Discussion Towards a Library of Software Sustainability Requirements

Rui Carvalho
INOV, Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal

Alberto Rodrigues da Silva
INESCID, Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal

***Abstract*—This paper presents an initial approach for improving the development of sustainable software by using rigorous requirements specification. Sustainable development implies a resource management that at the same time guarantees the satisfaction of the present and future generations, usually defined in a holistic way that involves social, economic, and environmental dimensions. Software sustainability is software endurance that leads software development to accomplish results effectively for a long period. To reach software sustainability we argue that requirements specification, including the identification of well-known vulnerabilities and goals, should be considered in the process and we preliminary discuss the intent to define a library of reusability requirements for this domain.**

***Index Terms*—Requirements Specification, Software Sustainability, Sustainability, Reusability.**

I. Introduction

There is a growing concern regarding the deterioration of the environment throughout recent decades [3][18][55][27]. This concern goes hand in hand with the awareness that ecosystem services are fundamental [76]. People, communities, organizations, and countries are investing their time, money, and research efforts searching solutions to eliminate or reduce environmental threats. Sustainability shall be considered and incorporated into product design (eco-design) [26], production (supply-chain management, and technological new "green" solutions) and even social processes [36]. Some authors, as Manzini and Vezzoli [26], consider Life-Cycle-Assessment (LCA) as a fundamental tool to analyze the production cycle and identify all the possible environmental impacts. This approach implies that deep knowledge and the capability to understand scientific data [22] have become a very important tool in research. Other authors consider the use of information systems (IS) to optimize processes [11] [66]. However, these proposals do not guarantee environmental sustainability, especially when we take into account the fast negative impact of climate change [1] and the three dimensions of sustainable development [75]: social, economic, and environmental.

Software Sustainability is a good example of this reality because software design can have an important impact on the usage of resources such as energy or time. Dastbaz, Pattinson, and Akhagar [55] claim that the environmental impact of Information Technologies is enormous, stressing that the zettabyte era already uses about 50 % more energy than global aviation. Additionally, many authors [8] point that software sustainability it is not just about the minimization of energy demand but it is involved with a set of qualities, such as those specified in ISO/IEC 25010:2011 [40], with a focus on sustainability, namely involving the following qualities: functionality, efficiency, compatibility, usability, security, maintenance, portability, and reliability. Software design can allow the possibility to update or reuse, capabilities that are linked, for example, to qualities such as usability or maintenance, and are paramount [33], having a very important impact on smartphones [63].

In this research we discuss the adoption of the ITLingo RSL language [38] to rigorously define software sustainability requirements. RSL is a controlled natural language that helps the specification of requirements and tests systematically and rigorously. It allows specifying requirements through a set of constructs logically arranged into views according to multiple concerns that exist at diverse abstraction levels. RSL is the base of our work because it already supports concepts as vulnerabilities and goals/solutions and, as a consequence, our task was to build new ones with a sustainable focus. In this context, the work by Gonçalves and Silva [52] was inspirational.

This paper is structured in 8 sections. Section II presents relevant background. Section III introduces and discusses the related work that influenced our research. Sections IV and V present some examples of sustainability software vulnerabilities and goals based on the work by Imran and Kosar [2]. Section VI briefly discusses some relations that might be established between vulnerabilities and goals. Section VII discusses the current status of our research, considering the main references from the community. Lastly, Section VIII presents the conclusion and identifies open issues for future work.

II. Background

Sustainable development, especially environmental conservation, is central to the international community [48][28][31]. Issues like global warming, water, and food availability [1], and the water-energy nexus [37], when environmental threats are real, together with strong population growth, determine the UN agenda. In 1987, Oslo, at the Brundtland Report ("Our Common Future"), sustainable development is defined as "development that meets the needs of the present without compromising the ability of future generations to meet their own

needs". As a consequence UN's challenge is to help find solutions to meet present needs with a reasonable use of resources [74][75]. Boretti and Rosa [1] stress that the UN considers the usage of nature-based-solutions (NBS) to reduce the water scarcity problems that are already a reality in some regions of the world and will be even more obvious in the future, due to the increasing demand for water, reduction of water resources and increasing pollution, driven by dramatic population and economic growth. Nevertheless, they also refer that those solutions are often quite inadequate to tackle the serious problem of water scarcity. This means that technology, in purely technological solutions or together with NBS, can also be seriously considered and studied.

Alone, just like the UN, the scientific community, including the Information Technology (IT) community, is searching for new solutions and tools to help in the environmental protection, creating greener technologies [56][67][57][6][47][60][34][51][72]. Regarding software sustainability the Karlskrona manifesto for sustainability design [15] is a landmark because, it clearly states that the IT community is aware of the importance of this topic, its context, possible principles, and commitments. For example, for software practitioners, this manifesto mentions that they should try to identify the effects of their project on technical, economic, environmental sustainability. To ask questions about how to incorporate the principles into daily practice, to think about the social and individual dimensions, and talk about it with colleagues, are recommended approaches.

A first step towards software sustainability is to identify the main consequences of using IT and its potentialities to build a greener economy. Two patterns in the literature correspond to two main effects that IT has in the environment [45]. On one hand, there is an attempt to strengthen the positive impact of using IS to improve the eco-sustainability of business and society (often named "Green IS"), in which IS can promote the reuse of waste and energy and can serve as a tool for industrial symbiosis. An example of "Green IS" is a software product that helps a driver finding a solution path to a certain destiny. If the solution is accurate the driver saves gas and there is a lower carbon emission. On the contrary, if the solution is wrong there is an increase in carbon emission due to wrong behaviors [33].

On the other hand, there is an attempt to mitigate the negative environmental impact of IT production, use and disposal (often named "Green IT"). As a particular situation of "Green IT" it has emerged the term "software sustainability", when software is considered as a product that must have certain properties of usability and profitability to be economically sustainable [20]. The Software Sustainability Institute mentions that sustainable software is the one that you use today and "will be available - and continue to be improved and supported in the future" [2]. During the last decades, the "Green IT" approach becomes more important, as IT becomes more technologically complex and powerful [55]. The next section will look in more detail to this paramount approach.

III. Related Work

To create a positive impact in the context of environmental sustainability some authors look to the characteristics of software and their contribution is fundamental to our work.

Venters et al. [22] consider sustainability the capacity to endure in changing environments, as a critical concern for software architecture and practice, and that means that a software products, due to its inherent complexity, can show an increased technical debt [50][77], leading to coupling and cohesion issues, for example: unsystematic and undocumented design decisions; architectural knowledge loss; sustainability debt and the negative effects of flawed architectural design choices. This is an important perspective on software sustainability due to the impacts, both economic and environmental, of software "death". Additionally, Venters et al [22] have important works taking this point of view as their central concern. To estimate software sustainability we need software metrics, whose outputs will also help in the decision of using available resources to mitigate one limitation instead of others. For example, the Node Count and Edge Count are metrics that are related to complexity and stability, as quality attributes, and are related to maintenance, when evaluating if there is an excessive number of decisions and trace links. The number of lines of code is also considered a relevant metric for complexity [78].

Requirements specification is one of the initial steps of software engineering when the scope of the system and stakeholders' needs and concerns are iteratively elaborated [64][9][49][4]. On top of this, Requirements Engineering (RE) for Sustainability (RE4S) [13] corresponds to the usage of RE and sustainable development techniques to improve the environmental, economic and social sustainability [53] of software systems, because they influence the environment and the economy due to their direct and indirect effects. In some ways authors look to the green IT objective with care without specifying green requirements and a reaction to this reality emerges Green Requirements Engineering (GRE) as the same concept with a very specific focus on the environmental impacts of systems. Penzenstadler [11] claim that RE can be used for the improvement of the sustainability of software systems adding a few new instances of known requirements types. Fundamental to this purpose are the Guiding Questions for GRE, namely: (i) Does the system have an explicit sustainability purpose? (ii) Which impact does the system have on the environment? (iii) Is there a stakeholder for environmental sustainability? and (iv) What are the sustainability goals and constraints for the system? In question (i) if a system does not have such an explicit purpose it can be evaluated if such an aspect is desirable and feasible to add. In question (ii) the analyst evaluates direct (first-order), indirect (second-order), and systematic and potential rebound effects (third-order), hopefully with the help of systemic thinking [23].

Naumann et al. [69] put their effort into the creation of a new model, GREENSOFT, that puts together Green IT and Green IS. Their claim is a cradle-to-grave product life

cycle model tailored for software products, inspired in Life Cycle Assessment (LCA), and also sustainability metrics and criteria for software, together with extensions and guidance for sustainable design and development. They identify metrics for different phases of the software life cycle and different types of software, also considering the different effects of production and usage. Kern et al. [25] build their analysis on top of the GREENSOFT framework and present a new model by which they intend to describe the causal chains from software products to all their impacts, across all product life cycle, also on natural resources. They go even further and envision the objective of green labeling for software.

The research of these authors made possible more rigorous and structured approaches such as the one proposed by Gonçalves and Silva [52] and Silva and Sequeira [6]. These authors look to usability and cyber-security qualities, identify concerns related to its classification, and extend RSL language to include specifications for usability. If we define usability as the quality attribute that assesses how easy it is to use some product, we might accept that rigorous usability specification is possibly very relevant to software sustainability and for environmental sustainability.

Nevertheless, having this related work why is it so difficult to move forward in software sustainability? First, there are requirements conflicts; for example, between the development time and the desired quality of a software system, and economic goals and environmental ones when environmental friendly implies more expensive alternative solutions. Second, when there is an effort to achieve a sustainability purpose it is essential to mention a contribution to a higher cause explicitly instead of assigning costs, because the protection of the environment is hard to quantify with monetary or qualitative measures. Third, legislation has established environmental regulations but there is a need for legal extensions, because the existing regulations mainly ignore the very specific software qualities that are important to its sustainability. In this context, the exceptional example of safety or security laws is worth a closer look, due to its proximity to software products' reality. Fourth, when we link risk management and environmental sustainability we acknowledge that safety and security are parts of or are strongly related to sustainability [12], because, from a users' point of view, only positive outputs from the software used should be expected. At the same time, Krumdieck [68] believe that only by disruptive change and completely transforming our systems we can stop resource depletion in the long run. Quantic technology, with its higher computing capabilities, eventually together with Artificial Intelligence, can be part of this change [29]. Additionally, incorporating sustainability as an explicit objective in software systems development, together with creative confidence [73], methodological guidance and awareness are essential for software engineering for sustainability. For example, the IEEE 830 Recommendations for Software Requirements Specifications and the ISO 25000 on Software Quality, on top of ISO standard families on environmental management [41] and social responsibility [42] are landmarks towards this green purpose, because many of the suggested requirements for software sustainability are not new, they are the same or were adapted [61].

Sanchez et al. [58] point some vulnerabilities of software and stress the major importance of those related to computer security, due to the numerous cases of cybercrime. The use of software vulnerabilities' databases, such as https://nvd.nist.gov/, facilitates the sharing of information and when available, provides a value for Severity, a metric that is used to obtain the impact, when multiplied by the Presence, defined as the number of products that are affected by a vulnerability divided by the total number of products of that year. The authors created a data collection tool (https://github.com/mariocalin/nistAnalysis) and identified very important vulnerabilities in 2015, for example: buffer overflow in the Google Chrome browser versions ranging from 0.1 through 43 (which allow attackers to obtain operating system privileges); buffer overflow in the Linux Kernel, in versions 3.X and .X; a driver that is used in the Qualcomm Innovation Center (QuIC), allowing gest users to obtain operating system privileges or produce a DoS in the Linux 3.x kernel. When looking for the categories of vulnerabilities, the authors first mention the Improper Restriction of Operations within the Bounds of a Memory Buffer. Certainly, we can admit the possible importance of these vulnerabilities to the sustainability of any software.

Several authors, such as [20][22][59][14][10][59][44], focus their research on the study of software sustainability requirements. Venters et al. [20][21] try to understand the reason behind the ambiguity of software sustainability and clarify this concept. They look in detail to non-functional requirements, e.g. quality requirements, stressing some aspects such as modifiability, functional correctness, availability, interoperability, and recoverability [20][21][22][59]. Becker et al. [14] pay special attention to the strong link between software sustainability and our society, as far as human relations and quality of life are concerned.

Other authors as Paech et al. [10] try to develop a systematic process for deriving sustainability requirements for a specific system, meaning the identification of a checklist of general and IT-specific details for each sustainability dimensions (environmental, technical, social, economic and individual) and influences between them, together with a new model and new concepts as Needs and Effects (between Needs), negative, neutral or positive. This checklist can be paramount during the development of a system to respect a due balance between different dimensions, and respective stakeholders, to achieve sustainability. In the context of our research we look to these Needs as if they were named Goals.

Systematic literature reviews (SLRs) are also developed by other authors [17][24][65][2] to understand the state-of-the-art in software sustainability measures. Imran and Kosar [2] used evidence-based software engineering (EBSE) and grouped the existing research efforts in two

groups: one of the non-technical level (documentation, sustainability, manifestos, training of software engineers, funding of software projects and leadership skills of project managers to achieve sustainability) and another of the technical level (software design, coding and user experience attributes). A very noteworthy conclusion of these authors' work is that regardless of the advantages of sustainable software the research community was only able to present a limited number of approaches that contribute to sustainability and they require extra research. Nevertheless, this SLR is presented as a one-stop-service for researchers and software engineers willing to learn about this research topic.

By the time this paper was being written the Covid-19 crisis hit the world and the importance of IT and software requirements become even clearer. Smit [70] point out that "we must solve for virus and the economy" and this "starts with battling the virus", something we must admit is obvious because without a society with people enjoying freedom of movements, in modern democratic eastern societies, there cannot exist a market economy. The issues arise, as the authors claim, when software products, e.g. mobile apps, control the citizens' movements to control virus' propagation. Requisites such as privacy and quality conflict when a new software product must be produced in a very short period-of-time, to help save lives. Summing up, in such an extreme situation, we question if there can be a proper balance between sustainability productivity requisites and acceptable productivity as far as the production of new software is considered. To obtain this balance the identification of sustainable software vulnerabilities and goals is essential.

IV. Vulnerabilities (Problems)

A vulnerability is a flaw in the system under consideration [54]. We present here vulnerabilities identified based on the work by Imran and Kosar [2]. These vulnerabilities are grouped considering their distinct characteristics, namely (1) technical vulnerabilities such as (a) software design principles; (b) coding principles; (c) user experience, and (2) non-technical vulnerabilities.

The following vulnerabilities are related to *software design principles*, which shall be applied from the beginning until the end of a software development life cycle:

- **(V1) No change management process**. There is NO proper change management process and consequently there is NO effective sustainable adaptation during the product time life.
- **(V2) No requirements prioritization**. There is NO effective requirement prioritization process, which means the most important sustainability requirements might be neglected.
- **(V3) No code reuse concerns**. There is NO possibility to reuse code throughout the software development process.
- **(V4) No security concerns**. There are NO security concerns when designing and developing a software product; so, its users avoid using it regardless of its effective features and possible improvements over time.
- **(V5) No availability concerns**. The software product is NOT available for use since the beginning of its life cycle. There is NO possibility to increase users' loyalty and feedback because users cannot always experience the benefits of using the software.

The following vulnerabilities are related to *coding principles*, which are the mechanisms of structuring the software source code or restructuring older code to improve its maintainability and evolvability.

- **(V6) No refactoring concerns.** There are NO tools and features to reuse old code and recreate software, consequently refactoring activities are much more difficult if not impossible.
- **(V7) No coding standard concerns.** Because there are NO coding standards for writing software, the development team has difficulties to present results coherently.
- **(V8) No testing practices.** During the software testing process, there is NO proper checklists and testing guidelines, so further software testing is NOT easy to implement.
- **(V9) No data stewardship concerns.** There are NO mechanisms to allow easy and secure access to data and its manipulation.
- **(V10) No code smell concerns.** Due to its complexity, code smells techniques are NOT easy to implement and adopt; that means that software code problems are NOT easily identifiable.

The following two vulnerabilities are related to the *user experience*. User experience is the set of design practices that tailor the user experience to match the needs and expectations of the users concerning their use cases, hence improving their satisfaction which causes them to persist with the software for a longer time.

- **(V11) No end-users feedback channels.** There is NO communication and feedback channels between end-users and software developers, for example to report and notify about suggested improvements or bugs.
- **(V12) No optimal graphical user interface.** The graphic user interface does not allow an effective and efficient end-user interaction.

The following vulnerabilities are non-technical and are mainly related to *operational support*, including management:

- **(V13) No effective documentation to support development.** There is NO effective documentation to ensure that the software related documents (e.g., requirements docu-

ment, design document, test document, and maintenance document) provide a clear and precise communication among the technical and non-technical stakeholders. This means that there would be communication noise that would create serious limitations for the effective development and usage of the software. Non-technical aspect of software sustainability.

- **(V14) Not compliant with sustainability manifestos' principles.** There are NO concerns with sustainability manifestos' principles, the commitments and policies which must be followed by a software stakeholder to ensure sustainability. NO usage of the Karlskrona manifesto's principles, defined by Becker et al. [15], means NO sustainability concerns during development and usage. Sustainability aware users do NOT trust the software because there is NO system visibility to allow informed responsible choice. Non-technical aspect of software sustainability. (Type: *Sustainability_Operational_Support*)
- **(V15) No training concerns.** There is NO training of software engineers or end-users, respectively on the technical and use techniques. That means there is NO process of learning to understand the newer and more sustainable software development best practices.
- **(V16) No funding for sustainable software design.** There is NO funding or there is NO sum of money which is provided to software teams to build more sustainable software. Funding is essential for sustainable software in the long run (C. A. Stewart et al 2015) [14]. This funding could be through grants, crowd-sourcing, or start-up funding. Non-technical aspect of software sustainability.
- **(V17) No leadership concerns for change management.** There is no leadership or, in other words, there are NO skills through which a project manager handles change management, as well as time, cost management, or HR management [14].

To control or mitigate these vulnerabilities some possible sustainability solutions were identified and defined as requirements defined as "goals".

V. Goals (Solutions)

Also consider the work by Imran and Kosar [2] and by Ahmad et al. [65] we can capture and define the following goals for sustainable software. Indeed the following requirements are defined as general goals, but we intend to classify them according to quality properties like performance, maintainability, usability, etc. [40]. Even, these examples are discussable because they are stated in a general and fuzzy way. This means that for more effective specifications, these general goals should be further refined in more specific requirements.

Technical Goals:

- **(G1) It shall be flexible and easy to be adapted**. The software product shall have the capability of adaptation, presenting no major changes, and no need for new training for end-users, saving time and other resources. (Type: *Maintainability*)
- **(G2) Software shall be effective.** The software product shall obtain valuable and effective results. (Type: *Reliability*)
- **(G3) Software shall be efficient.** Software's tasks and transactions shall be performed in an efficient time response. (Type: *Performance*)

Constraints or Non-technical Goals:

- **(G4) It shall allow cost reduction.** From a business perspective, a sustainable software shall guarantee cost reduction and profit increase in the long term. (Type: *Organizational*)
- **(G5) It shall contribute to the scientific progress.** Software shall have an important role in research. There is a need for a strategic plan that will conduct the necessary activities like training, prototyping, and implementations with a goal for more sustainable software. (Type: *Cultural*)

VI. Vulnerabilities and Goals

Goals (defined as solutions) can help to prevent or mitigate the impact of vulnerabilities (defined as problems). Thus, goals and vulnerabilities can be crossing such as illustrated in Table I. Possible links between goals (solutions) and vulnerabilities (problems) can be the following: (1) the goal *It shall be flexible and easy to be adapted* can imply the elimination of vulnerabilities such as those in the groups of software design principles and coding principles; (2) the goal *Software shall be effective* can imply to eliminate mainly technical vulnerabilities; (3) the goal *Software shall be efficient* can imply the elimination of several technical vulnerabilities, mainly related to software design principles; (4) the goal *It shall allow costs reduction* can imply the elimination of technical and some non-technical vulnerabilities and, finally, (5) the goal *It shall contribute to the scientific progress* might imply eliminating those vulnerabilities in groups of software design principles and coding principles, together with user experience. Table I shows these links.

These links between goals and vulnerabilities suggest that to reach a goal we need to mitigate certain vulnerabilities. This mitigation implies the adoption of specific actions. Again, considering the work by Imran and Kosar [2], we here discuss four groups of goals to mitigate four vulnerabilities, each one exemplifying each vulnerabilities' group.

TABLE I. LINKS BETWEEN GOALS AND VULNERABILITIES

		Goals (Solutions)				
		G1	*G2*	*G3*	*G4*	*G5*
Vulnerabilities (Problems)	*V1*	X	X	X	X	X
	V2	X	X	X	X	X
	V3	X	X	X	X	X
	V4	X	X		X	X
	V5	X	X	X	X	X
	V6	X	X		X	X
	V7	X	X		X	X
	V8	X	X		X	X
	V9	X	X		X	X
	V10	X	X		X	X
	V11		X		X	X
	V12				X	X
	V13	X	X	X		
	V14				X	
	V15				X	
	V16		X		X	
	V17	X	X			

To mitigate vulnerability ***No change management process*** (V1) to contribute to accomplishing the goal ***It shall be flexible and easy to be adapted*** (G1) it can be suggested the next group of goals:

- **(S1) Implement a Software Change Management (SCM) process**. "This process is the task of tracking and controlling changes in the software, which is part of the cross-disciplinary field of configuration management." In this context, we might consider revision control and establishment of baseline. Proper SCM will ensure that the changes are integrated in a consistent manner, which will make sure that the software continues to evolve with changing user requirements. Popular tools and repositories to support SCM process are Git tools like GitHub or Bit-Bucket (Type: *Maintainability*)

Change can be a very important variable during a software's lifetime especially if there are changes with a very high impact. To evaluate the impact of specific changes and identify the best ways to accommodate them there can be a full-time responsible person. This person might identify the need for further training by certain professionals, such as software developers. Complexity might arise in such a way that other goals should be considered at the same time and a global task of prioritizing different goals might be essential.

To mitigate vulnerability ***No refactoring concerns*** (V6) to contribute to accomplishing the goal ***It shall be flexible and easy to be adapted*** (G1) it can be suggested the next group of goals:

- **(S2) Implement effective refactoring support.** "Refactoring of software is the process of changing the object-oriented structure in such a way that it does not interfere with the external functionality of the software." [90]. (Type: *Maintainability*)

Refactoring can also be considered essential to software adaptation. If users need time and effort to learn how to use software and if major changes are needed to make it more effective and efficient it is important to make those changes invisible to them. Consequently, if a developer can change the structure of a software changing the objects' disposal, maintaining their liaisons to graphic users interface, then, this is an advantage. This implies a code analysis.

To mitigate vulnerability ***No end-users feedback channels*** (V11) to contribute to accomplishing the goal ***It shall be flexible and easy to be adapted*** (G1) it can be suggested the next group of goals:

- **(S3) Implementation of a process to facilitate User Feedback.** "This is the process of analyzing the viewpoints and experience shared by users which provides software engineers an in-depth analysis of the features which are liked by various users, bugs which users want to be fixed." [18]. The collection of information on what the users think is important for the longevity of the software is paramount and it can include conducting formal surveys and informally talking to users via conferences and meetings. The main objective is to make better quality software. (Type: *Maintainability*).

The existence of available contacts and communication channels between the firm and users and then between the design and development team is fundamental. Additionally, a proactive approach to motivate users is also essential. Users should be aware of the importance of their contribution and this is not easily tested using any automatic testing tool, the only way to estimate such awareness is to count the number of feedback messages sent by them.

To mitigate vulnerability ***No effective documentation to support development*** (V13) to contribute to accomplishing the goal ***It shall be flexible and easy to be adapted*** (G1) it can be suggested the next group of goals:

- **(S4) Implementation of a process to create and validate Effective Documentation.** "Effective documentation is the process of writing software design documents and user manual such that they are useful to engineers for understanding, coding, and maintaining the software." [4]. The user manual should be written simply and straightforwardly making it useful to the users, allowing them to fully understand the software' functionalities and use it in the most effective and efficient way. Effective documentation should also be valuable for a long time to avoid update and reprint costs and this is only possible if it includes all the relevant information and the software GUI does not change very often, regardless of any refactoring process. (Type: *Maintainability*)

From software designers and developers, documentation should include the information needed to describe the structure of the software but also the reasoning behind the definition of algorithms and GUI approach. If a new requirement is addressed and the team understands that code changes are needed then all the members of that team must know exactly where are located in the software the relevant objects or pieces of code that should be adapted to comply with that "new" requirement. Prioritization of requirements means that the team can have a chain of requirements to address each one at a time or several of them at the same time and this has different impacts on their work. Also, to document implementation errors proper descriptions should be maintained to solve any similar or equal possible errors that might exist in the future, in such a way we could save time and resources.

After looking to these groups of goals we present a simple example of the dynamics between goals and vulnerabilities. If a software product presents the vulnerability **(V7)** ***No coding standard concerns***, affecting the very abstract goal **(G1)** ***It shall be flexible and easy to be adapted***, there is a series of detailed goals we should consider. First, these detailed goals should be enumerated and then a prioritization can be determined.

A list of detailed goals might be: (1) attribute the task to a person in the team; (2) identify different standards in existing code; (3) identify each one is preferable according to the organization objectives and type of software developed; (4) exemplify, changes for each type of change needed; (5) distribute the task per team member, per module or functionality.

Examples of coding standards for open software are GNOME programming, Linux kernel coding, GNU programming standard, etc and an example, for closed software is CERT coding standard. Certainly, the culture of each organization might be essential to determine which standard will be used but, we believe, the user requirements of the software might also be fundamental to this decision. For example, if a variable, in a very large database, is always named with a convention that mentions, first, a generic name and then its usefulness, we might admit this information is more valuable to the programmer then the algorithm that might use that variable.

The complexity and usefulness of this approach will be further discussed in the next section.

VII. Discussion

As a result of our research we were able to identify goals to overcome some software vulnerabilities. These goals were defined as solutions or best practices, with a positive and abstract semantic. Considering that our object of study, software products, is constituted with a very diverse group of products, e.g., each target problem implies almost entirely a new software product type with specific characteristics, we assumed an abstract level to describe all of them. Concrete solutions would imply a very small subset of the universe of software products. At the same time, the identified vulnerabilities that correspond to a negative statement of the optimal characteristics pointed in sustainable software. This conjugations of very abstract solutions, on one hand, with very concise vulnerabilities, on the other hand, means that we face a "conflict" between the possible identification of what we need, i.e. goals, to obtain sustainable software, and the detailed steps we must take to mitigate vulnerabilities. Questions arise: (1) What do we need to have a quality and efficient software? (2) What do we need to reduce costs through software products? etc. For example, for the first question we must simplify software products and make them focused on the desired tasks they should perform. Furthermore, discussions around metrics and software development tell us that a software product with a reduced number of decisions until it finds a desirable solution is preferable to others that gives us the same result with a longer solution path, for example, a more complex algorithm.

Looking closer, if the problem and also its solution are well defined, we may admit that a software development team has to translate this knowledge into the programming of the software mechanics. This implies, we believe, that (1) rigorous requirement specification is essential and (2) it is fundamental the awareness that it is the diversity of software products and consequently vulnerabilities' diversity that justify the actual difficulty to assume major steps towards a "real" sustainable software.

Furthermore, another degree of complexity arises if we assume that hardware might greatly change the very nature of the questions around software sustainability [32][62]. A possible example might be quantum computation and parallel computation that creates the possibility to solve small parts of a problem at the same time and the end join subsolutions to find the major target problem solution. In this context, there would be a significant time saving and probably also energy savings. Could we use this technology to solve a mathematical complex problem using software agents and in the end argue that the software sustainability problem remains true because we could then move forward to even more complex problems?

The search for concrete solutions in a complex environment, as exemplified in the previous section and discussed here, means that if we test software and would like to make that test automatically, we must be aware of the existence of technical and non-technical vulnerabilities. The first ones are easier to test and the second ones can be inferred to a certain extent. Furthermore, the link between vulnerabilities and their metrics should be clear to be properly specified. This means that after some sustainability software tests are performed, we should expect meaningful outputs with clear and easily validated solutions. This might mean that tests to specific characteristics and not process realities might be simple, leading to a more sustainable reality.

VIII. Conclusion

Environmental sustainability is a major concern due to climate change and pollution that together have a major impact on our societies. Sustainable development is a holistic concept with three dimensions: social, economic, and environmental, all of them with close links, and when we look to software sustainability this remains true. In our work, we were able to identify goals to sustainability (requirements) in a way that creates possibly a first solid step towards a truly sustainable software. This remains true even if software sustainability metrics are not usually available for users and they are not aware of software sustainability. Nevertheless, our identification of very specific vulnerabilities are possibly a promising field of work considering the awareness of development teams and the clear links established with goals.

RE is a tool to accomplish this objective, but, despite this fact, there is a need to extend and improve research applications and their use [11]. In other words, in this field of engineering, sustainability continues far behind other quality concerns, such as safety and usability. Eventually, new first-order, second-order, and third-order effects of software products should be evaluated and new strategies to mitigate them should be created. The work by Naumann, Dick, Kern and Johann [69] provides a holistic vision over Green IT and Green IS and a rigorous quantification effort that might help to identify analysis gaps, and help us build new tools for software sustainability, afterward. We simplified our research avoiding working on the interactions between hardware and software, as is the reality of cyber-physical equipment such as mobile phones.

In this paper we present our first contribution to the field of sustainable software using RE, identifying and specifying vulnerabilities and goals possibly capable to obtain a new generation of more sustainable software, in such a way that even software users might experience a higher level of awareness than today. A relevant example of applicability might be the growing number of smartphones [71] and the high volume of energy consumption they create due to a growing number of mobile software products [63], that motivates us to continue our work.

Future work is the study of sustainability requirements and solutions for cyber-physical equipment [35], looking closely at energy consumption concerning software performance and structure. The development of a vulnerability scoring system similar to CVSS [30] for security but simple and adapted to sustainability, is also a possibility. Finally, another possible future work path is the creation of tests and recommendations for industry. We envision the creation of a reusable library of concerns for software sustainability, using ITLingo-Studio and the DSL named RSL (Requirements Specification Language) technologies, to specify rigorous requirements, even if starting from very abstract goals that will be further detailed in specific goals. The usage of TSL (Testing Specification Language) for Test Cases Specification is also an objective. To sum up, we would like to test software to identify possible improvements from a sustainability point of view.

References

[1] A. Boretti, and L. Rosa, “Reassessing the projections of the World Water Development Report”, Nature Partner Journals Clean Water, pp. 1-6, 2019.

[2] A. Imran and T. Kosar, “Software Sustainability: A Systematic Literature Review and Comprehensive Analysis”, Journal of Information and Software Technology, 2019

[3] A. Malhotra, N. Melville and R. Watson, “Spurring Impactful Research on Information Systems and Environmental Sustainability”, MIS Quarterly, vol. 37, pp. 1265-1274, 2013.

[4] A. Paiva, D. Maciel and A. Silva: “From Requirements to Automated Acceptance Tests with the RSL Language”, ENASE (Selected Papers), pp. 39-57, 2019.

[5] A. Silva and A. Paiva, “Towards a Test Specification Language for Information Systems: Focus on Data Entity and State Machine Tests”, MODELSWARD, 2018.

[6] A. Silva and C. Sequeira, “Towards a Library of Usability Requirements”, Proceedings of ACM SAC’2020, ACM, 2020.

[7] A. Silva, J. Caramujo, S. Monfared and P. Calado, “Improving the Specification and Analysis of Privacy Policies”, The RSLingo4Privacy Approach, ICEIS, 2016.

[8] A. Tonini, M. Marques and L. Brisolara, “Uma Avaliação da Sustentabilidade de Softwares com Foco em Aplicativos Móveis”, Conference: Simpósio Brasileiro de Qualidade de Software (Manaus), 2015.

[9] B. Nuseibeh and S. Easterbrook. “Requirements Engineering: A Roadmap”. In Proceedings of the Conference on the Future of Software Engineering, pp. 35–46, New York, NY, USA, 2000. ACM.

[10] B. Paech, A. Moreira, J. Araujo and P. Kaiser, “Towards a Systematic Process for the Elicitation of Sustainability Requirement”, 2019.

[11] B. Penzenstadler, “Infusing Green: Requirements Engineering for Green In and Through Software Systems”, Third International Workshop on Requirements Engineering for Sustainable Systems, pp. 44-53, 2014.

[12] B. Penzenstadler, A. Raturi, D. Richardson and B. Tomlinson, "Safety, Security, Now Sustainability: The Nonfunctional Requirement for the 21st Century," in IEEE Software, vol. 31, no. 3, pp. 40-47, May-June 2014..

[13] B. Penzenstadler, B. Tomlinson and D. Richardson. “Re4s: Support environmental sustainability by requirements engineering”, International Workshop on Requirements Engineering for Sustainable Systems, 2012.

[14] C. Becker et al., “Requirements: The Key to Sustainability”. IEEE Software, vol. 33, pp. 56-65, 2016.

[15] C. Becker et al., “Sustainability design and software: The Karlskrona manifesto”, in Proceedings of the 37th International Conference on Software Engineering vol. 2, ICSE’15, IEEEPress, Piscataway, NJ, USA, 2015, pp. 467–476.

[16] C. Calero and M. Piattini, editors. “Green in Software Engineering”. Springer, to be published 2014.

[17] C. Calero, M. Bertoa and M. Moroga, “A Systematic Literature Review for Software Sustainability Measures”, GREENS 2013, San Francisco, CA, USA, 2013.

[18] C. Chu, Y. Luh, T. Li and H. Chen, “Economical green product design based on simplified computer-aided product structure variation”, Computers in Industry, 60, pp. 485-500, 2009.

[19] C. Stewart, W. Barnett, E. Wernert, J. Wernert, V. Welch and R. Knepper, “Sustained software for cyberinfrastructure: Analyses of successful efforts with a focus on nsf-funded software” in: Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models, SCREAM’15,ACM,NewYork, NY, USA, 2015, pp. 63–72.

[20] C. Venters et al. (a), “Software Sustainability: The Modern Tower of Babel”, CEUR Workshop Proceedings, 2014.

[21] C. Venters et al. (b), “The Blind Men and the Elephant: Towards an Empirical Evaluation Framework for Software Sustainability” Journal of Open Research Software, vol. 2, 2014.

[22] C. Venters et al., “Software sustainability: Research and practice from a software architecture viewpoint”, The Journal of Systems and Software, vol. 138, pp. 174-188, 2018.

[23] D. Meadows, “Leverage points – places to intervene in a system”, 1999.

[24] D. Vidmar and A. Pucihar, “Systematic Literature Review: Effects of Digital Technology on Business Models and Sustainability”, Research and Practical Issues of Enterprise Information Systems, pp.12-23, December 2019.

[25] E. Kern, L. Hilty, A. Guldner, Y. Maksimov, A. Filler, J. Groger and S. Naumann, “Sustainable software products – Towards assessment criteria for resource and energy efficiency”, Future Generation Computer Systems, pp. 199-210, 2018.

[26] E. Manzini and C. Vezzoli (2011) “O Desenvolvimento de Produtos Sustentáveis. Os Requesitos Ambientais dos Produtos Industriais.”, Edusp; Ciencias Biologicas e Naturais edition, Brasil, 2011.

[27] EU, “European Green Deal”, 2019.

[28] F. Nerini, A. Slob, R. Engstrom and E. Trutnevyte, “A research and innovation agenda for zero-emission European cities”, Sustainability, Nr 11, pp. 1-13, 2019.

[29] F. Tacchino, C. Macchiavello, D. Gerace and D. Bajoni, “An artificial neuron implemented on an actual quantum processor”, Quantum Information, 2019.

[30] FIRST – Forum of Incident Response and Security Teams, Common Vulnerability Scoring System, 2018.

[31] G. Ferguson and T.Gleeson, “Vulnerability of coastal aquifers to groundwater use and climate change”, Nature Climate Change, Nr.2, 342-345, 2012.

[32] G. Reis, J. Chang, N. Vachharajani, R. Rangan and D. August, “SWIFT: Software-Implemented Hardware Fault Tolerance”, IEEE Conference Publication, pp. 1-12, 2005.

[33] H. Arndt, B. Dziubaczyk, and M. Mokosch, “Impact of Design on the Sustainability of Mobile Applications”, Information Technology in Environmental Engineering, Springer Berlin Heidelberg, 2014.

[34] H. Koziolek, “Sustainability evaluation of software architectures: a systematic review”. CM SIGSOFT Symp.-ISARCS on Quality of Software Architectures-QoSA and Architecting Critical Systems-ISARCS, QoSA’115, 2015.

[35] H. Reza, J. Straub, N. Alexander, C. Korvald, J. Hubber and A. Chawla, “Toward Requirements Engineering of Cyber-Physical Systems: Modeling CubeSat”, IEEE, pp.1-13, 2016.

[36] I. Wallimann-Helmer, L. Meyer, K. Mintz-Woo, T. Schinko and O. Serdeczny in “Loss and Damage from Climate Change Chapter 2: The Ethical Challenges in the Context of Climate Loss and Damage”, Springer Open, 2019, pp. XXII + 557.

[37] IEA, “Energy and water”, 19 February 2020.

[38] INESC-ID (2017), “ITLingo”, itbox.inesc-id.pt/ITLingo

[39] Institute for Software Research, Software Engineering for Sustainability (SE4S), University of California, Accessed 19 February 2020.

[40] ISO IEC (25010:2011): Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE)

[41] ISO, “ISO 14000 Environmental Management”, 2004.

[42] ISO, “ISO 26000 Guidance on social responsibility”, 2010.

[43] ITLingo - Specification Languages for the IT domain (2018), https://www.researchgate.net/project/ITLingo-Specification-Languages-for-the-IT-domain. Accessed May 2020

[44] J. Caramujo, A. Silva, S. Monfared, A. Ribeiro, P. Calado and T. Breaux, “RSL-IL4Privacy: a domain-specific language for the rigorous specification of privacy policies”, Requirements Engineering, vol. 24, pp. 1-26, 2019.

[45] J. Dedrick, "Green IS: Concepts and Issues for Information Systems Research," Communications of the Association for Information Systems, vol. 27, 11, 2010.

[46] J. Huber, D. Jung, E. Schaule and C. Weinhardt, “Goal Framing in Smart Charging - Increasing BEV users' charging flexibility with Digital Nudges”, 27th European Conference on Information Systems (ECIS), At Stockholm & Uppsala, Sweden, June 2019.

[47] J. Kasurinen,M. Palacin-Silva and E. Vahala, “What concerns game developers?: A study on game development processes, sustainability and metrics”. In: Proceedings of the 8th Workshop on Emerging Trends in Software Metrics, WET-SoM’17, 2017, pp. 15-21.

[48] J. Mattheus and F. Boltz, “The Shifting Boundaries of Sustainability Science: Are We Doomed Yet?”, PLoS Biology, vol. 10, Issue 6, 2012.

[49] J. Seixas, A. Ribeiro, A. Silva, “A Model-Driven Approach for Developing Responsive Web Apps”, ENASE: 257-264, 2019.

[50] J. Yli-Huumo, “The role of technical debt in software development”, The LUT Lappeenranta University of Technology, Thesis for the degree of Doctor of Science (Technology), pp. 109, 2017.

[51] K. Rannenberg, “IFIP Advances in Information and Communication Technology”, pp. 1-10, May 2015.

[52] L. Gonçalves and A. Silva, “Towards a Catalogue of Reusable Security Requirements, Risks and Vulnerabilities”, Proceedings of International Conference on Information Systems Development'2018, AIS, 2018, pp. 1 – 11.

[53] M. Al Hinai and R. Chitchyan, “Engineering Requirements for Social Sustainability”, Conference ICT for Sustainability, 2016.

[54] M. Bishop, “Vulnerabilities Analysis”, International symposium on recent advices in intrusion detection, 1999.

[55] M. Dastbaz, C. Pattinson and B. Akhagar, “Green Technology A Sustainable Approach”, MK, pp. XXV + 321, 2015.

[56] M. Goralski and T. Tan, “Artificial Intelligence and sustainable development”, The International Journal of Management Education, 18, pp. 1-9, March 2020.

[57] M. Hilty and B. Aebischer. ICT for sustainability: an emerging research field. In: ICT Innovations for Sustainability. Springer International Publishing, pp. 3–36, 2015.

[58] M. Sanchez, J. Gea, J. Alemán, J. Garcerán, and A. Toval, “Software Vulnerabilities Overview: A Descriptive Study”, vol. 25, Number 2, 2020.

[59] N. Fernandez and P. Lago, “Characterizing the contribution of quality requirements to software sustainability”, The Journal of Systems and Software, 137, 2018.

[60] N. Melville, “Information Systems Innovation for Environmental Sustainability”, 2013.

[61] P. Lago, S. Koçak, I. Crnkovic and B. Penzenstadler, “Framing sustainability as a property of software quality”. Commun, ACM vol. 58, 10, pp. 70-78, 2015.

[62] P. Ong and R. Yan, “Power-conscious software design-a framework for modeling software on hardware”, Proceedings of 1994 IEEE Symposium on Low Power Electronics, San Diego, CA, USA, pp. 36-37, 1994.

[63] P. Tarasewich, "Designing mobile commerce applications", Communications of the ACM, vol. 46, 12, pp. 57-60, 2003.

[64] P. Zave, “Classification of research efforts in requirements engineering”. ACM Computing Surveys, vol. 29, 4, pp. 315-321, 1997.

[65] R. Ahmad, A. Hussain, and F. Baharom, “Software Sustainability Characteristic for Software Development towards Long Living Software”, WSEAS TRANSACTIONS on BUSINESS and ECONOMICS, vol. 15, 2018.

[66] R. Gholami, A. Sulaiman, T. Ramayah and A. Molla, “Senior managers’ perception on green information systems (IS) adoption and environmental performance: Results from field survey”, Information & Management, 50, pp. 431-438, 2013.

[67] S. Hettrick. “Research Software Sustainability: Report on a Knowledge Exchange Workshop”, 2016.

[68] S. Krumdieck, “The Survival Spectrum: The key to Transition Engineering of complex systems” In Proceedings of the ASME International Mechanical Engineering Congress & Exposition, 2011.

[69] 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, pp. 294-304, 2011.

[70] S. Smit, M. Hirt, K. Buehler, S. Lund, E. Greenberg and A. Govindarajan, “Safeguarding our lives and our livelihoods: The imperative of our time”, McKinsey & Company, 2020.

[71] STATISTA, “Number of smartphone users worldwide”, 2020.

[72] T. Hey, S. Tansley and K. Tolle, “The Fourth Paradigm: Data Intensive Scientific Discovery”. Microsoft Research Redmond, 2009.

[73] T. Kelley and D. Kelley (2013), “Creative Confidence: Unleashing the Creative Potential Within Us All”. Crown Business.

[74] United Nations, “Artificial Intelligence for Sustainable Development: Synthesis Report”, Mobile Learning Week 2019.

[75] United Nations, “Transforming our World: The 2030 Agenda for Sustainable Development”, pp. 1-41, 25 to 27 September 2019.

[76] USDA, “More About Ecosystem Services”, 7 October 2016.

[77] W. Behutiye, P. Rodríguez, M. Oivo and A. Tosun, “Analyzing the concept of technical debt in the context of agile software development: A systematic literature review”, Information and Software Technology, vol. 82, pp. 139-158, February 2017.

[78] Y. Shin and L. Williams, “An initial study on the use of execution complexity metrics as indicators of software vulnerabilities”, SESS’11, Waikiki, Honolulu, HI, USA, 2011.