Building Up Green Software Life Cycle Model

Lauri Kivimäki*, Laura Partanen†, Jari Porras†, Kimmo Tarkkanen‡,
Anne-Marie Tuikka‡, Jari-Matti Mäkelä*, Tuomas Mäkilä*
* *University of Turku*
Turku, Finland
Emails: {lvkivi,jmjmak,tusuma}@utu.fi
† *LUT University*
Lahti & Lappeenranta, Finland
Emails: {laura.partanen,jari.porras}@lut.fi
‡ *Turku University of Applied Sciences*
Turku, Finland
Emails: {kimmo.tarkkanen,anne-marie.tuikka}@turkuamk.fi

***Abstract*—With the increased focus and importance placed on the twin green and digital transition, also the development of digital products and software systems must take environmental sustainability into account. At present, the environmental impact of software systems has gained more and more attention from both the software development and the software procurement side. Researchers have provided many conceptual models of green software, but there is a lack of comprehensive, contemporary models to cover the overall picture, the whole life cycle of software. Aim of this study is to present the Green Software Life Cycle (GSLC) model that incorporates sustainable practices and metrics. The model is formed by gathering promising practices and metrics from the existing body of knowledge and mapping them to the software life cycle phases. The theoretical model is then validated by interviewing software practitioners. The model encompasses the whole software life cycle, and should provide inspiration for not only to the software developers, but to all stakeholders who are participating the software life cycle and aim for more environmentally sustainable digital products. Although the GSLC model provides a starting point towards making the software life cycle greener, further refinement and empirical validation is necessary in the future.**

***Index Terms*—software engineering, green software, software lifecycle, green software lifecycle model, green ICT**

I. Introduction

With sustainability concerns becoming increasingly relevant in the era of digitalization, also the role of Green ICT has come into focus. Historically, the role of software in contributing towards environmental sustainability concerns has received less attention than hardware [23], [45]. As demand for green digitalization and IT has increased and with the resource consumption of hardware being ultimately reliant on the executed software, it seems imperative that software, and alongside that processes in software engineering, is made greener.

The field of software engineering has not been restricted by hardware performance in modern times [50]. In the early days of software, computers were so limited in memory and processing power that everything down to lines of code had to be optimized. With the rapid development described by Moore's law, these restrictions fell off. For 50 years, software engineering has enjoyed exponential performance improvements, allowing software to increase massively in size and complexity [29].

In 1997, the CTO of Microsoft, Nathan Myhrvold, proposed four laws of software which essentially state that software demands will always expand to meet hardware supply and that better hardware is bought and developed because software demands it [32]. What this means is that no matter how much computers advance, it will not be enough. The only way to restrain this expansion is by focusing on the efficiency of the software, and limiting its hardware requirements.

Software is always an outcome of a software development process. One viewpoint to efficiency, and environmental sustainability in general, is that they can be seen as quality attributes of the software and as such they could be considered as a part of the software development life-cycle [35]. There's a need for a model that gives both an overview of the environmental sustainable software development life-cycle process, and concrete suggestions of methods and metrics to improve sustainability of the end product.

The research question of this study is "how can environmental sustainability aspects be incorporated into the software development life-cycle?". The research question is answered by building a software development life-cycle model that incorporates green concerns whilst suggesting metrics and criteria that can be used to work towards greenness. In addition, this model is validated through interviews of software practitioners. The goal is to build a holistic model covering all the essential phases of software life cycle independently of used methodologies (e.g. SCRUM, Kanban, DevOps). At the same time the model concentrates on addressing environmental sustainability concerns and not all facets of sustainability.

II. Related Work

Efficiency of the software can be looked at from many different perspectives. Calero & Piattini [9] use both software sustainability and green software terms to mean low power consumption and waste free software. They do not mean only the final product, i.e. production-level software system, but

also consider the software development process. Thus it is important to consider efficiency of both the product (software system) as well as the development process of the product (software engineering).

The existing research on green software engineering is multifaceted but not at all comprehensive. Since it is necessary to measure things that we want to objectively investigate, research of sustainability metrics has been a subject of some focus. Secondly, attention has also been given to theoretical models introducing relevant sustainability concerns of software and ways to address them in software engineering activities. The third main type of research focuses on optimizing the sustainability of specific aspects of software systems like the choice of programming languages or architectures.

When considering the greenness of a software product, it is important to consider the whole life cycle as shown by Taina [46] already more than decade ago. Kern et al. [33] have presented a more holistic model, GREENSOFT, that emphasizes sustainability metrics and measurements in different phases of the software product life cycle. They also show that measuring the efficiency of the code is rather challenging. In addition to the GREENSOFT model, other authors have also presented models [30] [43] where they map sustainability activities and metrics into the software life cycle phases, but most of these are already from early 2010's. Various authors have tackled the measuring challenge from different perspectives. Bozzelli et al. [26] presented in their SLR a comprehensive list of proposed metrics and classification of them. Ahmad et al. [2] compared hardware and software based approaches for energy profiling. Procaccianti et al. [40] showed that measuring software energy efficiency is rather complex and has almost chaotic behavior.

interaction design (HCI) in environmental sustainability of the software. The recent studies have focused on the practical energy efficiency of the software. Pereira et al. [38], Georgiou [14], and Abdulsalam et al. [1] have studied the energy efficiency of various programming languages. Another line of research studies the impact of algorithms to the energy efficiency of the programme, e.g Rashid et al. [41] and Meissner et al. [31]. Capra et al. [10] showed that the use of application development environments has a detrimental effect on software energy efficiency. This is partly due to the use of external libraries as shown by Pinto et al. [39].

While energy efficiency is just one characteristics, the work on the sustainable software engineering processes typically consider also other sustainability dimensions, e.g. social and economic, as shown by the work of Jagroep et al. [19], Lago et al. [27] and Karita et al. [21]. In addition to focusing on the holistic software development life cycle, studies have focused on specific parts of the process. Lots of work has been done on integrating sustainability into the requirements phase, e.g. the work of Becker et al. [4] and Condori-Fernandez & Lago [12]. Jagroep et al. [19] links the energy efficiency of the software and Venters et al. [48] the general software sustainability to the architecture design. Finally the work of Blevis [7] focuses on the role of

Sustainability debt [5], i.e. the difference between a solution and its sustainability ideal, is one option to evaluate sustainability of software. It can be used to follow the sustainability of the software in relation to changing conditions. It is especially important to evaluate in the planning phase if the software is built by choosing the right components.

III. Research Methods

The Green Software Life Cycle (GSLC) model aims at describing key methods and metrics that could be applied in each software engineering and development life cycle (SDLC) phase to make the software development more environmentally sustainable. In methodological terms, the Green Software Life Cycle model presented in this paper has been built based on 1) existing body of knowledge on the SDLC models 2) review of research literature around the environmental sustainability factors, indicators and practices in software development, and 3) interviews of software developer companies and software users who apply sustainable and environmentally responsible methods in their daily development and business activities. The model development and interviews were coordinated in the meetings and the workshops of a R&D project, which aimed at developing a green criteria for public software procurement. The approach is illustrated in Figure 1.

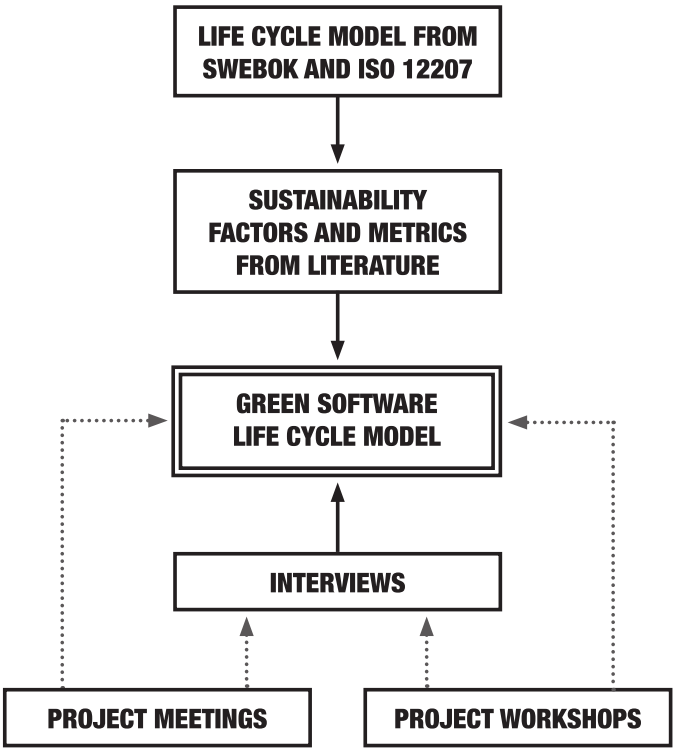

Figure 1. Approach for conducting the research and the model development.

The review of existing research literature aimed at finding the most promising methods and metrics related to sustainability of software development. Although sustainability has multiple dimensions, the literature review and the model primarily focus on the environmental and economical dimensions. The literature review was conducted by reading 164 research

publications about sustainable SDLCs, software sustainability indicators, metrics and factors, and green ICT in general. The key findings of each publication were collected into a summary table and the most promising publications for the Green Software Life Cycle model were selected in the review sessions which most of the authors participated.

We supplemented the green practices found from the academic literature with views from software development and procurement practitioners. Therefore, we performed interviews, which involved both software developers and procurement experts. The total number of interviews was 18, with 6 public and private sector procurement units and 11 software development companies (one was interviewed twice). Software development companies were SMEs as well as large companies delivering software products and services both in domestic and in international markets.

Interviews were conducted in 2022-2023 during the R&D project. Depending on the participating organization, interviews might be conducted as group interviews (n=9) or individual interviews (n=9). Altogether 15 procurement experts were interviewed. Participants from software companies included representatives of upper management (n=4), middle management (n=2), sustainability management (n=2), software development (n=4) and IT consultancy (n=4).

Each interview lasted about an hour and had several interview themes, such as a) sustainability in organizational strategy and business, b) sustainability of software products, c) sustainable software development practices, d) sustainability during the software operation phase, and e) specific organizational measures and criteria for sustainability. The focus of the interviews was mainly on the environmental sustainability. With the procurement units (n=6), the themes were the same yet focusing on the used and potential criteria and measurements in their requests-for-proposals (RFPs). First part of the software developer interviews (n=9) followed the same structure as the procurement expers interviews. Later interviews (n=3) with the software development companies had the focus on the criteria developed. The interviews were transcribed for the analysis of creating new criteria and for collecting existing practices of green software procurement.

For the study presented in this paper, the interviews serve as empirical examples of how green software development practices and metrics divide into the SDLC phases in real-life. The interview results are therefore mapped onto the SDLC phases and discussed as examples of green development practices and metrics currently in use in software development companies and user organizations.

IV. Green Software Development Life Cycle Model

The Green Software Life Cycle (GSLC) model presented in this paper has been built around the generally recognized phases of the software engineering and development life cycle (SDLC). The Software Engineering Body of Knowledge (SWEBOK) [8] and the ISO/IEC/IEEE 12207:2017 Software Life Cycle Processes standard [18] were used as the main references to define the exact phases for the underlying SDLC model. From the SWEBOK five knowledge areas defining the established SDLC phases were selected. These phases are requirements, design, construction, testing and maintenance. These traditional SDLC phases were supplemented by acquisition and disposal phases from the ISO/IEC/IEEE 12207:2017 standard to take the whole software life cycle into account. Although the phases are presented sequentially in the GSLC model, it also allows iterative or parallel execution and thus targets to be methodology agnostic.

Disposal has an integral role in such systems because most of the energy and materials can be returned to production, distribution or use phases. Hence, the disposal is added as a phase to our software life cycle model. In addition, acquisition is added at the start of our software life cycle model. This is important because sustainability does not reset for each separate software product, and is instead a continuous process.

Adopting these phases beyond the traditional SDLC scope was inspired by the concept of circular economy which aims at sustainable coexistence of the economy and the environment by adapting the idea of closed loops to different fields of business. In its ideal form, closed loops enable circular flow of materials and other resources through the whole life cycle of a product or a service without generating any waste. In practice, completely closed loop-systems are not possible, thus circular economy can be understood as dynamic economic systems "in which resource input and waste, emission, and energy leakages are minimized by cycling, extending, intensifying, and dematerializing material and energy loops" [13].

In this Section, it is described how the environmental sustainability is taken into account in each phase of the GSLC model. In addition, the GSLC model is illustrated in Figure 2. In the illustration, the key environmental sustainability factors and the most important environmental sustainability indicators and metrics identified in the literature review are mapped into the software life cycles phases. Therefore the illustration also acts as a reference guide for the existing research related to each life cycle phase.

A. Acquisition Phase

The idea of sustainability debt has been one of the guiding principles in developing this model and should guide all software life cycle phases from acquisition to disposal. Sustainability debt is essentially the difference between a proposed solution and the ideal scenario where all sustainability requirements have been met [5]. It is the hidden effect of past decisions about software-intensive systems that negatively affect economic, technical, environmental, social, and individual sustainability of the system under design. Effects in these dimensions can manifest themselves on three different levels: (1) the direct effects of the software system production and use; (2) enabling effects that arise from the ongoing use of the software system, and (3) systemic changes caused by the use of the software system on a larger scale over time. The GSLC model address mainly level 1 effects.

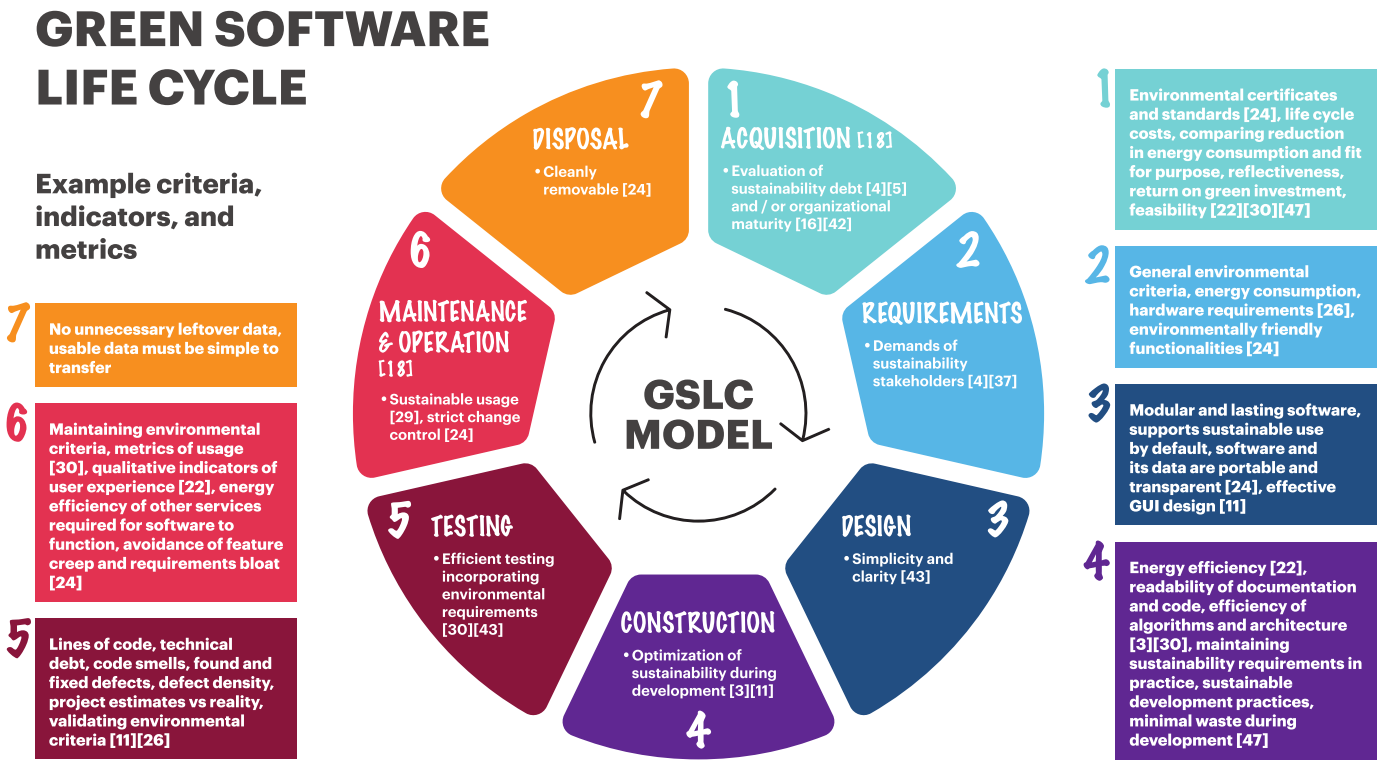

Figure 2. Illustration of Green Software Life Cycle model. The environmental sustainability factors are presented in the middle of the illustration. The factors define the key methodological approaches for achieving environmental sustainability in each life cycle phase. Detailed environmental sustainability metrics and indicators for each phase are presented in the separate boxes.

Sustainability debt can be compared to technical debt and both of them should be kept in mind when analyzing the economical feasibility of software development. When acquiring new software, sustainability debt should be used to disqualify those solutions that regress the sustainability situation, unless the need for new software is thoroughly analyzed. When preparing requests-for-proposals it should be considered unfeasible to develop software that does not follow sustainable methods.

B. Requirements Phase

The software development process properly starts at requirements engineering. As has been stated before, green digitalization necessitates that software bear responsibility for its part in energy consumption. This means that starting at the requirements phase the energy consumption and hardware requirements of software should be addressed and restricted. It is important to minimize both of these factors and ensure the maximal amount of backwards compatibility to reduce the amount of e-waste produced [24]. These requirements should be validated during later phases of the development process to ensure compliance.

A common activity during the requirements phase is stakeholder analysis, which encompasses all persons, groups and organizations that affect the project or have an interest in it [8]. To promote environmental concerns stakeholders such as customers and regulatory organs should demand sustainability concerns to be addressed [4], [37]. Environmental sustainability should also be included as its own stakeholder:e.g. being energy efficient has value in its own right, and will also save money in the long run.

Assessment or evaluation criteria for the greenness of software can also be converted into requirements for software. For example the procurer of software can demand that certain criteria are fulfilled, which then transform into requirements for the software. These can include such things as energy consumption, hardware capacity required, and functional requirements allowing sustainable use. Software should support continued development, have clear documentation and data formats and have no functionalities that are hidden from the user. [24]

C. Design Phase

After the requirements are defined, the software can be designed to fulfill its functional requirements and quality attributes. The fine-tuning of requirements will continue during the whole software development process usually iteratively. After the design process, the construction of the software should be possible.

As design is based on requirements, the primary goal of environmental sustainable design should be making the program as simple and clear as possible [43]. Simplicity is important because there should be no unnecessary work or technical debt added to the project, and clarity is important

for similar reasons. The sustainability debt [5] is also easier to manage with a simple design. Design should also support environmentally sustainable use by allowing the user to change functionalities, function in a transparent manner, give feedback on resource usage and release unnecessary hardware capacity temporarily [24].

Ideally software architecture and design should be as lean as possible. The technical architecture is based on efficiency and functionality, implementing only the required functions. Data structures, algorithms and other such considerations should be chosen based on efficiency instead of whether or not they are easy and fast to implement. While technical architecture is important, the usability should also be designed to be as fluent and efficient as possible. The design process should be done with great care, as it has a great impact on the final product and modifications to the design later in the life cycle can have an immense impact on the total cost of development.

Cloud services, data centers, the communication protocols and server architecture should also be taken into consideration. Many decisions made in the design phase are not strictly related to development of software but still greatly influence the life cycle of the software. There are many potential metrics to be used here, some of them being extensively researched and having widespread use.

D. Construction Phase

The construction phase is where the implementation begins in earnest. Optimizing environmental sustainability during development means that the contents of the written code-base are managed to be efficient and functional. Without oversight into the quality of code, problems such as unnecessary abstraction layers and poor usage of frameworks can easily appear and be hard to remove later on. These both increase maintenance costs and unnecessarily decrease the run-time efficiency of the software product [6].

In general focus on developer productivity should be lessened to increase focus on code efficiency, so that the end result would be more sustainable. Throwaway code and prototypes that cannot be reused are examples of practices that should be avoided unless absolutely necessary. Abstractions and other techniques can also unnecessarily burden the application. Refactoring the code is also a potential source of both gains and losses in sustainability. Determining whether or not refactoring will lead to sustainability gains in the long run will require more intricate tools for analysis of low-level code.

At this point it is possible to start measuring environmental factors related to the software product and its use. Some extremely important metrics are energy consumption and by extension carbon footprint, the hardware specs required by the software, and the algorithmic efficiency of the code [22], [28], [47]. Depending on the desired level of abstraction it may be necessary to determine the power consumption of different software units and elements [20], and the energy consumed by the computation and communication components [15]. This allows developers to focus on optimizing those parts of software that need it most. Energy consumption must also be related to the rate of green energy sources used, but it can be difficult to determine this information with current tools [17].

It is important to note that also the actual development practices can greatly vary in their environmental sustainability. Some important factors are to avoid hard copies of documents, business trips and contact meetings. Remote meetings with digital documents will save on energy [28]. Developers should also minimize the amount of waste during the software life cycle. Waste in this context means the amount of work that does not have discernible value to end users of the software product [22], [47]. Technical debt is one possible contributor to waste and should be minimized when possible.

E. Testing Phase

During the testing phase the most important concern traditionally is finding and fixing software defects and ensuring the software product is functional. During this phase the environmental requirements and greenness of the software should also be validated [30] [43]. Validating whether or not the criteria for standards and certificates are being upheld will lend them more credibility and ensure that software will actually have to care about being efficient. Optimizing tests for energy efficiency and sustainability is difficult, as automated testing can cost large amounts of energy while being mostly unnecessary. Still, defects must be found and fixed to avoid unnecessary energy consumption and end-user work.

When it comes to actually measuring the environmental impacts and energy consumption of a software product, it is important to note that the key metric should be energy efficiency, the ratio of energy consumed to the amount of useful work done. Using low amounts of energy does not matter if no useful work gets done, and consuming excessive amounts of energy is not sustainable. Energy efficiency can be abstracted to different levels, for example the energy efficiency of telecommunication devices can also be used as a metric. Energy efficiency has been refined further into metrics such as energy proportionality, which addresses the fact that e.g. a CPU is often at peak efficiency when it is at high utilization, but spends most of its operative time at low utilization [44].

F. Maintenance and Operation Phase

During the maintenance and operation phase of the software, it is critical that the original requirements are fulfilled as well as possible. The maintenance and operation phase is usually the longest phase of the software life cycle, and most likely its environmental impact will be highest. Moderating updates means that any updates and new functionalities that also raise the operating requirements of hardware are to be avoided. Maximizing the functional life of hardware is very important to sustainability as it reduces e-waste and, as stated earlier, software requirements are essentially what invalidates hardware that could still function. Minimizing increases in resource consumption should also be a key principle. In an ideal situation the update lowers the resource consumption of

the software product. This can often be unrealistic, so the main takeaway is to moderate updates and do careful change control.

While the software developers maintain and update the product, the users of the software also deserve some attention as software use has a great impact on its total energy efficiency. To encourage sustainable use the software should be designed to have environmentally friendly default settings and functions, and it should advise users on how to best use it in an efficient manner [28]. Enlightening users on efficient usage habits is an important part of green digitalization, and at the moment the tools for end users to improve their own footprint can be lacking.

Due to the extremely widespread use of cloud services, attention should also be directed towards factors influencing their energy efficiency. Such factors are the usage rate of a server [25], optimization of the network and data transfers, and the management of scheduling of virtual machines. One problem is that the users and even the technical administrators of a software product may have little to no chance to influence these factors. During the operational use of software, performance indicators such as simultaneous user sessions / watt [44], transactions / watt and data transferred / watt [49] can be used to analyze KPI and GPI. Energy efficiency and minimizing the electric bill will lead to both financial and sustainability gains.

G. Disposal Phase

As maintenance ends and the product faces its end of use, disposal becomes relevant. This should be a short phase in the whole software life cycle, but it is still desirable to make it as smooth as possible. In software engineering the phase of disposal is usually connected to the disposal of hardware and the recycling of electronic equipment. When considering software, this phase is usually called deactivation [22], [33].

Cradle-to-cradle thinking with the software components of the system is also necessary to ensure holistically sustainable digital products. Therefore, the authors of this paper propose to use the term disposal also when discussing the end of the software life cycle. It is important for sustainability that the data generated during the software life cycle can be transferred and used in the systems that replace it, and that the choice of new platforms, devices or products is not restricted at this point. It is important that disposal is easy for the user and no trace of the product remains beyond the data intended to be transferred for future use.

The disposal phase is linked to the first phase of acquisition with the need to take the disposal phase into account in the very early stages of the software life cycle. This also justifies the cyclic appearance of the GSLC model visualization as illustrated in Figure 2.

V. Validation of Model

To validate the GSLC model it was analyzed in relation to empirical data collected through interviews with software developers and procurement experts (as described in Section III). Due to the nature of the interviews and the small sample size, qualitative analyses of the data was seen as the best approach. Abductive coding was employed to understand if empirical data supports that criteria, indicators and metrics identified through literature are relevant for creating environmentally sustainable software and if empirical data supports the conceptualization of sustainability factors and their mapping to the phases of the GSLC model.

In the acquisition phase, interviewees from procurement units expressed the need for certifications that they could ask from the tender. Such certifications could focus on the sustainability of the organization or on the skills of the software developers in relation to green coding practices. Interviews with representatives of software companies revealed that tenders have often acknowledged the need to reduce environmental footprint. However, none of them had joined public competitive tendering that would require environmental certification. Another theme that emerged from interviewees with software developers was the careful evaluation of the need for new technological solutions.

Interviewees from software companies highlighted that customer demand is essential for developing environmentally sustainable software. Most software companies had skills and knowledge to develop sustainable software however they were not always able to implement these practices because green practices were not appreciated or financed by the customers. In the requirement phase, a procuring organization would have the opportunity to set requirements that enable and facilitate environmentally sustainable software development. In addition, it is important to recognize situations which require a lot of processing power efficient or repetitive calculations to focus optimization resources where it matters most.

In the design phase, software design should be simple for both the technical architecture and the user experience. It was brought up by interviewees from software companies that environmentally sustainable design practices are also usually software design best practices. One interviewee expressed this in the following way: “In design, the lower the hierarchy or the shorter the customer paths, the better, and at the same time it usually serves quite a few other things: accessibility, usability and this kind”.

In the construction phase, optimization of sustainability during software development was appreciated by both procurement experts and software developers. Procurement experts mentioned efficient use of algorithms and compact solutions to reduce the need for data storage. Software companies brought up the quality and usability of the code to minimize waste during software development. They emphasized the efficient use of frameworks which was perceived to have a higher impact than low-level code optimization. For example, one interviewee said: “We make a lot of user interfaces, so we’ve talked a lot about a framework like a Remix (server-side rendered apps), which basically takes web development a bit back to the good old days when we didn’t load JavaScript just in case we had to, i.e. we render on the HTML server and just serve that HTML there in the web browser and then JavaScript if there is a use for it. We try to get such a practice, that you

don't have to load all JavaScript directly into your browser every time."

The procurement experts had relatively limited experience on testing environmental sustainability requirements of software they had acquired. Some of them had experience on testing sustainability of their own hardware. However, representatives of software companies had some suggestions regarding testing sustainability requirements of software. They mentioned response and download times of web applications as good sustainability indicators. One of the interviewees mentioned that they are able to track whether their API calls transfer only the relevant data. This enables better optimization.

In the maintenance and operation phase, procurement units as well as software companies emphasized the role of cloud service providers and data center operators as the gate keepers of sustainability for production software. Environmental sustainability of the cloud service can be improved by choosing renewable energy and utilizing the heat produced by the servers for beneficial purposes. The use of cloud resources should be closely monitored and optimized to avoid any unnecessary use of these resources. In addition, software companies mentioned the importance of metrics to enable monitoring the energy efficiency of using their software during operational use.

To the most part, the empirical data supports the feasibility of the identified sustainability factors in the GSLC model. Interviewees mentioned many criteria, indicators and metrics that were also identified from literature. They also mentioned issues and practices that were not present in the model, especially the importance of cloud service providers related to the environmental sustainability of the software during the maintenance and operation phase. Although the data seems to support the model, the characteristics of the disposal phase could not be validated due to lack of interview data on the phase. The reason might be that traditionally this phase has not been widely recognized by the software practitioners. Whatever the reason is, the further research regarding this phase is clearly needed.

VI. Discussion

In this article we have studied the relationship between environmental sustainability and the software life cycle. We have been able to identify relevant sustainability factors from existing research for each phase of the software life cycle, propose the holistic Green Software Life Cycle model based on this knowledge, and proof that the model mostly resembles the actual software life cycle activities from environmental sustainability point of view. Special care should be given to phases related to the identification of need, maintenance and use. Most of the decisions affecting a software systems' environmental impact will be done in these phases. Sustainability should be addressed as early as possible in the life cycle, as it is more expensive and time-consuming to make design changes later in the life cycle of a software system. The maintenance and operation phase will in most cases be the longest part of the life cycle and the one where the brunt of the environmental impact will be caused.

In essence, to make the cradle-to-grave progression of software sustainable it is important to determine whether or not it is feasible to even start the development. If the costs incurred during development are greater than any potential gains made, the development work should be seen unfeasible. This also applies to maintenance and potential updates. The end of the life activities should also receive consideration well before the disposal phase.

In addition to describing sustainability factors, our model includes concrete criteria, indicators and metrics to be used as part of software life cycle. These have value for the practitioners who are aiming to acquire or develop sustainable software but also for the researchers who wish to explore the usability, suitability and impact of these criteria, indicators and metrics in different scenarios. We have also been able to identify the need for developing new criteria, indicators and metrics because the existing ones do not capture all the necessary aspects of each sustainability factor.

When reflecting the results of this paper to the sustainable software life cycle models [33] [30] [43] presented in the research literature, it can be said that the GSLC model is well aligned with those. Since most of these models are from early 2010's, the GSLC model supplements these with the most recent research advancements in the field. Also, the scope of these models varies compared to the GSLC model: GREENSOFT [33] is a larger, more conseptual model, where life cycle phases are only one part, whereas the model of Mahdmoud and Ahmad [30] define more detailed processes inside the life cycle phases to address sustainability requirements. The life cycle model of Shenoy and Eeratta [43] resembles the GSLC model most but concentrates on more technical issues. As a summary, the GSLC model builds up on these pioneering models while taking the latest research into account. The GSLC model also broadens the scope slightly to take all the life cycle phases from cradle to grave into consideration and thus provides support to wider number of practitioners participating the different software life cycle phases.

Lastly, it should be noted that the whole environmental impact of software is dependent not only on the software developers but also on the buyers, the administrators and the users of a system. It has been recently brought forth that the responsibility of emissions caused by software systems should be divided among all the stakeholders in order to create incentives for all the stakeholders to minimize the impacts [36]. However, this is mostly beyond the scope of this article, but worth emphasizing, that the responsibility does not rely only on the software developers' shoulders.

VII. Conclusion

In this paper we presented the theoretical Green Software Life Cycle (GSLC) model that maps methods and metrics of green software engineering into recognized software life cycle phases: acquisition, requirements [elicitation], design, construction, testing, maintenance and operation, and disposal.

The GSLC model shows that these metrics and methods of environmental sustainability vary between different phases, although the decisions made in earlier phases affect the latter. Key factors during different phases can be characterized as 1) evaluation of sustainability debt from the very first phases, 2) elicitation of requirements from the sustainability stakeholders, 3) simple and clear design, 4) optimization during construction, 5) testing of environmental requirements, 6) moderated updates during maintenance and constant monitoring of system's sustainability, and 7) clean and efficient disposal of the system. We also validated the model through interview data of software developers and software procurement experts, and showed that the factors and the metrics of the theoretical GSLC model are present in actual software life cycle phases.

The presented model connects the concepts presented in the green ICT research literature to the actual software life cycle phases. This lowers the threshold for the software developers, the software procurement experts and other software practitioners to take environmental sustainability into consideration in their work independently of used development methodologies. The model forms a starting point for further academic discussion and together with other similar models lays a foundation for the standardization work of green software life cycle practices. In the future, the created model should be further validated in different real-life software acquisition, development and operation scenarios by interviewing software practitioners, and further elaborated by mapping existing and upcoming academic research into the model.

REFERENCES

[1] Sarah Abdulsalam, Donna Lakomski, Qijun Gu, Tongdan Jin, and Ziliang Zong. Program Energy Efficiency: The Impact of Language, Compiler and Implementation Choices. In *International Green Computing Conference*, pages 1–6, 2014.

[2] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab. Hamid, Feng Xia, and Muhammad Shiraz. A Review on Mobile Application Energy Profiling: Taxonomy, State-of-the-art, and Open Research Issues. *Journal of Network and Computer Applications*, 58:42–59, 2015.

[3] Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, and Antonio Vetrò. Understanding Green Software Development: A Conceptual Framework. *IT Professional*, 17(1):44–50, 2015. Conference Name: IT Professional.

[4] Christoph Becker, Stefanie Betz, Ruzanna Chitchyan, Leticia Duboc, Steve M. Easterbrook, Birgit Penzenstadler, Norbet Seyff, and Colin C. Venters. Requirements: The Key to Sustainability. *IEEE Software*, 33(1):56–65, 2016. Conference Name: IEEE Software.

[5] Stefanie Betz, Christoph Becker, Ruzanna Chitchyan, Leticia Duboc, Steve M. Easterbrook, Birgit Penzenstadler, Norbert Seyff, and Colin C. Venters. Sustainability Debt: A Metaphor to Support Sustainability Design Decisions. In *Proceedings of the Fourth International Workshop on Requirements Engineering for Sustainable Systems, RE4SuSy 2015, co-located with the 23rd IEEE International Requirements Engineering Conference (RE 2015), Ottawa, Canada, August 24, 2015*, volume 1416 of *CEUR Workshop Proceedings*, pages 55–53. CEUR-WS.org, 2015.

[6] Suparna Bhattacharya, Karthick Rajamani, K. Gopinath, and Manish Gupta. The Interplay of Software Bloat, Hardware Energy Proportionality and System Bottlenecks. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 1–5. Association for Computing Machinery, 2011.

[7] Eli Blevis. Sustainable Interaction Design: Invention & Disposal, Renewal & Reuse. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 503–512. Association for Computing Machinery, 2007.

[8] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, 3rd edition, 2014.

[9] Coral Calero and Mario Piattini. Introduction to Green in Software Engineering. In *Green in Software Engineering*, pages 3–27. Springer International Publishing, 2015.

[10] Eugenio Capra, Chiara Francalanci, and Sandra A. Slaughter. Is Software "Green"? Application Development Environments and Energy Efficiency in Open Source Applications. *Information and Software Technology*, 54(1):60–71, 2012.

[11] Nitin Singh Chauhan and Ashutosh Saxena. A Green Software Development Life Cycle for Cloud Computing. *IT Professional*, 15(1):28–34, 2013. Conference Name: IT Professional.

[12] Nelly Condori-Fernandez and Patricia Lago. Characterizing the Contribution of Quality Requirements to Software Sustainability. *Journal of Systems and Software*, 137:289–305, 2018.

[13] Martin Geissdoerfer, Marina P. P. Pieroni, Daniela C. A. Pigosso, and Khaled Soufani. Circular Business Models: A Review. *Journal of Cleaner Production*, 277:123741, 2020.

[14] Stefanos Georgiou, Maria Kechagia, and Diomidis Spinellis. Analyzing Programming Languages' Energy Consumption: An Empirical Study. In *Proceedings of the 21st Pan-Hellenic Conference on Informatics*, PCI '17, pages 1–6. Association for Computing Machinery, 2017.

[15] Havva Gülay Gürbüz and Bedir Tekinerdogan. Software Metrics for Green Parallel Computing of Big Data Systems. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 345–348, 2016.

[16] Albert Hankel, Lisa Oud, Maiko Saan, and Patricia Lago. A Maturity Model for Green ICT: The Case of the SURF Green ICT Maturity Model. In *Proceedings of the 28th Conference on Environmental Informatics - Informatics for Environmental Protection, Sustainable Development and Risk Management*. BIS-Verlag, Oldenburg, 2014.

[17] Md Mohaimenul Hossain, Jean-Philippe Georges, Eric Rondeau, and Thierry Divoux. Energy, Carbon and Renewable Energy: Candidate Metrics for Green-Aware Routing? *Sensors*, 19(13):2901, 2019. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute.

[18] ISO Central Secretary. Systems and software engineering – Software life cycle processes. Standard ISO/IEC/IEEE 12207:2017(E), International Organization for Standardization, Geneva, CH, 2017.

[19] Erik Jagroep, Jan Martijn van der Werf, Sjaak Brinkkemper, Leen Blom, and Rob van Vliet. Extending Software Architecture Views with an Energy Consumption Perspective. *Computing*, 99(6):553–573, 2017.

[20] Erik A. Jagroep, Jan Martijn E. M. van der Werf, Ruvar Spauwen, Leen Blom, Rob van Vliet, and Sjaak Brinkkemper. An Energy Consumption Perspective on Software Architecture. In *Software Architecture*, Lecture Notes in Computer Science, pages 239–247. Springer International Publishing, 2015.

[21] Leila Karita, Brunna C. Mourão, and Ivan Machado. Software Industry Awareness on Green and Sustainable Software Engineering: A State-of-the-practice Survey. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES '19, pages 501–510. Association for Computing Machinery, 2019.

[22] Eva Kern, Markus Dick, Stefan Naumann, Achim Guldner, and Timo Johann. Green Software and Green Software Engineering–Definitions, Measurements, and Quality Aspects. In *Proceedings of the First International Conference on Information and Communication Technologies for Sustainability*, pages 87–94. Universität Zürich, 2013.

[23] Eva Kern, Markus Dick, Stefan Naumann, and Tim Hiller. Impacts of Software and its Engineering on the Carbon Footprint of ICT. *Environmental Impact Assessment Review*, 52:53–61, 2015.

[24] Eva Kern, Lorenz M. Hilty, Achim Guldner, Yuliyan V. Maksimov, Andreas Filler, Jens Gröger, and Stefan Naumann. Sustainable software products—towards assessment criteria for resource and energy efficiency. *Future Generation Computer Systems*, 86:199–210, 2018.

[25] Barbara Krumay and Roman Brandtweiner. Measuring the Environmental Impact of ICT Hardware. *International Journal of Sustainable Development and Planning*, 11(6):1064–1076, 2016.

[26] Patricia Lago, Qing Gu, and Paolo Bozzelli. *A Systematic Literature Review of Green Software Metrics*. VU Technical Report, 2014.

[27] Patricia Lago, Sedef Akinli Koçak, Ivica Crnkovic, and Birgit Penzenstadler. Framing Sustainability as a Property of Software Quality. *Commun. ACM*, 58(10):70–78, 2015.

[28] Giuseppe Lami, Luigi Buglione, and Fabrizio Fabbrini. Derivation of Green Metrics for Software. In *Software Process Improvement and Ca-*

pability Determination, Communications in Computer and Information Science, pages 13–24. Springer, 2013.

[29] Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. There's plenty of room at the Top: What will drive computer performance after Moore's law? *Science*, 368(6495):eaam9744, 2020. Publisher: American Association for the Advancement of Science.

[30] Sara S. Mahmoud and Imtiaz Ahmad. A Green Model for Sustainable Software Engineering. *International Journal of Software Engineering and Its Applications*, 7(4):55–74, 2013.

[31] Maximilian Meissner, Supriya Kamthania, Nishant Rawtani, James Bucek, Klaus-Dieter Lange, and Samuel Kounev. Experience and Guidelines for Sorting Algorithm Choices and Their Energy Efficiency. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*, ICPE '22, pages 137–144. Association for Computing Machinery, 2022.

[32] Nathan Myhrvold. The Future of Software, the Software Industry, and Windows '47. In *ACM97: The Next 50 Years of Computing*, ACM '97, page 1. Association for Computing Machinery, 1997.

[33] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The GREENSOFT Model: A Reference Model for Green and Sustainable Software and its Engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[34] Bendra Ojameruaye and Rami Bahsoon. Sustainability ArchDebts: An Economics-Driven Approach for Evaluating Sustainable Requirements. In *Software Sustainability*, pages 369–398. Springer International Publishing, 2021.

[35] Abdullateef Shola Oyedeji. Early Investigation Towards Defining and Measuring Sustainability as a Quality Attribute in Software Systems. Master's thesis, LUT University, 2016. Accepted: 2016-08-04.

[36] Laura Partanen, Antti Sipilä, and Jari Porras. Energy consumption and CO2 emissions of a Software – Who Is Responsible? In *https://ceur-ws.org/Vol-3621/poster-paper2.pdf*, volume 2023. Springer, 2023.

[37] Birgit Penzenstadler, Henning Femmer, and Debra Richardson. Who Is the Advocate? Stakeholders for Sustainability. In *Proceedings of the 2nd International Workshop on Green and Sustainable Software*, GREENS '13, pages 70–77. IEEE Press, 2013.

[38] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Ranking Programming Languages by Energy Efficiency. *Science of Computer Programming*, 205:102609, 2021.

[39] Gustavo Pinto, Francisco Soares-Neto, and Fernando Castor. Refactoring for Energy Efficiency: A Reflection on the State of the Art. In *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software*, pages 29–35, 2015.

[40] Giuseppe Procaccianti, Patricia Lago, Antonio Vetrò, Daniel Méndez Fernández, and Roel Wieringa. The Green Lab: Experimentation in Software Energy Efficiency. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 941–942, 2015. ISSN: 1558-1225.

[41] Mohammad Rashid, Luca Ardito, and Marco Torchiano. Energy Consumption Analysis of Algorithms Implementations. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–4, 2015. ISSN: 1949-3789.

[42] Nasir Rashid, Siffat Ullah Khan, Habib Ullah Khan, and Muhammad Ilyas. Green-Agile Maturity Model: An Evaluation Framework for Global Software Development Vendors. *IEEE Access*, 9:71868–71886, 2021. Conference Name: IEEE Access.

[43] Sanath. S. Shenoy and Raghavendra Eeratta. Green Software Development Model: An Approach Towards Sustainable Software Development. In *2011 Annual IEEE India Conference*, pages 1–6, 2011. ISSN: 2325-9418.

[44] Balaji Subramaniam. Metrics, Models and Methodologies for Energy-proportional Computing. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, CCGRID '14, pages 575–578. IEEE Press, 2014.

[45] Jakub Swacha. Models of Sustainable Software: A Scoping Review. *Sustainability*, 14(1):551, 2022. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[46] Juha Taina. How Green Is Your Software? In *Software Business*, Lecture Notes in Business Information Processing, pages 151–162. Springer, 2010.

[47] Juha Taina. Good, Bad, and Beautiful Software - In Search of Green Software Quality Factors. *CEPIS UPGRADE*, 2011(4):22–27, 2011.

[48] Colin C. Venters, Rafael Capilla, Stefanie Betz, Birgit Penzenstadler, Tom Crick, Steve Crouch, Elisa Yumi Nakagawa, Christoph Becker, and Carlos Carrillo. Software Sustainability: Research and Practice From a Software Architecture Viewpoint. *Journal of Systems and Software*, 138:174–188, 2018.

[49] Zane Wei and Da Qi Ren. Review of Energy Aware Big Data Computing Measurements, Benchmark Methods and Performance Analysis. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–4, 2014. ISSN: 1095-2055.

[50] Guoqing Xu, Nick Mitchell, Matthew Arnold, Atanas Rountev, and Gary Sevitsky. Software Bloat Analysis: Finding, Removing, and Preventing Performance Problems in Modern Large-scale Object-oriented Applications. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 421–426. Association for Computing Machinery, 2010.