	`context shoppingCart::addProduct(p):void … endif`
CO_2 Emissions	(0.5mg; 0.004)
PoFoD	(0.000015; 0.0000005)
Resource Usage	r_{CPU} (0.005MI; 0000.6)
Costs	(0.0018€; 0.0004)

Figure 5.3: Operational view - extract from C. Atkinson and T. Schulze [83]



CollectingProducts
[products > 1] removeProduct() 0,1
addProduct() 0,5
0,35
addProduct()
0,9
0,05
Empty
[products = 1] removeProduct()
[products > 1] removeProduct()
0,05
0,07
[limit not OK] pay()
checkout()
[limit OK and card OK] pay()
0,8
0,1
0,03
Checkout
[limit OK but card not OK] pay() 0,05
[products = 1] removeProduct()

Figure 5.4: Behavioural view - extract from C. Atkinson and T. Schulze [83]

Impacts	Calculation models	Solutions identified
Software modularity on energy efficiency of software execution		S. Bhattacharya et al. [103] (Commentary) S. Bhattacharya et al. [102] (Problem identification and solution)
High level design decisions on energy consumption of software		K. S. Vallerio et al. [75] (Problem identification and solution) C. Sahin et al. [78] (Problem identification and solution) S. Pasricha et al. [81] (Evaluation research) F. Jiang, et al. [82] (Problem identification and solution)
Users consumption habits and service providers optimization techniques of IT service on the environmental impact of the supplier data center		C. Atkinson and T. Schulze [83] (Problem identification and solution)

Table 5.2: Table of research works by impacts identified and by contribution

5.3.2.4 Conclusion

Structure coverage

Based on the table 5.2 our observations on the state of the literature are as follows:

First, we can classify the environmental impacts studied in the literature for the design phase into a general impact category, itself divided into three categories:

- Impact of design phase on software operating phase
  - Software modularity on energy efficiency of software execution
  - High level design decisions on energy consumption of software
  - Users consumption habits and service providers optimization techniques of IT service on the environmental impact of the supplier data center

Secondly, we can observe the same things as in the previous phase. No article offers both an estimation model and one or more solutions for a given environmental impact. Within the framework of our research, we have not collected any research works allowing, for a given environmental impact, to have an estimate and a solution allowing the reduction of this impact.

Finally, concerning the types of research works, we can observe that the most represented type of article is the problem identification and solutions. We also found an evaluation research, which brings the number of research-based research works to 7 out of 11 for the design phase. Finally, as for the previous phase, we did not find any validation research.

Impacts

Concerning the impacts studied, we did not find any mention of impact such as the influence of design on life expectancy of the software or on the maintenance phase environmental impact.

Calculation models

Concerning impact calculation models, as in the previous phase, no line of code is created yet at this point. However, the idea of a bottom-up mapping as defended by C. Sahin et al. [26] is still relevant in this phase and should make it possible to estimate the environmental impact at this level. On the same subject, Eva Kern et al. [53] estimate, citing the research works: A. Hindle [84], Dick et al. [87] and Johann et al. [89], that we are approaching the point where designers will be able to make design decisions based on the energy impact of these choices. It is also interesting to note that general environmental impact estimation models (like J. Taina [52] and Eva Kern et al. [53]), taking into account several phases of the software life cycle, do not take into account the design phase, its impacts being mixed with those of the requirements engineering and development phases.

Solutions

Regarding solutions, apart from C. Sahin et al. [78] proposing to choose design that can limit energy consumption, we have not found other research works proposing solutions using reuse with the aim at reducing the environmental impact. The main consequence of reuse could be to reduce the working time in this phase. With the assumption that a decrease in working time decreases the energy required to achieve the design phase, this could therefore reduces negative environmental impacts. The reuse can be for example the use of: architectural patterns (to reuse knowledge), application frameworks (for reuse of application subsystems) and software product lines (for reuse of architectures).

Finally, the impact of user interaction design has only been studied at the energy level and mainly at the mobile device level, according to K. S. Vallerio et al. [75]. We think that the use of fast interfaces (and therefore simple, light and easy to handle) could be more energy efficient because they would allow to perform tasks quickly and thus reduce the time of application activity, as proposed by K. S. Vallerio et al. [75].

5.3.3 Development

5.3.3.1 Impacts

The authors who dealt with this third phase identified several types of impact.

Development process execution on its own carbon footprint Eva Kern et al. [53] identified that working methods during the development process, such as meetings, video conferences, car travel, ... are responsible for CO2 emissions.

Development activities on their own sustainability S. Mahmoud et al. [70] have identified metrics that can be used to measure the environmental impact of software development.

Development activities on software energy consumption E. Capra et al. [17] identified that the increased usage of frameworks and external libraries have an impact on the energy consumption of the software developed. Dick et al. [87] identified that depending on the scenarios of use, a web browser of one firm will consume more energy than another.

Use of usual metrics on software energy consumption R. Verdecchia et al. [107] identified that standard code smells would improve energy efficiency at runtime. A. A. Bangash et al. [111] identified that some structural metrics correlated with software energy consumption.

API parameterization and choice on software energy consumption J. Singh et al. [92] and I. Manotas et al. [108] identified that for a given code, different APIs have different effects on the energy consumption of the software at runtime.

Source code component on software energy consumption R. Pereira et al. [113], D. Li et al. [114], H. S. Zhu et al. [112], X. Li et al. [88] and A. Banerjee et al. [109] identified that some code components were responsible for higher energy consumption than other code components. For example, A. Banerjee et al. [109] have shown that misuse of Android API system calls has a significant impact on energy consumption. The authors have shown that the resources used by applications during their execution are sometimes not released by them and lead to unnecessary energy consumption, even after the application in question has been closed.

Running time and amount of data moved on software energy consumption H. Acar et al. [27], J. Pallister [91], I. Stirb [106] and YongKang Zhu et al. [105] and K. Eder et al. [76] identified that a decrease in execution time is related to a decrease in software energy consumption. In addition, K. Eder et al. [76] and F. A. Ali et al. [85] have identified that the amount of data exchanged (between components of the same system for K. Eder et al. [76] or through the network to F. A. Ali et al. [85]) have an impact on the energy consumption associated with the execution of the software. Finally, R. Pereira et al. [110] identified links between the speed and memory usage of several languages and energy consumption of the software developed in these languages.

Code optimization on software energy consumption S. Murugesan et al. [93] identified, among other things, that the performance of the algorithms used has an impact on the software's energy consumption. M. Valluri and L. K. John [90] identified that compiler optimization type, focused on reducing the number of instruction, make it possible to reduce software energy consumption. S. Bhattacharya et al. [104] identified that the number of objects created by a Java program is related to software energy consumption.

5.3.3.2 Calculation models

A Development process execution on its own carbon footprint

A.1 Calculation of carbon footprint of the development phase

In their article, Eva Kern et al. [53] present the results of the application of a method for calculating the CO2 emissions of the development phase and apply this method to a real project. In order to evaluate CO2 emissions, they focus on the company's infrastructure and company's IT infrastructure CO2 emissions per year and multiply it by the number of years of software development. They also focused on CO2 emissions related to employee commuting during the development phase. For this, they use the average number of working days per year per person. Their results show that in the case of software distributed in small quantities, the development phase is the phase with the greatest environmental impact (it is important to note that of all the development phases, the authors only consider the development and maintenance phases in their research).

B Development activities on software energy consumption

The calculations models identified in research works dealing with this impact are intended to calculate:

1. Software's functionalities energy consumption
2. Softwares energy consumption on desktop and server based on theoretical user behavior

B.1 Software's functionalities energy consumption

E. Capra et al. [17] have developed a hardware measure based on a dynamic analysis of the code. The authors developed a method to compare the energy efficiency of software using hardware measurement and benchmarks they developed. The method used to measure energy efficiency uses hardware measurement tools whose data are recorded on a different machine than the one on which the tested software is running. The energy efficiency is calculated from the energy required by a software to perform a set of tasks, compared to the energy required by other software to perform the same set of tasks. The benchmark is created from an automation of the realization of one or more sets of tasks. The authors' result is that their calculation method effectively compares the energy consumption of several softwares. However the authors could not find other methods to calculate the energy efficiency of software and suggest that research should be conducted to refine their calculation method and develop new ones, like the energy efficiency of hardware where several types of metrics have been well defined and standardized.

B.2 Softwares energy consumption on desktop and server based on theoretical user behavior

Dick et al. [87] have defined the characteristics of a scenario based estimation of energy consumption on the basis of dynamic analysis. The authors found that other forms of measurement are possible depending on the type of software developed. When the software works through user interaction, tools that take into account usage scenarios are more appropriate. They considered that this kind of tool should be able to be used by developers, users and system administrators. This consideration led them to not consider the execution of part of the source code to perform the measurements, but rather task completion scenarios. The main difference with tools that focus on source code measurement is that task-based tools can be used to compare software for the same tasks that users and administrators would look for.

C Source code component on software energy consumption

C.1 Calculate softwares energy consumption on Android based on theoretical user behavior

X. Li et al. [88] have applied a model of energy consumption evaluation from a dynamic analysis of the source code of an Android application. The evaluation was conducted using scenarios to simulate a typical use of the evaluated mobile application. This allowed the developers to manually refactor the code on the most resource-intensive elements and thus gain between 6% and 50% of energy consumption according to the scenarios.

D Running time and amount of data moved on software energy consumption

The calculation models identified in the research works dealing with this impact are intended to calculate:

1. Java applications energy consumption at the source code line level
2. Mobile application's energy consumption at the method calls level

D.1 Java applications energy consumption at the source code line level

H. Acar et al. [27] have developed a software measure based on a dynamic analysis of the code. The authors studied the impact of the source code on the energy consumption of software execution. For this, they have developed a software method to measure the energy consumption of code written in Java. The advantage of their approach, compared to other software based measures, is that it takes into account several resources (CPU, RAM and hard disk) instead of being limited to one, like other existing tools (PowerAPI, SPAN, Joulemeter). This allows for more accurate measurements. Another advantage, compared to material measurement methods, is that it is not necessary to invest in equipment offering precise measurements and there is the possibility of targeting a specific process during measurements.

D.2 Mobile application's energy consumption at the method calls level

F. A. Ali et al. [85] have developed a model to estimate the energy consumption of mobile applications. The resources that the authors have taken into account in their model are the CPU, RAM and Wi-Fi, other resources being excluded from the model. These resources have been chosen with a view to dynamically offload certain parts of the application if they consume too many resources when running on mobile. The authors' results are that the consumption estimate is correct with an error of less than 10%.

5.3.3.3 Solutions

A Development activities on their own sustainability

A.1 Adapt reused component to only meet the requirements of the application under development

S. Mahmoud et al. [70] have developed a software development model covering all phases of the software life cycle and have integrated the customization of external or internal software components, used to build the developed solution, in the implementation phase. Indeed, the authors think that since component reuse causes an unnecessary increase in the amount of code and functionality, they propose to modify components so that they only meet the requirements of the application under development. However, the authors do not explain in concrete terms what this modification would consist of. Nor do they prove that the energy required to perform library or other software changes, considering that this is possible in all cases, does not exceed the energy of development from scratch. Still without demonstrating the concrete effects of the solutions they put forward, the authors believe that the choice of efficient algorithms and data structures enables software sustainability to be achieved and cite several general good practices in this area.

B Use of usual metrics on software energy consumption

B.1 Use of standard code smells and software metrics

R. Verdecchia et al. [107] studied the possible use of standard code smells to reduce energy consumption and the use of standard software metrics to evaluate the energy efficiency of an application. The code smells used were selected because they were detectable and automatically correctable by the detection and refactoring tool the authors wanted to use. The authors' results were that the code smells tested reduce the energy consumption of the applications to which they have been applied. However, the authors were unable to observe a relationship between the software metrics they studied and the energy consumption of the applications tested. However, not all software metrics are uncorrelated with any application's energy consumption. A. A. Bangash et al. [111] have identified two sets of software metrics with a satisfactory correlation to Android application energy consumption.

C API parameterization and choice on software energy consumption

Solutions identified in research works dealing with this impact are the following:

1. Choose API with least software energy consumption
2. Automatically generate the most energy efficient version of a Java application

C.1 Choose API with least software energy consumption

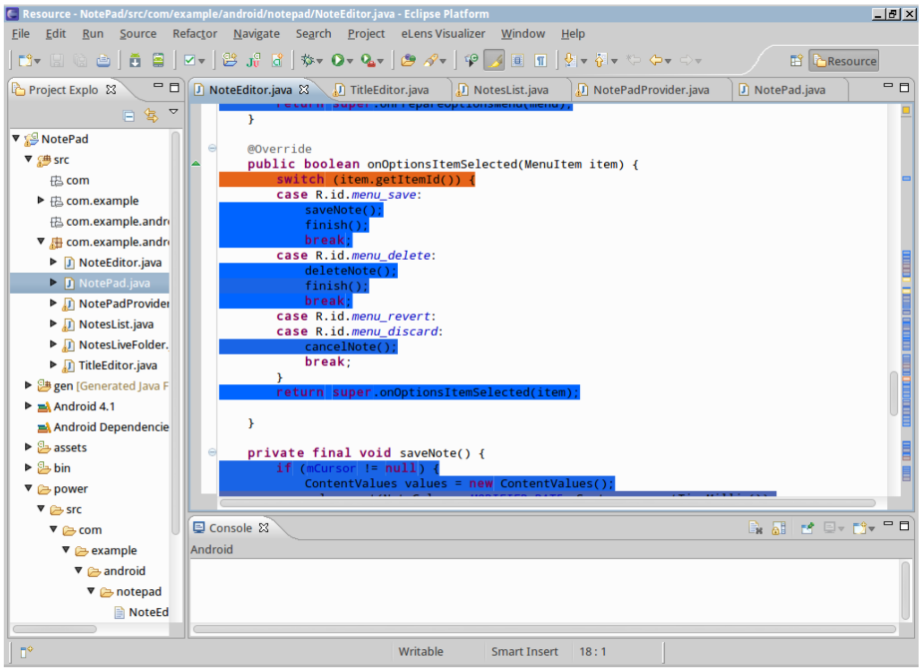
The parameterization of the APIs can also be a possible solution to limit the energy consumption. J. Singh et al. [92] conducted a study on the impact on software power consumption of API choices allowing the implementation of basic file processing in Java (reading, copying, compression, decompression). The goal of the authors was to use APIs with at least one modifiable parameter (the buffer size) to compare the energy efficiency difference between the parameterizable API and the others. Their results have been that adjusting the buffer size has a significant positive effect on energy efficiency. This work opens the way for more research to gather an information base allowing developers to make an informed choice about the APIs they use to gain energy efficiency.

C.2 Automatically generate the most energy efficient version of a Java application

I. Manotas et al. [108] developed a framework model called SEEDS, aimed at enabling developers to develop more energy efficient applications by making only high level decisions. Indeed, to use the framework, the developer would only have to provide the code of its application, the tests of its application and the changes it authorizes the framework to make, and the framework would automatically generate the most energy efficient version. The authors validated their model by creating an instantiation of it in the form of an API selector, automatically choosing the least energy-consuming libraries implementation to use in a given application. The core of the model's operation is the automatic generation of several versions of the application and the evaluation of the energy consumption of each, in order to select the best one. The main disadvantage of this model is that generating and evaluating different versions can take several dozen hours or more.

D Source code component on software energy consumption

Solutions identified in research works dealing with this impact are the following:



```java
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.menu_save:
            saveNote();
            finish();
            break;
        case R.id.menu_delete:
            deleteNote();
            finish();
            break;
        case R.id.menu_revert:
        case R.id.menu_discard:
            cancelNote();
            break;
        }
        return super.onOptionsItemSelected(item);

    }

    private final void saveNote() {
        if (mCursor != null) {
            ContentValues values = new ContentValues();
```

Figure 5.5: Illustration of the visualization of energy consumption by code line in the vLens tool - extract from D. Li et al. [114]

1. Display the energy consumption of each line of code relative to the other software lines
2. Automatically detect, target and correct energy problems in mobile application
3. Target source code elements that cause abnormal energy consumption
4. Take into account sustainability criteria in context aware software

D.1 Display the energy consumption of each line of code relative to the other software lines

D. Li et al. [114] have developed a prototype, called vLens, to link to each code line, an indication of its energy consumption relative to the other code lines of the program (the figure 5.5 shows a view). To achieve this relationship between energy consumption and code line, the authors used a hardware measurement system and a static analysis of the program code. The results of their prototype show that it is extremely accurate in identifying the most resource-intensive lines of code within a reasonable time. However, the result of their solution is dependent on the quality of the use cases that are run to obtain a profile of the application's energy consumption.

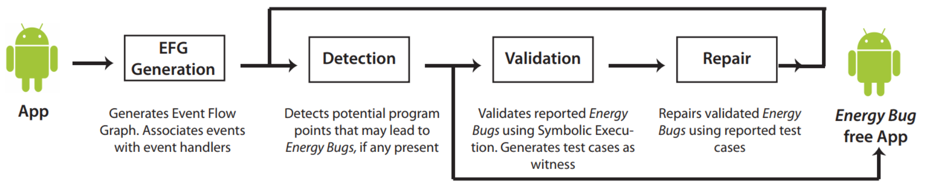

App
EFG Generation
Generates Event Flow Graph. Associates events with event handlers
Detection
Detects potential program points that may lead to Energy Bugs, if any present
Validation
Validates reported Energy Bugs using Symbolic Execution. Generates test cases as witness
Repair
Repairs validated Energy Bugs using reported test cases
Energy Bug free App

Figure 5.6: System Overview - extract from A. Banerjee et al. [109]

D.2 Automatically detect, target and correct energy problems in mobile application

A. Banerjee et al. [109] have developed an hybrid solution, mixing static and dynamic analysis to automatically detect, target and correct energy problems in mobile applications related to poor resource management. Unlike the article written by D. Li et al. [114], the solution proposed by the authors is not dependent on the tests provided because these are automatically generated by the tool developed. The authors implemented their solution as an Ecplise plug-in, called EnergyPatch. The figure 5.6 represents an overview of the system, where static analysis can detect possible energy problems and where dynamic analysis, supported by a hardware measurement system, can validate these detections. The result of the study carried out by the authors is that their solution allows to generate tests targeting energy problems and allows to correct these problems with a significant energy gain.

D.3 Target source code elements that cause abnormal energy consumption

R. Pereira et al. [113], have defined the basis for a technique that can target elements of the source code that cause abnormal energy consumption. As with the article of A. Banerjee et al. [109], the technique is composed of a dynamic and static analysis of the code, with the difference that it can theoretically be used independently of the development target (mobile, embedded, data center, etc.).

D.4 Take into account sustainability criteria in context aware software

In the area of context aware software implementation, allowing the software to react to the environment, H. S. Zhu et al. [112] have developed a programming language based on a Java extension, called ECO. This language aims to take into account sustainability criteria such as energy consumption or heat dissipation. This extension is used by packaging the most resource-consuming parts of the developed program code into blocks. In each of these blocks, the programmer must then define a method to calculate the progress in executing the code. Then, the programmer must give the normal value of the sustainability criterion taken into account (the percentage of battery that the code can consume for example). Finally, the programmer must define the parameters of his code that may vary during execution (for example, the resolution of an image generated by the program). When executing the code, ECO checks that the code follows a progress that fits into the sustainability constraint. If this is not the case, ECO automatically adapts the code based on the parameters defined by the developer.

E Running time and amount of data moved on software energy consumption

E.1 Configure compilers to optimize the running time and amount of data moved

In his thesis, J. Pallister [91] studied the differences between optimization for energy and optimization for execution time. The author defined energy consumption as the average power required by the program multiplied by the execution time. Thus, optimization for execution time allows a reduction in energy consumption if optimization does not increase the average power required by the program. The author's results are that the optimization parameters for time almost always result in an optimization of energy consumption. However, no optimization parameter for time is effective independently of the targeted platform.

YongKang Zhu et al. [105] and more recently I. Stirb [106] conducted research on the energy impact of loop fusion optimization, limiting the number of instructions and optimizing data placement. The authors found that this type of optimization significantly reduces the energy impact of the compiled program by almost systematically reducing the execution time.

F Code optimization on software energy consumption

Solutions identified in research works dealing with this impact are the following:

1. Use common sense methods to optimize the code
2. Configure compilers to optimize the code
3. Optimize number of objects created by a Java at running time

F.1 Use common sense methods to optimize the code

S. Murugesan et al. [93] have written an important book in the field of Green IT, quoted 1010 times when this thesis has been write[2], dealing with both the hardware part and the software part. In the chapter dealing with Green Softwares, the authors went through methods to limit the energy consumption induced by the software. The methods treated are illustrated by examples of applications and associated energy savings. However, the authors only cite possible solutions without implementing them.

The first method is the implementation of efficient algorithms, which reduces the CPU usage time. The main assumption is that energy gain in execution time would not be upset by an increase in energy consumption due to more intensive CPU utilization. The authors nevertheless consider that it is probably more interesting to limit the consumption of the CPU while it is in idle than while it is in activity. The challenges we can observe are that having several possibilities

[2]August 2018

for implementing algorithms and having the possibility of comparing them is not always possible.

The second method is the use of multi threading, which reduces the CPU usage time. The effect sought is that since the power consumption is not directly proportional to the number of threads running in parallel, running the software on multiple threads in parallel reduces the consumption of the software execution. An important challenge is that the software must be divisible and executable in parallel, which is not always the case.

The third method is the improvement of the energy consumption during idle phases, called idle efficiency. Idle efficiency is an important axis of energy gain because a software will spend the vast majority of its execution time in this state. The axes allowing a reduction of the use of the resources while the software is in idle state are for example to avoid the active waitings, not to leave a timer with an unnecessarily small resolution or to limit the background activity.

F.2 Configure compilers to optimize the code

M. Valluri and L. K. John [90] conducted a quantitative study in order to compare the effect of four optimizations performed by a compiler on the energy consumption and power dissipation of the generated software. Their results were that lower energy consumption is directly related to the optimization of the number of instruction, with methods of loop unrolling or inlining of function calls for example.

F.3 Optimize number of objects created by a Java at running time

S. Bhattacharya et al. [104] carried out an experimental study studying the factors influencing the performance and energy consumption of bloat software. The hardware resource considered was the CPU (with among other factors, the number of cores, the hyper-threading level and the size of the caches). The energy consumption measurements were made using a benchmark measuring the consumption of the entire machine, without distinguishing between its resources. The result of their study was that optimizing the number of objects created by a Java program had an impact on both performance and consumption. However, the magnitude of the impact depended on hardware and software factors, such as the configuration of the JVM or the use of power management features.

5.3.3.4 Conclusion

Structure coverage

Based on the table 5.3 our observations on the state of the literature are as follows:

First, we can classify the environmental impacts studied in the literature for the development phase into eight impacts categories:

- Development process execution on its own carbon footprint

Impacts	Calculation models	Solutions identified
Development process execution on its own carbon footprint	Eva Kern et al. [53] (Commentary)	
Development activities on their own sustainability		S. Mahmoud et al. [70] (Problem identification and solution)
Development activities on software energy consumption	E. Capra et al. [17] (Problem identification and solution) Dick et al. [87] (Problem identification and solution)	
Use of usual metrics on software energy consumption		R. Verdecchia et al. [107] (Validation Research) A. A. Bangash et al. [111] (Validation Research)
API parameterization and choice on software energy consumption		J. Singh et al. [92] (Problem identification and solution) I. Manotas et al. [108] (Validation Research)
Source code component on software energy consumption	X. Li et al. [88] (Problem identification and solution)	D. Li et al. [114] (Validation research) A. Banerjee et al. [109] (Problem identification and solution) R. Pereira et al. [113] (Technical note) H. S. Zhu et al. [112] (Validation research)
Running time and amount of data moved	H. Acar et al. [27] (Validation research) F. A. Ali et al. [85] (Validation research)	J. Pallister [91] (Problem identification and solution) I. Stirb [106] (Problem identification and solution) YongKang Zhu et al. [105] (Evaluation research)

Code optimization on software energy consumption		S. Murugesan et al. [93] (Commentary) M. Valluri and L. K. John [90] (Validation Research) S. Bhattacharya et al. [104] (Case Study)

Table 5.3: Table of research works by impacts identified and by contribution

- Development activities on their own sustainability
- Development activities on software energy consumption
- Use of usual metrics on software energy consumption
- API parameterization and choice on software energy consumption
- Source code component on software energy consumption
- Running time and amount of data moved on software energy consumption
- Code optimization on software energy consumption

Secondly, we can observe that contrary to the previous phases, we were able to find studies allowing, for the same impact, to have calculation methods and solutions in order to decrease this impact. However, we can observe that as in previous phases, no article offers both an estimation model and one or more solutions for a given environmental impact.

Finally, concerning the types of research works, we can observe that, except for the impact of the development process execution on its own carbon footprint, there are research-based research works in all the identified impacts.

Impacts

Concerning the impacts studied, an important difference with previous phases is that since the software can be run and tested here, its impact induced by its use of hardware resources can therefore be estimated. Thus, the impact of the development process itself (including software testing) and the impact on the software operation phase could be addressed in the literature. However, we only found studies dealing with this second type of impact.

Some authors have identified impacts without developing an estimation or solution model.

E. Capra et al. [17] identified that the development and maintenance phases have an impact on energy consumption. In fact, since the purpose of these frameworks and libraries is generally to be used in as many cases as possible, their implementations would tend not to be energy efficient. An important observation made by the authors is that performance is not synonymous with energy efficiency, which prevents the systematic use of existing research in the field of optimization.

The authors studied the influence of development environments on the energy efficiency of the software developed. To measure this influence, they have developed a new metric: "framework entropy" which measures the number of external libraries used by the developers as well as the complexity of the constructions, to provide an indication of the influence of the development environment on the source code.

The results of their study is that small and medium sized applications have an energy efficiency that increases with the "framework entropy", whereas this effect is reversed for large applications. An explanation given by the authors is that small applications are potentially developed from scratch and developers are not able, in a limited time, to develop more efficient implementations than those developed in libraries. According to the authors, large applications would suffer from the complexity of the interactions between the different layers of libraries. It is important to note that the sizes of the applications mentioned by the authors are very small compared to the applications distributed by the industry, which means that the vast majority of them could have an energy efficiency which decreases with the increase of the "framework entropy".

R. Pereira et al. [110] have identified that programming languages have different impacts on the energy consumption of software developed with these languages. The authors compared the energy efficiency of 27 programming languages as well as paradigms and execution types associated with these languages. They have for example shown that programs written in C language are more energy efficient than program written in Python language, Perl or PHP. The results of their research also showed that the languages generating the fastest executions were not necessarily the most energy efficient.

Calculation models

Concerning calculation models, we were able to find several researches dealing with the measurement of the software's energy consumption. Thus, according to E. Kern et al. [53], several methods have already been developed to calculate the energy impact of the source code. The domain in which we found the most research is the mobile software developments. F. A. Ali et al. [85] have for example found in 2016, seven models of consumption of mobile devices. Nevertheless, according to G. Procaccianti [86], it is possible to find calculation models of energy consumption in other domains like embedded system, servers and desktop PC.

Johann et al. [89] believe that the energy consumption measure should be automated to increase the use of calculation models. They consider that it is necessary to integrate these models in the current development environments to execute them by builds tools. This integration would also make it possible to collect the developers' measurement data and derive an average consumption of the programs, which could be refined according to their type and thus be able to quickly and automatically identify energy consumption problems. According to the authors, the sharing of consumption information would also make it possible to develop more efficient models of energy consumption.

Nevertheless, the refinement of software dynamic analysis and the increase of their use could increase and improve knowledge of the consumption-code relationship. Thus, future static analysis tools could use this knowledge to perfect themselves. In addition, this knowledge could later allow to consolidate the knowledge of the consumption-design relationship in order to allow designers to develop more energy efficient solutions, as defended by C. Sahin et al. [26].

In order to facilitate future calculation model creation, K. Eder et al. [76], introduced four characteristics that should be present in the calculation models for estimating energy efficiency of software under development:

1. The calculation models should aim at an "initial energy analysis", carried out in order to obtain an "energy profile" of the software. The energy profile gives the distribution of the energy consumption within a given software, which makes it possible to identify the elements to be modified if the developers wish to reduce the energy consumption of the software.

2. The model must make it possible to make choices at high levels of abstraction, until the selection of the deployment platform. This is justified by to their findings that the lower the level of abstraction, the easier it is to produce accurate measurements and the more the energy consumption is measured on a high level of abstraction, the more this information is useful for software developers. As a result, the model should be able to evaluate energy consumption, regardless of the implementation of the software and the type of hardware, to allow comparisons to be made.

3. For a given platform, the model should allow comparisons across multiple configurations.

4. The model should also consider the data structures and algorithms selected.

Solutions

Regarding solutions, the ones we have found are all aimed at helping developers to produce software whose operation phase will be as environmentally friendly as possible. Several solutions proposed in the research works would allow developers to obtain information regarding the energy consumption of their code, in order to identify elements with abnormal or excessive consumption. This type of solution are the most advanced at present. Another solution would be to use languages that aims to take into account sustainability criteria. This type of solution has so far only been studied for software aware of their environment. Another solution, offering more automation, would be to use frameworks that would automatically generate a resource-efficient version of the developed software. For now, this type of solution works by generating multiple versions of software and choosing the most energy efficient one. In addition, it asks the developers to provide the tests and changes that the version generator can make themselves. Further research could be conducted to simplify the use of these generators, improve their operation or allow them to optimize parameters other than energy consumption. Another possible solution would be to use existing code smells and metrics to evaluate the energy efficiency of software under development. However, research focused on this type of solution has not yet yielded any results that would allow this solution to be considered viable. Finally, another possible solutions, mentioned in the research works, would be to allow developers to parameterize the APIs, libraries or compilers they use in order to limit their impact on the resource consumption of the code. However, these researches are only at the exploration stage.

5.3.4 Maintenance

5.3.4.1 Impacts

The authors who dealt with this fourth phase identified two types of impact.

Maintenance process execution on its own carbon footprint J. Taina [52] and Eva Kern et al. [53] identified that the carbon footprint of the maintenance phase relate to the electrical consumption and heating needs of the buildings, the infrastructure needed for process, the working methods as well as the commuting to work of the stakeholders.

Changes made to software on software energy consumption A. Hindle [84] and N. Amsel et al. [94] identified that successive builds and versions of software have a tendency to see their energy consumption increase. For example, A. Hindle [84] showed, by comparing the energy consumption of 500 Firefox builds, that energy consumption tended to increase as software changes. G. Pinto et al. [96] assumed in their article that refactoring activity has an impact on energy consumption and that it was necessary to address it.

5.3.4.2 Calculation models

A Maintenance process execution on its own carbon footprint

A.1 Calculation of carbon footprint of the maintenance phase

J. Taina [52] has developed a model of carbon footprint calculations of the development process that takes into account the maintenance phase and this phase accounted for 83% of the total impacts of the development process. However, we lack information to draw conclusions about the importance of this phase for all types of software.

Eva Kern et al. [53] have applied the same model that they used for the development phase to estimate the carbon footprint of the maintenance phase (see section 5.3.3.2).

5.3.4.3 Solutions

A Changes made to software on software energy consumption

Solutions identified in research works dealing with this impact are the following:

1. Limit software degradation
2. Inform users about the evolution of the environmental impact as the software is updated

3. Modify the software to reduce its energy impact

A.1 Limit software degradation

A. Hindle [84] studied the evolution of energy consumption through the different versions of Firefox and showed that the consumption increased according to the successive versions. However, an interesting point is that he discovered that the metric of the number of lines of code does not correlate with the evolution of energy consumption, contrary to a widespread idea. The increase in consumption seems to go hand in hand with software degradation. N. Amsel et al. [94] speak of "feature creep", "second-system effect" or "software bloat" to express this degradation.

A.2 Inform users about the evolution of the environmental impact as the software is updated

N. Amsel et al. [94] conducted a survey, using Amazon's Mechanical Turk, to find the users' sensibility on software sustainability during software updates. The results observed were that users are often unaware of the problem of software sustainability. In addition, none of the respondents takes into account the sustainability dimension when they update softwares they use. This means, among other things, that companies may have difficulties in highlighting the sustainability of their software as an added value.

A.3 Modify the software to reduce its energy impact

Regarding the use of refactoring to reduce the energy impact of the software, G. Pinto et al. [96] have done a review of the literature on this subject. The authors were interested at the application level and observed that there was no implementation at the time of their research with the aim of reducing the energy impact. Nevertheless, they have gone through the literature to find implementation opportunities but the implementation work remains to be done[3].

5.3.4.4 Conclusion

Structure coverage

Based on the table 5.4 our observations on the state of the literature are as follows:

First, we can classify the environmental impacts studied in the literature for the maintenance phase into two impacts categories:

- Maintenance process execution on its own carbon footprint
- Changes made to software on software energy consumption

[3]In the industry, however, we can cite GreenSpector, which offers business refactoring services to improve the energy efficiency and user experience of mobile applications and websites. https://greenspector.com/

Impacts	Calculation models	Solutions identified
Maintenance process execution on its own carbon footprint	Eva Kern et al. [53] (Commentary) J. Taina [52] (Problem identification and solution)	
Changes made to software on software energy consumption		N. Amsel et al. [94] (Commentary) G. Pinto et al. [96] (Literature Review) A. Hindle [84] (Case study)

Table 5.4: Table of research works by impacts identified and by contribution

Secondly, we can observe that the small number of researches we have found does not allow us to have at least an impact calculation model and a solution for the same impact.

Finally, concerning the types of research works, we can observe that most of the research works are not based on research. The only research-based article is in the form of problem identification and solution and we have not found any validation or evaluation research.

Impacts

Concerning the impacts studied, they concern the maintenance process itself and the impact on the operation phase of software maintenance. However, regarding the latter impact, no study was found regarding the impact of bug fixing, or adding features in particular, on the resource consumption of the software. In addition, researches are still needed to determine the effect of the use of solutions in previous phases on the efforts required to carry out the maintenance phase. Indeed, some practices may have the effect of reducing the effort required to maintain the software, such as reducing the number of unused features for example and others may have the effect of increasing the effort required to maintain the software, by making the code more difficult to maintain if its optimization makes it difficult to understand, for example.

Calculation models

Concerning impact calculation models, we only found calculations models about the maintenance process impact itself and none about the impact on software's operation phase.

Solutions

Regarding solutions, we found no author dealing with the possibility of using refactoring to reduce the environmental impact of software already developed, except for the work of G. Pinto et al. [96] who have done a review of the literature on this subject and observed that there was no implementation at the time of their research with the aim of reducing the energy impact. In addition, A. Hindle [84] believes that too little number of studies has been conducted to study the energy evolution of revisions and that developers should be made aware of this. Finally, we believed that maintenance phase could be the one that most influences the software's life because we estimate that the longer the life of a software is, the longer the life of the hardware supporting it. However, a longer life does not mean that successive changes do not increase resource requirements over time. To limit this risk and really slow the renewal cycles of software and hardware, solutions should be studied and implemented in future research.

5.3.5 Operation

5.3.5.1 Impacts

The authors who dealt with this fifth phase identified four types of impact.

Various sources on software operation environmental impact J. Bonvoisin [35] identified that different sources can influence the general environmental impacts of software operation. Eva Kern et al. [53] identified some parameters that could influence CO2 emissions of the operation phase: the type of software, the estimated length of use, the need to buy new equipment or not, the amount and the type of user.

Application Performance on software energy consumption Alexander Kipp et al [22] collected "green" metrics related to multiple levels of IT service center infrastructure, including the application level. In the latter, they consider that metrics mainly related to the performance of an application (Response time, Availability,...) influence the energy consumption generated by the application so that the higher the performance of an application, the greater its weight on the infrastructure, and therefore the energy consumption that results from it, will be high.

Customer behavior on software energy consumption C. Atkinson et al. [24] identified that different type of customer of an IT service has different effect on the energy consumption of the hosting provider. N. Amsel et al. [94] consider that users who are aware of the environmental impact of the software they use can modify their behavior and thus reduce the energy consumption generated by their activities.

Software administration on operational phase environmental impact Regarding software administration, S. Naumann et al [5] identified several sources of impact, such as the installation of the software, its updates and the support to the users.

5.3.5.2 Calculation models

A Various sources on software operation environmental impact

A.1 Calculate environmental impact of the operating phase

J. Bonvoisin [35] proposed an environmental impact model of information services. These services are processes that generate information from raw data. The information services considered by the author are used for decision making. He does not consider cultural or entertainment services. The proposed model has a generic nature. The categories of impacts measured are not specified, they are left to the choice of the author of the calculation. However, even if the design of the model leaves flexibility in the choice of types of impacts to be considered, the model is limited to calculating the impacts of the operation phase, the other phases being ignored by the author.

5.3.5.3 Solutions

A Application Performance on software energy consumption

A.1 Create new branches of Green computing

Alexander Kipp et al. [22] respond to the problem of the difficulty to measure the environmental impact for each software layer. The authors of the article intend to create a new branch of Green Computing, in two parts :

- "The co-design of energy-aware information systems and their underlying services and IT service center architectures in order to satisfy users requirements such as performance and QoS, as well as green environment requirements through emissions control as a result of an energy efficient usage"
- "The run-time management of IT Center energy efficiency, which will exploit the adaptive behavior of the system at run time, both at the application and IT architecture levels."

B Customer behavior on software energy consumption

B.1 Influence user behaviour

According to C. Atkinson et al. [24], during the operation phase, the data centers that host the IT services do not have the possibility to see or modify their implementation in order to optimize their impact on energy consumption. However, the authors identified that the behavior of service users influences the energy consumption generated by them. They therefore propose two solutions aimed at influencing user behavior. The first solution is based on Green Specifications. This would consist in limiting the use of a service to a user on the basis of a quantity of environmental impact, such as the CO2 emissions linked to the use of the service, which the user would have the budget to consume. The other solution would be to adapt billing to the state of the electricity network, such as increasing the prices of services consuming the most resources when the network

Impacts	Calculation models	Solutions identified
Various sources on software operation environmental impact	J. Bonvoisin [35] (Problem identification and solution)	
Application performance on software energy consumption		Alexander Kipp et al [22] (Problem identification and solution)
Customer behavior on software energy consumption		N. Amsel et al. [94] (Commentary) Atkinson et al. [24] (Commentary)

Table 5.5: Table of research works by impacts identified and by contribution

is heavily used and/or the origin of the electricity produced is not renewable.

N. Amsel et al. [94] have developed a tool called GreenTracker for users to help them compare the energy consumption of applications of the same functionality and make them aware of the differences that may exist.

5.3.5.4 Conclusion

Structure coverage

Based on the table 5.5 our observations on the state of the literature are as follows:

First we can classify the environmental impacts studied in the literature for the operation phase into three impacts categories:

- Various sources on software operation environmental impact
- Application Performance on software energy consumption
- Customer behavior on software energy consumption

Secondly, as for the previous phase, research works that we found does not allow us to have at least an impact calculation model and a solution for the same impact.

Finally, concerning the types of research works, we can observe that the distribution is equitable between research-based and non research-based research works.

Impacts

Concerning the impacts studied, these all concern the use of the software. However, we believe that direct impacts of this phase could have three causes, the distribution, use and administration of the software.

Regarding the distribution, it would differ according to the type of software. For an embedded software, the distribution of the software would not be different from that of the hardware and would therefore have no additional impact on the impact of the distribution of the equipment. In the case of a mobile application or most public software, the dematerialized distribution is the most widespread. The intangible distribution of the software is composed of several stages. According to J. Taina [97], four steps are identifiable: the sending of the software to the download platforms, the upload of the software on these platforms, the sale of the software and the downloading of the software by the customers. In the case of MIS, cloud services or website, the impacts of the distribution phase are not distinct from the installation and integration phases, which are different from one product to another.

Concerning the use of the software, the environmental impacts could potentially be very different between each occurrence of the software. Indeed, we think that environmental impacts depend on the infrastructure on which the software is run and how the software is used. It would also be necessary to differentiate the impact of a software occurrence, which can be measured, and the impact of all occurrences of the software, which would be difficult to measure accurately without a precise measure of all occurrences. In addition, it would be only possible to measure software environmental impact a posteriori because it depends on its duration of use.

Regarding software administration, in addition to the sources identified by S. Naumann et al [5] (installation of the software, its updates and the support to the users), we think that when the software is no longer used and is replaced, data migration, which consist of transforming the data into a readable format for the new software, would also be a source of impact. Finally, the legislation would also be a direct source of impact because it imposes a backup of the logs concerning the personal data, as well as the data itself for a certain amount of time.

Calculation models

Concerning impact calculation models, the work of J. Bonvoisin [35] is the only one we could find on this subject. However, this remains at a very abstract level because it focuses on the digital service in general and it is difficult to imagine the concrete use of it.

Solutions

Regarding solutions, except for N. Amsel et al. [94] that would still allow the user to generate this information for their hardware, no research works dealing with the operation phase presents the idea that a software should provide information about the environmental impact of its use and development so that the user can adapt his use of it (the work of J. Cabot et al. [55] in the phase of requirement nevertheless defends this idea). We believe that this type of information could even be available before the software is acquired to allow, for example, the potential user to compare the environmental impacts of software providing the same service. However, the collection of such information would require the establishment of a monitoring and follow-up system to ensure that such information are correct. This type of systems could be built by drawing inspiration from existing monitoring systems in the world of software or manufacturing industries. Regarding the solution of Atkinson et al. [24], this one is based, among other things, on the idea that it would be possible to precisely measure the CO2 emissions of each code and data artifact (e.g. CO2 consumption per minutes of film viewed, for a given film), which is not possible at the moment.

Another solution, which we have not found in the literature, that could influence users' choice towards software that has less impact on the environment, is to use IT service billing methods that encourage the user to use as few resource as possible. However, even if this type of payment already exists for certain types of services (in the case of web services for example), they have not been created with the aim of reducing environmental impact and research is needed to assess the sustainability of this type of solution.

5.3.6 End of life

5.3.6.1 Impacts

We found no impacts of the end-of-life stage of the software in the literature.

5.3.6.2 Calculation models

We found no model for calculating the impacts of the end-of-life stage of the software.

5.3.6.3 Solutions

We found no solution for the end-of-life stage of the software.

5.3.6.4 Conclusion

Structure coverage

The fact that we did not find any research work in this phase may be due to the search method used. Indeed, it may have overshadowed the relevant papers because of a wrong selection of keywords or the use of inappropriate databases. Since the end-of-life phase is not a phase commonly integrated into software development standards, it has perhaps been treated under terms and concepts that have escaped us. However, without additional information, we consider that no environmental impact, impact assessment or solution aimed at limiting this impact have so far been studied.

Impacts

Environmental impacts have not yet been studied for this phase. Nevertheless, we estimate that impacts would be related to the recycling process of software elements. Even if the recycling process could be assimilated to the maintenance process because both processes involve a modification or destruction of an existing code, we consider that the maintenance stops when the software or the library changes identity.

Future studies could address the temporal dimension that exists between software life cycles, between the end of a cycle and the beginning of an other. The impacts of the transition from one generation to the next, and the means to implement, to limit these impacts could prove to be important information in the context of software eco-design. Other studies could focus on the environmental impact of data migration and transformation between a software and its replacement (whether this replacement is a more recent version of the software previously used or a completely different software). Finally, studies could address the environmental impact of destroying data and software when their storage is no longer necessary.

Calculation models

Although the impact assessment has not yet been done in the literature, we can identify several interesting concepts concerning the reuse of software elements. Indeed, the reuse of software artifact has been the subject of much research, even if it is not yet optimal in the industry, as suggested V. Bauer and A. Vetro [98]. First of all, the reuse is applicable to several levels of abstraction (knowledge, object, components, subsystem) and therefore has theoretically positive effects on development phases, with the exception of the maintenance phase, which may be more difficult when compatibility issues arise, according to Dabhade et al. [99] who carried out a review of the literature on software reuse.

Solutions

Finally, according to Mr. Dabhade et al. [99], the reuse process requires effort, both in the collection of artifacts and in their use. Indeed, during the collection of a software artifact for future reuse, it is necessary, among other things, to document the function, the interface and the protocols used by this artifact. In addition, it is necessary to make available and classify the elements to be reused, in a library for example. Finally, the authors identify three problems when using artifacts: the lack of support tool for integration, the "Not-Invented-Here Syndrome" of some developers preferring, by default, to develop themselves and the identification of relevant artifacts during the software design process, which can be difficult depending on the application domain and the developer community.

5.4 Effects between software life cycle phases

The goal of this section is to build a visual representation of all the effects, identified in the studied scientific literature, on the environmental impact of a phase due to the completion of a phase of the software life cycle. As a reminder, research works gathered here provide either a calculation (or an intuition of it) of the effect or a solution aimed at limiting this effect if it is negative for the environment (and vice versa).

The focus for the following visual representation is on the effects themselves, i.e. the source and destination of the effects, and the need to increase or decrease each effect in order to limit the overall environmental impact. The opportunity was also taken to add research works addressing each effect, to observe its frequency of identification in the literature.

The global scheme is shown in the figure 5.7 and targeted views on the impacts identified for each phase are shown in the following figures.

The graphical notations are as follows:

- *Yellow rectangles* represent the sources of impacts. They are in the phase in which this source of impact appears.
- *Yellow ovals* represent environmental impact targets. They are in the phase during which the environmental impact occurs.
- *Grey ovals* represent impact targets whose environmental character is not proven. They are in the phase in which the impact occurs.
- *Red hexagons* represent sustainability impact targets, i.e. more than an environmental impact. They are in the phase in which the impact occurs.
- *Arrows with solid lines* each represent an influence from a source to a target, the effect of which is negative for the environment.
- *Arrows with dotted lines* each represent an influence from a source to a target, whose effect is positive for the environment.

5.4.1 Observations phase by phase

The *requirement phase* (see figure 5.8) is the only phase whose identified impacts have targets independent of the life cycle phases. The impact targets are also the most diversified in this phase. This may be due in part to the fact that this phase is most strongly linked to the reasoning on the impacts of the software developed on its future environment (business or other) in which the future software will evolve.

The *design phase* (see figure 5.9) is the only phase that is not the subject of an impact target, even within the impact sources located in this same phase. This observation should be put into perspective, however, since there is no reason for research works of J. Taina [52] and Eva Kern et al. [53] could not be applicable to this phase.

The *development phase* (see figure 5.10) is clearly the most studied in terms of source of environmental impact, both in terms of the number of different sources identified and the number of articles having identified these impacts. One reason could be that this phase is considered the core of software construction. Another reason could be that this is the phase during which the concept of optimization of the software product can be applied and this search

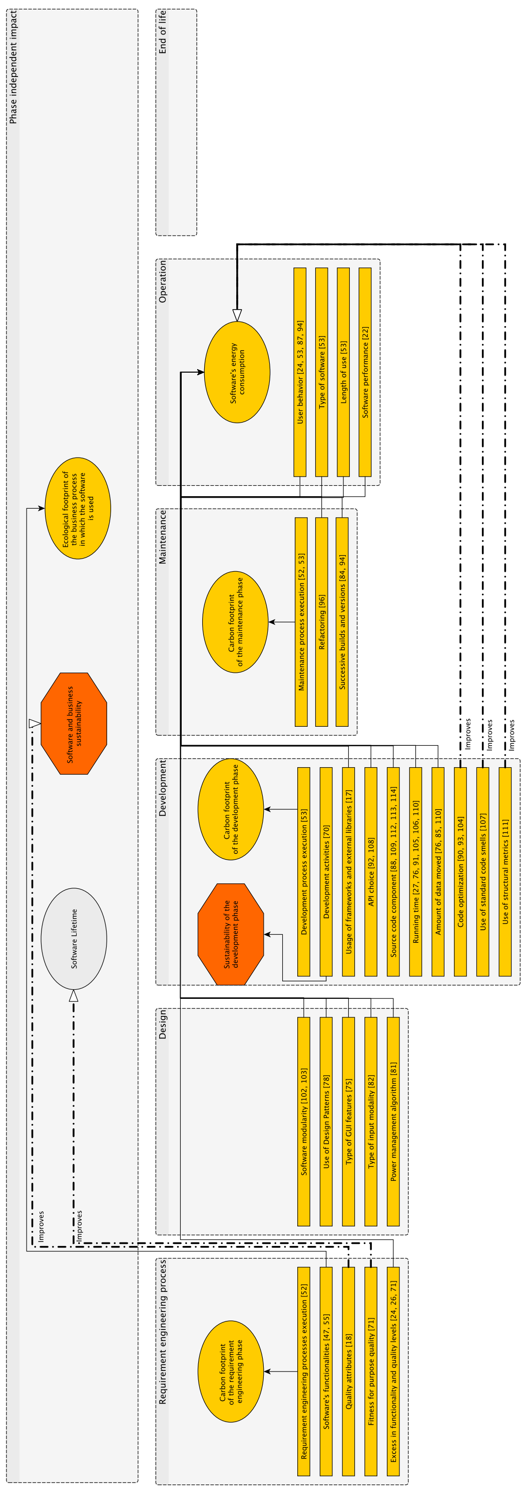
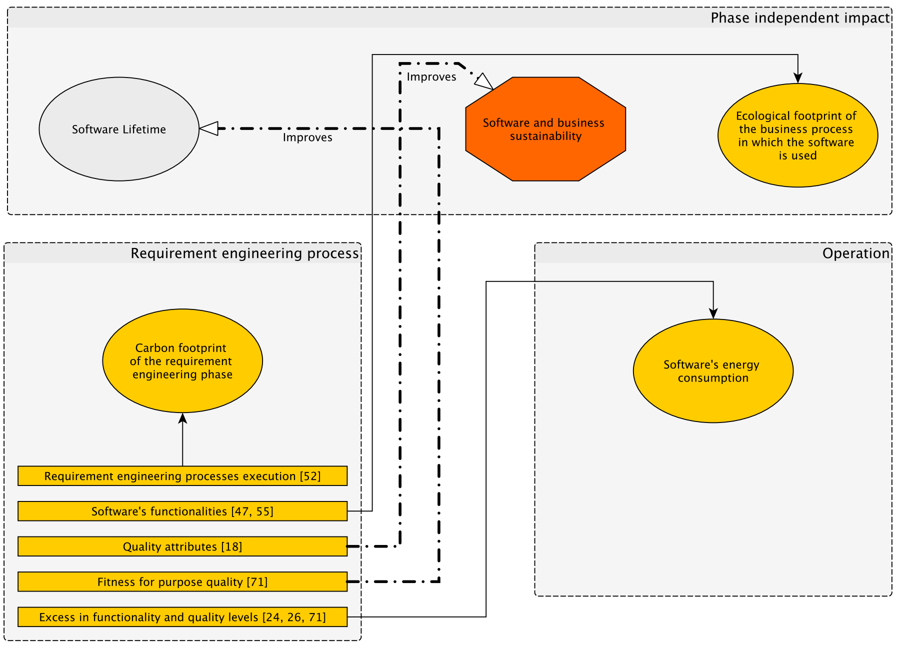

Phase independent impact
Ecological footprint of the business process in which the software is used
Software and business sustainability
Software Lifetime
Improves
Improves
End of life
Operation
Software's energy consumption
User behavior [24, 53, 87, 94]
Type of software [53]
Length of use [53]
Software performance [22]
Maintenance
Carbon footprint of the maintenance phase
Maintenance process execution [52, 53]
Refactoring [96]
Successive builds and versions [84, 94]
Improves
Improves
Improves
Development
Sustainability of the development phase
Carbon footprint of the development phase
Development process execution [53]
Development activities [70]
Usage of frameworks and external libraries [17]
API choice [92, 108]
Source code component [88, 109, 112, 113, 114]
Running time [27, 76, 91, 105, 106, 110]
Amount of data moved [76, 85, 110]
Code optimization [90, 93, 104]
Use of standard code smells [107]
Use of structural metrics [111]
Design
Software modularity [102, 103]
Use of Design Patterns [78]
Type of GUI features [75]
Type of input modality [82]
Power management algorithm [81]
Requirement engineering process
Carbon footprint of the requirement engineering phase
Requirement engineering processes execution [52]
Software's functionalities [47, 55]
Quality attributes [18]
Fitness for purpose quality [71]
Excess in functionality and quality levels [24, 26, 71]

Figure 5.7: Visual representation of sources and targets of impacts identified by the literature, sorted by life cycle phase



Phase independent impact
Improves
Software Lifetime
Improves
Software and business sustainability
Ecological footprint of the business process in which the software is used
Requirement engineering process
Carbon footprint of the requirement engineering phase
Requirement engineering processes execution [52]
Software's functionalities [47, 55]
Quality attributes [18]
Fitness for purpose quality [71]
Excess in functionality and quality levels [24, 26, 71]
Operation
Software's energy consumption

Figure 5.8: Representation of impacts whose sources are in the requirement phase

for optimization could be unconsciously the first reaction when addressing the environmental impacts of the software. This reason would be consistent with the fact that the most widely identified target is the energy consumption of the software during its *operation phase* (see figure 5.12). However, the above two reasons could also apply to the *maintenance phase* (see figure 5.11). However, this is one of the phases with the lowest number of impacts identified.

5.4.2 General observations

All impacts whose source and target are in different phases, target the operation phase. This is due to the over-representation of the software's energy consumption as an impact target.

Concerning the sources of impacts, outsourcing and reuse (of all kinds) have not been studied. These sources of impacts could be found in all phases of the life cycle and are therefore, in our opinion, potentially important sources of environmental impact.

Concerning the impacts that we have not found in the literature, it is strange to observe that none of the authors has studied the impact of requirement, development or design decisions on the maintenance phase, however often one of the most, if not the most important phase in terms of work effort, given its duration.

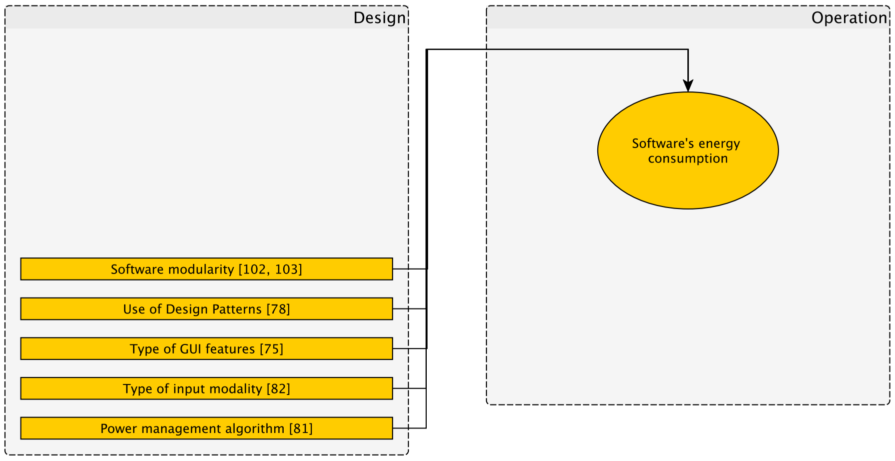

Design
Operation
Software's energy consumption
Software modularity [102, 103]
Use of Design Patterns [78]
Type of GUI features [75]
Type of input modality [82]
Power management algorithm [81]

Figure 5.9: Representation of impacts whose sources are in the design phase

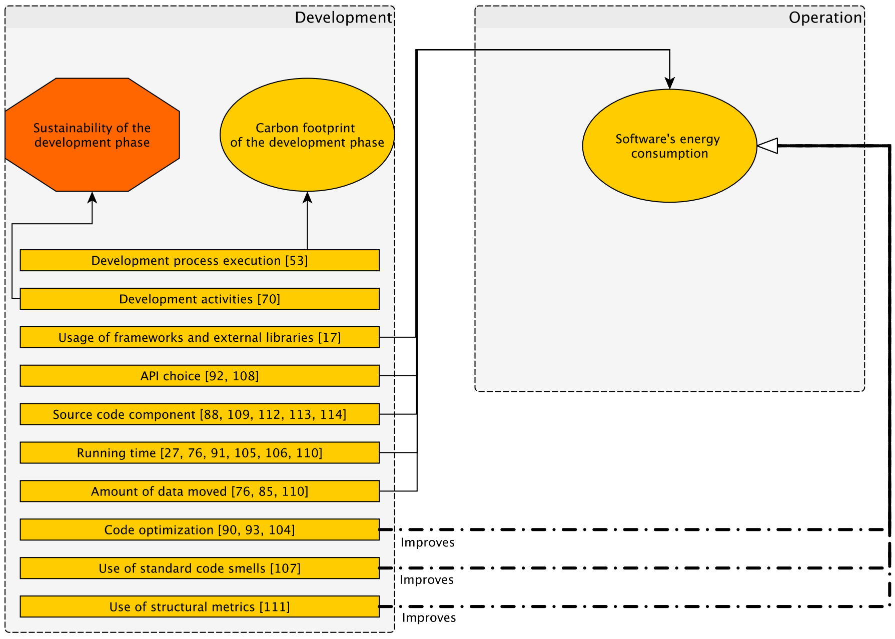

Development
Operation
Sustainability of the development phase
Carbon footprint of the development phase
Software's energy consumption
Development process execution [53]
Development activities [70]
Usage of frameworks and external libraries [17]
API choice [92, 108]
Source code component [88, 109, 112, 113, 114]
Running time [27, 76, 91, 105, 106, 110]
Amount of data moved [76, 85, 110]
Code optimization [90, 93, 104]
Use of standard code smells [107]
Use of structural metrics [111]
Improves
Improves
Improves

Figure 5.10: Representation of impacts whose sources are in the development phase

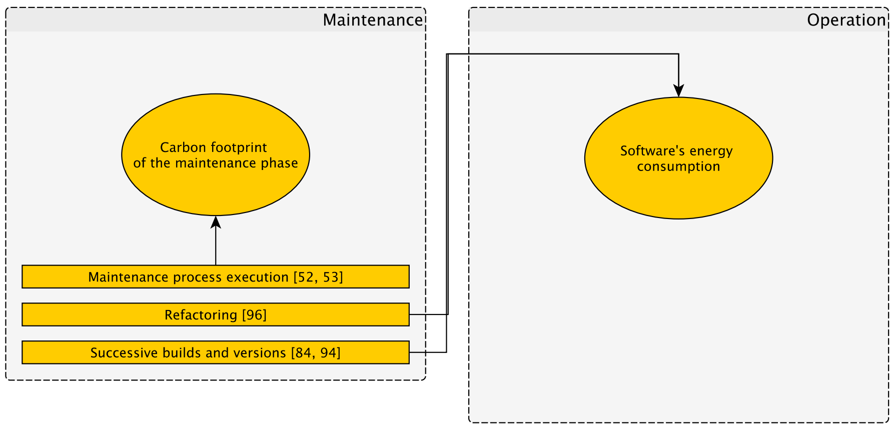

Maintenance
Operation
Carbon footprint of the maintenance phase
Software's energy consumption
Maintenance process execution [52, 53]
Refactoring [96]
Successive builds and versions [84, 94]

Figure 5.11: Representation of impacts whose sources are in the maintenance phase

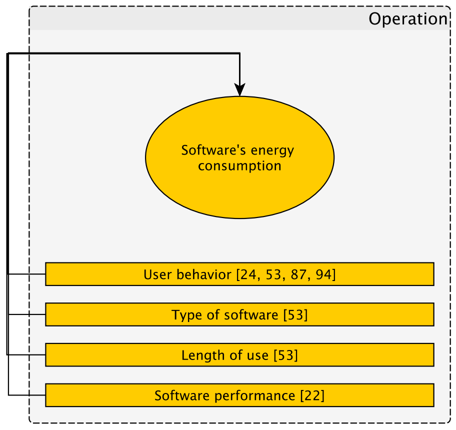

Operation
Software's energy consumption
User behavior [24, 53, 87, 94]
Type of software [53]
Length of use [53]
Software performance [22]

Figure 5.12: Representation of impacts whose sources are in the operation phase

Chapter 6

Research Challenges

This chapter presents areas in which future research work on software eco-design could be carried out in order to advance the field. Motivations behind the topics considered were derived from two different sources:

- The conclusions drawn in each phase of the survey carried out in this thesis, classified according to the structure used in the survey (i.e. impacts, calculation models and solutions)
- The challenges identified in the literature, linked to software eco-design in general, which fall within the scope of this thesis

For each research challenge, the motivation and the area in which future research work on software eco-design could be carried out is described. Research challenges are classified according to whether they concern :

1. Environmental impacts of the software life cycle
2. Models for calculating these environmental impacts
3. Solutions to limit these environmental impacts
4. General challenges concerning software eco-design

6.1 Environmental impacts of the software life cycle

This section addresses the research challenges related to more than one phase of the life cycle of environmental impacts. Research challenges related to a unique phase have already been identified in the conclusion of the corresponding phases.

6.1.1 Lack of data on the impact of reuse

The environmental impact of reuse, at all abstraction levels, has not yet been studied. Yet, the benefits of reuse could be to reduce the working time in the phase where reuse is used. Moreover, no impact of the end-of-life phase, has been the subject of research.

Therefore, directions for additional research could be to identify the environmental impacts of reuse throughout the life of the software. The reuse can be for example the use of: architectural patterns (to reuse knowledge), application frameworks (for reuse of application subsystems), software product lines (for reuse of architectures) or reuse of application system.

6.2 Models for calculating these environmental impacts

This section addresses the research challenges related to more than one phase of the life cycle of models of calculation. Research challenges related to a unique phase have already been identified in the conclusion of the corresponding phases.

6.2.1 Lack of data about environmental impacts of high-level design decisions

Today, it is not possible to obtain an estimate of the environmental impact of design decisions made before the development phase. It is in this context that the idea of a bottom-up mapping as defended by C. Sahin et al. [26] is interesting, because it would allow to build a link between the source code and its environmental impact but also between the design decisions that led to the source code and their environmental impact.

Therefore, directions for additional research could be to establish trends and effects between design decisions (broadly speaking) and their impact on energy consumption by doing two things. In the first place, use current methods to measure source code energy consumption: software methods, such as the one developed by H. Acar et al. [27], are the easiest to use, even if they are less accurate than hardware methods, such as the one developed by D. Li et al. [114]. Then use traceability techniques to establish links between the code and the associated high-level design elements, and repeat the work on a sufficient number of software artifacts and projects.

6.3 Solutions to limit environmental impacts

This section addresses the research challenges related to more than one phase of the life cycle of solutions to limit environmental impacts. Research challenges related to a unique phase have already been identified in the conclusion of the corresponding phases.

6.3.1 Lack of consistency of research related to solutions

With the exception of the development phase, which is the phase in which the research work has focused the most, no article has been found in the framework of this thesis, offering both the description of a model for calculating an environmental impact and solutions aimed at reducing this impact. It was also not possible to find different articles dealing with both a calculation model and a solution for the same environmental impact.

Therefore, directions for additional research could be to consolidate current research to obtain a calculation model for each impact for which a solution exists so that practitioners can verify that a solution has had the desired effect.

6.3.2 Side effects of solutions on other sources of impacts

The global nature of software eco-design means that the entire software life cycle must reduce its environmental impact and that it is therefore necessary to avoid that the reduction of a negative impact leads to an increase in another negative impact. In concrete terms, an idea addressed by J. Taina [52] and Eva Kern et al. [53] is that reducing human efforts to achieve a life cycle phase limits its environmental impact. However, the current means used to reduce human efforts would decrease the quality of the work carried out in this phase and therefore, would increase the environmental impact of the following phases. For example, Eva Kern et al. [53] consider that the automatic generation of code, which would reduce the environmental impact of the development phase, would potentially increase that of the operation phase, but cannot say, in the current state of knowledge, in what proportions this influence is exerted.

Therefore, directions for additional research could be to when creating and evaluating a solution to reduce a negative impact, also assess the impact on other sources of negative impacts.

6.4 General challenges concerning software eco-design

The general challenges described in this section are those which do not concern an impact or an associated solution, but which concern software eco-design in the broad sense.

6.4.1 Lack of agreement on the definition of software sustainability in the literature

C. C. Venters [31] explored existing definitions of software sustainability and observed that there were many existing definitions, each with different and sometimes even contradictory points of view (as seen in chapter 3).

Therefore, directions for additional research could be to, according to C. C. Venters [31], establish a common definition of software sustainability in order not to risk that the efforts put into research related to software sustainability are not effective.

6.4.2 Confusion between sustainability and longevity in the literature

Sustainable software is sometimes used in the literature to denote software that lasts over time. For example, D. S. Katz et al. [32], in a report on a workshop on sustainable software, defined sustainable software as : "Sustainable software has the capacity to endure such that it will continue to be available in the future, on new platforms, meeting new needs.". However, we haven't encountered proofs regarding that longevity can effectively reduce the environmental impact of software and the business processes that use it. For example, it is not possible today to know if a software designed to be used over a long period of time will help to combat hardware obsolescence.

Therefore, directions for additional research could be to estimate whether designing software with a longer lifespan effectively reduces its environmental impact. It could also be akin to seeking to facilitate comparative life cycle analyses between the impact of software in use and the impact of its replacement. As part of this research, the work of F. Ardente and F. Mathieux [66], could be an interesting basis. Even if the paper scope does not concern software, this work focuses on products whose use consumes energy, which N. van Nes and J. Cramer [80] have identified as an important factor for the interest of designing products with a longer lifespan.

6.4.3 Lack of accurate information on factors influencing software lifespan

If it turns out that longevity is compatible with the objectives of software eco-design, then it would be necessary to control the factors influencing the lifespan of the software. These can be, for example, the changing needs of users or the changing operating environment... This would also allow to know, for example, for a legacy system, the moment when it is better to replace it. In the context of manufactured products, N. van Nes and J. Cramer [80], have identified the factors that influence their lifespan, so that designers know what criteria to work on to increase the lifespan of products. However, this work is not entirely applicable to software and would require adaptations.

Therefore, directions for additional research could be to identify the specific factors that influence the lifespan of software to enable designers to maximize it.

6.4.4 Lack of use of renewable energy

Electricity generated by renewable energies is not accessible at all times and in all places. Eva Kern et al. [53] consider that it should be possible, when it is not possible to run software locally using green energy (or will not be possible if the software is still under development), to use a version running on a cloud running on green energy (or to adapt the architecture, in the case of software under development, so that its execution takes place in a green data center). The problem is that this architectural choice implies the generation of additional network usage, to transfer data related to the use of software running in a data center, that must be estimated in order not to lose the benefits associated with the use of renewable energy.

Therefore, directions for additional research could be to create calculation models in order to be able to estimate whether it is more advantageous to develop, or more generally, to use, software in the form of software-as-a-service (a server side execution) or in the form of software running at the client. F. A. Ali et al. [85] started this work for calculation models at runtime, as part of the dynamic offloading of mobile application.

Chapter 7

Conclusion

The first objective of this thesis was to give a clear definition of the concept of software eco-design. To this end, in Chapter 3, a definition has been formulated on the basis of existing information in the literature: *Software eco-design is the integration of environmental constraints in all software development processes, in order to reduce the environmental impact of the software during its life cycle, this integration being individually adapted to the organization developing the software, stakeholders and the type of software developed.* Then, in order to meet the second objective, which was to present current knowledge in the software eco-design approach, a literature review was carried out in chapter 5. In this review, emphasis was placed on environmental impacts, their calculation models and solutions in order to reduce their negative effects on the environment, for each phase of the software life cycle. Following this review, the factors influencing the environmental impact of the software life cycle, identified in the literature, were gathered in order to extract current trends. Finally, in Chapter 6, direction for additional research, based on identified research challenges, were presented.

The contribution of this work is the exploration and presentation of existing research works related to the technical part of the software eco-design approach. The aim is to provide a starting point for all members of the academic community interested in software eco-design.

7.1 Critical perspectives

As mentioned above, this thesis focused on the technical part of the software eco-design approach. Nevertheless, other parts of this process may be undertaken in the future:

- **Current challenges linked to the cohabitation between project management and the use of eco-design**. In the definition of eco-design software established in this thesis, eco-design is considered to be adapted to the company developing the software and therefore to its way of managing IT projects. This has not been taken into account in the scope of this thesis but could be in the future. A basis for this research would be the article by F. Brones et al. [126] which brings to the fore the difficulty of making project management and general eco-design coexisting.

- **Current challenges related to eco-design efficiency**. In order to prevent the positive effects of eco-design from being lost, the potential rebound effects should be estimated. For example, find barriers to set up in order to avoid that the time saved by the user, if the execution time of a software is decreased for example, is offset by a greater use of other applications, which would negate the positive effect; Or, in order to prevent the use of eco-designed software would make the user feel safer and increase their use, potentially offsetting the initial gains.

- **Current challenges related to the effectiveness of integrating environmental constraints for all types of development**. Indeed, in the future, establishing which development forms are best suited or least suited to software eco-design would be interesting in order to increase its effectiveness and spread. With that in mind, S. S. Mahmoud et al. [70] have already put forward the idea that the linear sequential life cycle development processes would be less suitable than the iterative development processes to develop sustainable software and S. Bhattacharya et al. [103] suggest that for some types of software, lean software development has shown its effectiveness in building environmentally effective software.

- **Current challenges related to difficulties in integrating environmental constraints by practitioners**. An interesting article already having results for requirement engineers is R. Chitchyan et al. [127]. More generally, the motivations linked to eco-design could be interesting to gather, such as the use of outsourcing, the number of copies distributed or the type of distribution (internal or external to the company) of software.

Finally, one way of strengthening this work without extending its scope would be to carry out a more systematic literature review.

This thesis was largely based on a scientific article carried out as part of a research internship with the French research group EcoInfo. It is currently[1] being reviewed for publication in the journal ACM Computing Surveys (CSUR).

[1]August 2018

Bibliography

[1] Andrew Dobson. Environment sustainabilities: An analysis and a typology. Env. Politics, 5, 1996

[2] United Nations World Commission on Environment and Development. Report: Our Common Future. In United Nations Conference on Environment and Development, 1987

[3] B. Penzenstadler, "Towards a Definition of Sustainability in and for Software Engineering", 28th ACM Annual Symposium on Applied Computing (SAC), (2013), pp. 1-3

[4] Lorenz Hilty et al. The relevance of information and communication technologies for environmental sustainability. Environm. Modelling & Software, 21(11), 2006.

[5] S. Naumann, M. Dick, E. Kern, and T. Johann, "The GREENSOFT Model: A reference model for green and sustainable software and its engineering," Sustainable Computing: Informatics and Systems, vol. 1, no. 4, pp. 294-304, Dec. 2011.

[6] M. Dick, S. Naumann, and N. Kuhn, "A model and selected instances of green and sustainable software," What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience, pp. 248-259, 2010.

[7] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," Information and Software Technology, vol. 55, no. 12, pp. 2049-2075, Dec. 2013.

[8] Lami, G., Fabbrini, F., Fusani, M.: Software sustainability from a process-centric perspective. In: Winkler, D., OConnor, R.V., Messnarz R. (eds.) EuroSPI 2012, CCIS 301, pp. 97-108. Springer, Berlin, (2012).

[9] Winkler, Dietmar, Rory V. OConnor, et Richard Messnarz, éd. Systems, Software and Services Process Improvement. Vol. 301. Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[10] M. Vautier and O. Philippot, "Is "software eco-design" a solution to reduce the environmental impact of electronic equipments?," 2016 Electronics Goes Green 2016 (EGG), Berlin, Germany, 2016, pp. 1-6

[11] Taina, J.: Good, Bad, and Beautiful Software - In Search of Green Software Quality Factors. CEPIS upgrade European journal Exploring Initial challenges for green software engineering, volume XII, pp. 22-27 (2011).

[12] Hilty, Lorenz M., et Bernard Aebischer, éd. ICT Innovations for Sustainability. Vol. 310. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2015.

[13] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch, ”Sustainability in software engineering: A systematic literature review,” 2012.

[14] DICK, Markus et NAUMANN, Stefan. Enhancing Software Engineering Processes towards Sustainable Software Product Design. In : EnviroInfo. 2010. p. 706-715

[15] M. Dastbaz, C. Pattinson, and B. Akhgar, Eds., Green information technology: a sustainable approach. Waltham, MA, USA: Elsevier, Morgan Kaufmann, 2015.

[16] Sierszecki, K .; Mikkonen, T Steffens, M .; Fogdal, T., Savolainen, J., ”Green Software: Greening What and How Much ?,” Software, IEEE, vol.31, no.3, pp.64,68, May-June 2014

[17] E. Capra, C. Francalanci, and S. A. Slaughter, ”Is software ”green”? Application development environments and energy efficiency in open source applications,” Information and Software Technology, vol. 54, no. 1, pp. 60-71, Jan. 2012.

[18] B. Penzenstadler, A. Raturi, D. Richardson, and B. Tomlinson, ”Safety, security, now sustainability: The nonfunctional requirement for the 21st century,” IEEE software, vol. 31, no. 3, pp. 40-47, 2014.

[19] P. Lago, Q. Gu, P. Bozzelli, and others, ”A systematic literature review of green software metrics,” 2014.

[20] Philippot, O. (2017). Eco-conception logicielle : retour d’expérience Greenspector, Impact des logiciels sur lenvironnement, quid de léco-conception ?, Grenoble, 2017. http://ecoinfo.cnrs.fr/?p=12072

[21] E. Capra, G. Formenti, C. Francalanci, and S. Gallazzi, ”The Impact of MIS Software on IT Energy Consumption.,” in ECIS, 2010, p. 95.

[22] A. Kipp, T. Jiang, M. Fugini, and I. Salomie, ”Layered Green Performance Indicators,” Future Generation Computer Systems, vol. 28, no. 2, pp. 478-489, Feb. 2012.

[23] Dr. Steigerwald B., Agrawal A., Software & Services Groupe, INTEL Corporation, Folsom, CA, USA - ”Developing Green Software : generic model for sustainability with process and product specific instances”, June 23, 2011.

[24] Atkinson, C.; Schulze, T.; Klingert, S., ”Facilitating Greener IT through Green Specifications,” Software, IEEE , vol.31, no.3, pp.56,63, May-June 2014.

[25] F. Albertao, J. Xiao, C. Tian, Y. Lu, K. Q. Zhang, and C. Liu, "Measuring the Sustainability Performance of Software Projects" 2010, pp. 369-373.

[26] C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Towards power reduction through improved software design," in Energytech, 2012 IEEE, 2012, pp. 1-6.

[27] H. Acar, G. Alptekin, J.-P. Gelas, and P. Ghodous, "The Impact of Source Code in Software on Power Consumption," International Journal of Electronic Business Management, vol. 14, pp. 42-52, 2016.

[28] J. Rockström et al., "A safe operating space for humanity," nature, vol. 461, no. 7263, pp. 472-475, 2009.

[29] Bihouix P., L'age des low-tech vers une civilisation techniquement soutenable, Paris, Editions du Seuil, 2014, 336 p

[30] S. Behrendt, W. Kahlenborn, M. Feil, C. Dereje, B. Raimund, D. Ruth, S. Michael, Rare Metals, Measures and Concepts for the Solution of the Problem of Conflict-aggravating Raw Material Extraction - The Example of Coltan, Umweltbundesamt, 2007.

[31] C. C. Venters, "Software Sustainability: The Modern Tower of Babel," p. 6.

[32] D. S. KATZ et al., "REPORT ON THE FOURTH WORKSHOP ON SUSTAINABLE SOFTWARE FOR SCIENCE: PRACTICE AND EXPERIENCES (WSSSPE4)," p. 46.

[33] Andrae, A.S.G., Andersson, D.R., Liu, J., 2005. Significance of intermediate production processes in life cycle assessment of electronic products assessed using a generic compact model. Journal of Cleaner Production, 13(13-14), 1269-1279

[34] Baudry, I., Lelah, A., Brissaud, D., 2012. Data Collection of Chemicals used in Microelectronic Manufacturing Processes for Environmental Studies. Présenté à : 19th CIRP Conference on Life Cycle Engineering, 23 mai 2012, Berkeley, Californie, Etats-Unis. In: Leveraging Technology for a Sustainable World, Springer, Berlin - Heidelberg, pp. 521-526.]

[35] J. Bonvoisin, "Analyse environnementale et éco-conception de services informationnels," Université Grenoble Alpes, 2012.

[36] Schütz, H., Moll, S., Bringezu, S., 2004. Rapport: Globalisation and the shifting environmental burden - Material flows of the European Union - Which globalisation is sustainable? (Wuppertal Papers - N134e - ISSN 0949-5266). Wuppertal Institut für Klima, Umwelt, Energie GmbH

[37] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester, "Overall ict footprint and green communication technologies," in ISCCSP10: 4th International Symposium on Communications, Control and Signal Processing, 2010, pp. 1 -6

[38] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "A preliminary study of the impact of software engineering on GreenIT," 2012, pp. 21-27.

[39] Erkman, S., 2004. Vers une écologie industrielle, éditions Charles Léopold Mayer.

[40] Flipo, F., Gossart, C., 2008. Infrastructure numérique et environnement: l'impossible domestication de l'effet rebond. Presented at: Colloque international services, innovation et développement durable, March 2008, Poitiers, France

[41] V. C. Coroama, D. Schien, C. Preist, and L. M. Hilty, "The energy intensity of the Internet: home and access networks," in ICT Innovations for Sustainability, Springer, 2015, pp. 137-155.

[42] Weber, C.L., Koomey, J.G., Matthews, H.S., 2010. The Energy and Climate Change Implications of Different Music Delivery Methods. Journal of Industrial Ecology, 14(5), 754-769

[43] Williams, E.D., Ayres, R.U., Heller, M., 2002. The 1.7 kilogram microchip: energy and material use in the production of semiconductor devices. Environ. Sci. Technol., 36(24), 5504-5510

[44] Mokhtarian, P.L., 2002. Telecommunications and Travel: The Case for Complementarity. Journal of Industrial Ecology, 6(2), 43-57.

[45] Cefriel A., "The Power of Software", 2008.

[46] Software Sustainability Institue Available at: http://www.software.ac.uk/about

[47] M. Mahaux, P. Heymans, and G. Saval, "Discovering sustainability requirements: an experience report," Requirements engineering: foundation for software quality, pp. 19-33, 2011.

[48] Loerincik, Y., 2006. Environmental impacts and benefits of information and communication technology infrastructure and services, using process and input output life cycle assessment, Thèse de Doctorat soutenue le 30 juin 2006, Ecole Polytechnique Fédérale de Lausanne.

[49] Greening, L.A., Greene, D.L., Difiglio, C., 2000. Energy efficiency and consumption - the rebound effect - a survey. Energy Policy, 28(6-7), 389-401.

[50] Hertwich, E.G., 2005. Consumption and the Rebound Effect: An Industrial Ecology Perspective. Journal of Industrial Ecology, 9(1-2), 85-98.

[51] L. M. Hilty, "Computing Efficiency, Sufficiency, and Self-sufficiency: A Model for Sustainability?," 2015.

[52] Taina J. How green is your software? In: Tyrvinen P, Cusumano MA, Jansen S, editors. Software Business, First International Conference, ICSOB 2010, Jyväskylä, Finland, June 21-23, 2010, Proceedings, Berlin, Heidelberg; 2010. p. 151-62.

[53] E. Kern, M. Dick, S. Naumann, and T. Hiller, "Impacts of software and its engineering on the carbon footprint of ICT," Environmental Impact Assessment Review, vol. 52, pp. 53-61, Apr. 2015.

[54] Vickery G, Mickoleit A. Greener and smarter: information technology can improve the environment in many ways. In: Noam EM, Pupillo LM, Kranz JJ, editors. Broadband networks, smart grids and climate change; 2013. p. 33-7.

[55] J. Cabot, S. Easterbrook, J. Horkoff, L. Lessard, S. Liaskos, and J.-N. Mazn, "Integrating sustainability in decision-making processes: A modelling strategy," in Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on, 2009, pp. 207-210.

[56] X. Franch, "The i framework: The way ahead," in Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on, 2012, pp. 1-3.

[57] B. Penzenstadler, A. Raturi, D. Richardson, C. Calero, H. Femmer, and X. Franch, "Systematic mapping study on software engineering for sustainability (SE4S)," in Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, p. 14.

[58] R. Abdullah, S. Abdullah, J. Din, and M. Tee, "A Systematic Literature Review Of Green Software Development In Collaborative Knowledge Management Environment," International Journal of Advanced Computer Technology (IJACT), vol. 9, p. 136, 2015.

[59] C. Marimuthu and K. Chandrasekaran, "Software Engineering Aspects of Green and Sustainable Software: A Systematic Mapping Study," 2017, pp. 34-44.

[60] I. Manotas et al., "An empirical study of practitioners perspectives on green software engineering," 2016, pp. 237-248.

[61] S. Kawamoto, M. Aoyagi, and Y. Ito, "Eco Design Guideline for Software Products," in Environmentally Conscious Design and Inverse Manufacturing, 2005. Eco Design 2005. Fourth International Symposium on, 2005, pp. 944-946.

[62] J. Pinto, M. Pérez, L. Mendoza, A. Grimán, and M. Kräuter, "Conceptual map of software environmental impact," AMCIS 2006 Proceedings, p. 487, 2006.

[63] C. Charbuillet and F. Vivat, Introduction: Quid de l'éco-conception? Objectifs, approches? Qu'est ce que réellement l'éco-conception? 2016.

[64] L. M. Hilty, P. Arnfalk, L. Erdmann, J. Goodman, M. Lehmann, and P. A. Wger, "The relevance of information and communication technologies for environmental sustainability - A prospective simulation study," Environmental Modelling & Software, vol. 21, no. 11, pp. 1618-1629, Nov. 2006.

[65] International Conference Informatics for Environmental Protection, CERN, and Gesellschaft für Informatik, Eds., Shring: proceedings of the 18th International Conference Informatics for Environmental Protection, October 21-23, 2004, CERN, Geneva (Switzerland). Genève: Éditions du Tricorne, 2004.

[66] F. Ardente and F. Mathieux, "Environmental assessment of the durability of energy-using products: method and application," Journal of Cleaner Production, vol. 74, pp. 62-73, Jul. 2014.

[67] C. Calero, M. F. Bertoa, and M. Á. Moraga, Sustainability and Quality: Icing on the Cake., in RE4SuSy@ RE, 2013.

[68] C. C. Venters et al., "The Blind Men and the Elephant: Towards an Empirical Evaluation Framework for Software Sustainability," Journal of Open Research Software, vol. 2, no. 1, Jul. 2014.

[69] S. S. Shenoy and R. Eeratta, "Green software development model: An approach towards sustainable software development," in India Conference (INDICON), 2011 Annual IEEE, 2011, pp. 1-6.

[70] S. S. Mahmoud and I. Ahmad, "A green model for sustainable software engineering," International Journal of Software Engineering and Its Applications, vol. 7, no. 4, pp. 55-74, 2013.

[71] M. Mahaux and C. Canon, "Integrating the complexity of sustainability in requirements engineering," in First international workshop on Requirements for Sustainable Systems, 2012.

[72] O. Shmueli and B. Ronen, "Excessive Software Development: Practices and Penalties," International Journal of Project Management, vol. 35, no. 1, pp. 13-27, Jan. 2017.

[73] R. K. Yin. Case Study Research: Design and Methods, 3rd edition. SAGE Publications, 2003.

[74] R. Karlsson and C. Luttropp, "EcoDesign: whats happening? An overview of the subject area of EcoDesign and of the papers in this special issue", Journal of Cleaner Production, vol. 14, no. 1516, pp. 12911298, janv, 2006.

[75] K. S. Vallerio, L. Zhong, and N. K. Jha, "Energy-efficient graphical user interface design," IEEE Transactions on Mobile Computing, vol. 5, no. 7, pp. 846-859, 2006.

[76] K. Eder and J. P. Gallagher, "Energy-Aware Software Engineering," in ICT - Energy Concepts for Energy Efficiency and Sustainability, G. Fagas, L. Gammaitoni, J. P. Gallagher, and D. J. Paul, Eds. InTech, 2017.

[77] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," Journal of Network and Computer Applications, vol. 78, pp. 97-115, Jan. 2017.

[78] C. Sahin et al., "Initial explorations on design pattern energy usage," in Green and Sustainable Software (GREENS), 2012 First International Workshop on, 2012, pp. 55-61.

[79] P. Avgeriou, M. Stal, and R. Hilliard, "Architecture Sustainability [Guest editors introduction]," IEEE Software, vol. 30, no. 6, pp. 40-44, 2013.

[80] N. van Nes and J. Cramer, "Product lifetime optimization: a challenging strategy towards more sustainable consumption patterns," Journal of Cleaner Production, vol. 14, no. 15-16, pp. 1307-1318, Jan. 2006.

[81] S. Pasricha, B. K. Donohoo, and C. Ohlsen, "A middleware framework for application-aware and user-specific energy optimization in smart mobile devices," Pervasive and Mobile Computing, vol. 20, pp. 47-63, Jul. 2015.

[82] F. Jiang, E. Zarepour, M. Hassan, A. Seneviratne, and P. Mohapatra, "Type, Talk, or Swype: Characterizing and comparing energy consumption of mobile input modalities," Pervasive and Mobile Computing, vol. 26, pp. 57-70, Feb. 2016.

[83] Colin Atkinson and Thomas Schulze, "Towards Application-Specific Impact Specifications and GreenSLAs," 2013.

[84] A. Hindle, "Green mining: investigating power consumption across versions," in Software Engineering (ICSE), 2012 34th International Conference on, 2012, pp. 1301-1304.

[85] F. A. Ali, P. Simoens, T. Verbelen, P. Demeester, and B. Dhoedt, "Mobile device power models for energy efficient dynamic offloading at runtime," Journal of Systems and Software, vol. 113, pp. 173-187, Mar. 2016.

[86] G. Procaccianti, H. Fernndez, and P. Lago, "Empirical evaluation of two best practices for energy-efficient software development," Journal of Systems and Software, vol. 117, pp. 185-198, Jul. 2016.

[87] M. Dick, E. Kern, J. Drangmeister, S. Naumann, and Johann, Timo, "Measurement and Rating of Software Induced Energy Consumption of Desktop PCs and Servers," presented at the EnviroInfo 2011, 2011.

[88] X. Li and J. P. Gallagher. A source-level energy optimization framework for mobile applications. In G. Bavota and M. Greiler, editors, 16th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2016), pages 31-40. IEEE Computer Society, 2016

[89] T. Johann, M. Dick, S. Naumann, and E. Kern, "How to measure energy-efficiency of software: Metrics and measurement results," in Proceedings of the First International Workshop on Green and Sustainable Software, 2012, pp. 51-54.

[90] M. Valluri and L. K. John, "Is Compiling for Performance-Compiling for Power?," in Interaction between compilers and computer architectures, Springer, 2001, pp. 101-115.

[91] J. Pallister, "Exploring the fundamental differences between compiler optimisations for energy and for performance.," University of Bristol, UK, 2016.

[92] J. Singh, K. Naik, and V. Mahinthan, "Impact of Developer Choices on Energy Consumption of Software on Servers," Procedia Computer Science, vol. 62, pp. 385-394, 2015.

[93] S. Murugesan, "Harnessing Green IT: Principles and Practices," in IT Professional, vol. 10, no. 1, pp. 24-33, Jan.-Feb. 2008. doi: 10.1109/MITP.2008.10

[94] N. Amsel, Z. Ibrahim, A. Malik and B. Tomlinson, "Toward sustainable software engineering: NIER track", 2011 IEEE 33rd International Conference on Software Engineering (ICSE), (2011), pp. 976-979

[95] M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999

[96] G. Pinto, F. Soares-Neto, and F. Castor, "Refactoring for Energy Efficiency: A Reflection on the State of the Art," 2015, pp. 29-35.

[97] J. Taina, "How green is your software?," ICSOB, vol. 51, pp. 151-162, 2010.

[98] V. Bauer and A. Vetro, "Comparing reuse practices in two large software-producing companies," Journal of Systems and Software, vol. 117, pp. 545-582, Jul. 2016.

[99] M. Dabhade, S. Suryawanshi, and R. Manjula, "A systematic review of software using domain engineering paradigms," in Green Engineering and Technology (IC-GET), 2016 Online International Conference on, 2016, pp. 1-6.

[100] B. Penzenstadler, B. Tomlinson, and D. Richardson, "Re4es: Support environmental sustainability by requirements engineering," in International Workshop on Requirements Engineering for Sustainable Systems, 2012.

[101] G. A. García-Mireles, M. . Moraga, F. García, C. Calero, and M. Piattini, "Interactions between environmental sustainability goals and software product quality: A mapping study," Information and Software Technology, Oct. 2017.

[102] S. Bhattacharya, K. Rajamani, K. Gopinath, and M. Gupta, "The interplay of software bloat, hardware energy proportionality and system bottlenecks," in Proceedings of the 4th Workshop on Power-Aware Computing and Systems, 2011, p. 1.

[103] S. Bhattacharya, K. Gopinath, K. Rajamani, and M. Gupta, "Software bloat and wasted joules: Is modularity a hurdle to green software?," Computer, vol. 44, no. 9, pp. 97-101, 2011.

[104] S. Bhattacharya, K. Rajamani, and M. Gupta, "Power-Performance Implications of Software Runtime Bloat," p. 18.

[105] YongKang Zhu, G. Magklis, M. L. Scott, Chen Ding, and D. H. Albonesi, "The energy impact of aggressive loop fusion," 2004, pp. 153-164.

[106] I. Stirb, "An implementation of loop fusion for improving performance and energy consumption of shared-memory parallel codes," 2017, pp. 519-526.

[107] R. Verdecchia, R. A. Saez, G. Procaccianti, and P. Lago, "Empirical Evaluation of the Energy Impact of Refactoring Code Smells," p. 10.

[108] I. Manotas, L. Pollock, and J. Clause, "SEEDS: a software engineers energy-optimization decision support framework," in Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 503-514.

[109] A. Banerjee, L. K. Chong, C. Ballabriga, and A. Roychoudhury, "Energy-Patch: Repairing Resource Leaks to Improve Energy-efficiency of Android Apps," IEEE Transactions on Software Engineering, pp. 1-1, 2017.

[110] R. Pereira et al., "Energy efficiency across programming languages: how do energy, time, and memory relate?," 2017, pp. 256-267.

[111] A. A. Bangash, H. Sahar, and M. O. Beg, "A Methodology for Relating Software Structure with Energy Consumption," 2017, pp. 111-120.

[112] H. S. Zhu, C. Lin, and Y. D. Liu, "A Programming Model for Sustainable Software," 2015, pp. 767-777.

[113] R. Pereira, T. Carcao, M. Couto, J. Cunha, J. P. Fernandes, and J. Saraiva, "Helping Programmers Improve the Energy Efficiency of Source Code," 2017, pp. 238-240.

[114] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating source line level energy information for Android applications," 2013, p. 78.

[115] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," p. 10.

[116] P. Knight and J. O. Jenkins, "Adopting and applying eco-design techniques: a practitioners perspective", Journal of Cleaner Production, vol. 17, n 5, p. 549-558, mars 2009.

[117] BSI. PD ISO/TR14062:2002, Environmental management - Integrating environmental aspects into product design and development. London: British Standards Institution; 2002.

[118] The Climate Group. "SMART 2020: Enabling the low carbon economy in the information age." The Global eSustainability Initiative, Brussels, Belgium, 2008.

[119] F. Berkhout and J. Hertin, "Impacts of Information and Communication Technologies on Environmental Sustainability: speculations and evidence," 2001.

[120] L. M. Hilty and B. Aebischer, "ICT for Sustainability: An Emerging Research Field," in ICT Innovations for Sustainability, vol. 310, L. M. Hilty and B. Aebischer, Eds. Cham: Springer International Publishing, 2015, pp. 3-36.

[121] M. Börjesson Rivera, C. Hakansson, A. Svenfelt, and G. Finnveden, "Including second order effects in environmental assessments of ICT," Environmental Modelling & Software, vol. 56, pp. 105-115, Jun. 2014.

[122] C. Okoli and K. Schabram, "A guide to conducting a systematic literature review of information systems research," 2010.

[123] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," 2014, pp. 1-10

[124] P. Bourque, R. E. Fairley, and IEEE Computer Society, Guide to the software engineering body of knowledge: Swebok. 2014.

[125] ISO/IEC 15288 "Systems engineering - System life cycle processes", International Standardization Organization/International Electrotechnical Commission, 2002, ISO,1, rue de Varembe, CH-1211 Geneve 20, Switzerland.

[126] F. Brones, M. M. de Carvalho, et E. de Senzi Zancul, "Ecodesign in project management: a missing link for the integration of sustainability in product development?", Journal of Cleaner Production, vol. 80, p. 106-118, oct. 2014.

[127] R. Chitchyan et al., "Sustainability design in requirements engineering: state of practice," 2016, pp. 533-542.