International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Greening the Digital Frontier: A Sustainable Approach to Software Solutions

Srividhya Chandrasekaran[1], Sriram Pollachi Subburaman[2]

[1]Senior Product Manager
Email: *schandrasekaran[at]spotify.com*

[2]Senior Software Engineer
Email: *pollach[at]adobe.com*

Abstract: ***In the 21st century, Sustainable Software Engineering practices have emerged as a pivotal process, transforming the landscape of traditional software engineering. In previous eras the focus was primarily on hardware and software development, and sustainability was often overlooked. Little attention was given to the technical, economic, environmental, social, and individual dimensions of sustainability. As software continues to play a crucial role in various aspects of our lives, contemporary software development practices have yielded significant negative impacts on the economy, society, humans, and the environment. To address these challenges, the concept of Climate Conscious Software Engineering has gained prominence. The shift towards green and sustainable software development seeks to create software that not only caters to the present and future needs of users but also minimizes adverse effects on the environment and society. This paradigm is increasingly influencing Global Software Engineering (GSE) practices. This paper delves into the foundational principles of Sustainable Software Engineering as defined by the Green Software Engineering foundation. It explores the issues and challenges in climate conscious software engineering and suggests a few recommendations based on reference studies.***

Keywords: Green Software, carbon reduction, software development, sustainability, software waste

1. Introduction

In current technological landscapes, the need to address environmental concerns has given rise to the concept of green software. This is a paradigm shift in software development that prioritizes the reduction of greenhouse gas emissions. Unlike usual approaches that often aim for carbon neutrality, the essence of sustainable software lies in actively minimizing emissions throughout its life cycle [1]. At the center of this paradigm is the pursuit of carbon efficiency, a strategy that revolves around enhancing energy efficiency, fostering carbon awareness, and optimizing hardware utilization [2].

As we delve into the nuanced realm of green software, it becomes imperative to understand the characteristics that define it and the pivotal role it plays in mitigating the ecological footprint of the digital domain. By focusing beyond the conventional path of carbon neutrality, green software endeavors to redefine the benchmarks for environmental sustainability in software development.

The crux of building sustainable software lies in an organizations commitment to minimizing carbon emissions through strategic interventions. This article explores the three primary activities involved in green software's carbon efficiency: the enhancement of energy efficiency, the cultivation of carbon awareness, and the optimization of hardware utilization [2]. Through a comprehensive examination of these facets, we aim to elucidate the transformative potential of green software in fostering a more sustainable and ecologically responsible digital future.

As we navigate through the landscape of green software, our objective is to contribute to sustainable computing. By clarifying these existing principles and practices that define green software, this article seeks to provide valuable insights for researchers, developers, and policymakers alike, fostering a joint commitment to building a carbon - neutral digital ecosystem.

2. Principles of Climate Conscious Software

6 core competencies are required to define, build, and run sustainable software applications according to Microsoft's training on sustainable software [4].

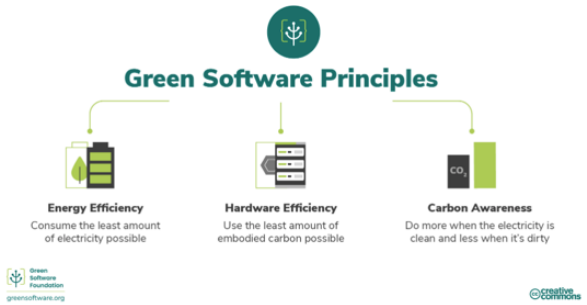

Figure 1: Green Software Principles [2] - Source: greensoftware. org

a) *Carbon efficiency*

This principle aims to minimize the amount of carbon, emitted from the most common Green House Gas. As per the United Nations Net - Zero coalition [5], carbon emissions need to be reduced by 45% by 2030 and reach net zero by 2050. For software development, this means efficient use of computational resources like reducing server load and maximizing the use of hardware resources.

b) Energy efficiency

Energy measures the amount of electricity used, and electricity is closely related to carbon emissions. This principle aims to maximize utilization rates by consolidating workload onto fewer servers with the highest utilization rates possible.

c) Carbon awareness:

This principle advocates for aligning application operation with the availability of carbon resources. In software development, this entails adjusting application runtime based on demand shifting, such as computing during periods or in regions with lower carbon intensity, which measures the amount of carbon emissions per kilowatt - hour of electricity consumed.

d) Hardware efficiency

This principle seeks to reduce the carbon emissions associated with device disposal by minimizing the amount of embedded carbon. One approach is to spread the carbon emissions over the expected lifespan of the device, thus mitigating environmental impact.

e) Measurement:

Software carbon intensity [6] quantifies the carbon emissions across various software applications. It represents the carbon equivalent emitted during specific times and locations where the software operates, measured in grams of carbon equivalent per kilowatt - hour

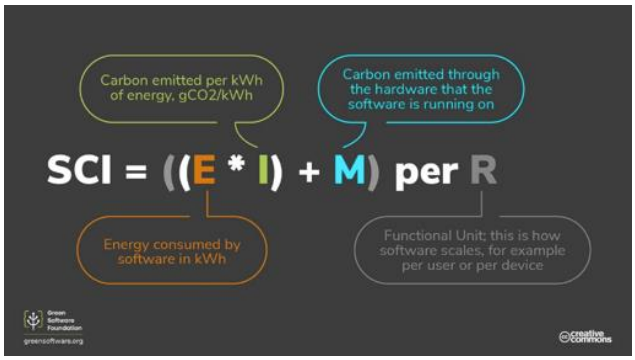

Figure 2: SCI calculation [6]

f) Climate commitments

- *Carbon Reduction:* At the highest level, carbon mitigation involves either offsets or abatement. Offsetting entails reducing emissions elsewhere, while reduction or elimination focuses on preventing carbon emissions. Offsets offer a means to complement carbon reduction efforts, with mechanisms like compensation and neutralization.
- *Carbon neutral*: The PAS 2060 standard was published by the British Standards Institution and its principles are used as the widely accepted carbon neutrality standard [7]
- *Net zero*: Net zero entails both reducing and offsetting residual emissions by employing carbon removal techniques. Mostly 90% of emissions are eliminated and the remaining 10% are permanently neutralized.

3. Process Issues and Challenges with Sustainable Software

The software development process continues to face challenges in maintaining relevance and sustainability, crucial for ensuring the quality of the end product. 'The Green Software development model' [3] introduces recommendations in each phase of software development: considering the shelf life of the software during Requirement gathering, achieving simplicity in design, using hardware resilient Application programming interface (API), Automating tests and promoting performance testing and resource profiling.

3.1 Challenges during the Software development process

Software development encompasses a broad spectrum of skills and disciplines, which include identifying user needs, values, and features essential for supporting the final product [8]. Typically, the software process consists of five primary phases: requirement specification, design, implementation, testing, and maintenance [9]. Certain activities within these phases may pose challenges, particularly in meeting contemporary demands such as reducing paper usage, minimizing e - waste generation, and managing carbon footprint and energy efficiency.

a) Requirement phase

The software requirements specification (SRS) is produced as a result of the requirement gathering and analysis process [10]. Some potential green analysis criteria consist of assessing viability, requirements, and tests [11]. Gathering requirement specifications electronically is essential to conserve resources like paper and protecting the environment. Software should also embrace new hardware technologies for improved energy efficiency and adaptability to power - down modes during operation [12]

b) Design phase

The software requirements specification (SRS) is the primary artifact during the design phase. The initial design should strike a balance to minimize the need for frequent design changes. It should not be overly extensive, but should instead encourage practices that conserve resources [13]

c) Implementation phase

During this phase, developers write source code in specific programming languages according to their preferences and the project's approach. The implementation phase focuses on programming, emphasizing the avoidance of duplicate code, custom hardware APIs, and resource - intensive APIs. Furthermore, practices such as paired programming, code reuse, and automated code generation support the minimization of energy consumption, thereby supporting energy conservation. [14]

d) Testing phase

The testing phase involves identifying and rectifying product errors until they align with the quality standards outlined in the SRS. Various types of tests, including integration and system testing, are conducted to ensure software's reliability and functionality. It is recommended to utilize automated testing and reuse test cases to assess performance scalability and resource usage [10]. Functionality and measurement could be used for the analysis of sustainable practices [8]. While functionality refers to all tests in the requirement, measurement specifies the product energy consumption [15]

e) Maintenance phase

Software maintenance occurs when issues arise that require repair or improvement. Maintenance ensures that the sustainability and quality of the product are upheld and

ensured beyond the development phase [15]. To enhance cost - effectiveness and energy efficiency, managers should offer training or courses to staff on both old and new programming languages to support the maintenance process. Environmental sustainability can be promoted during the implementation phase by ensuring that program development is clear and comprehensible to programmers. This facilitates swift internal maintenance work and contributes to improved energy efficiency, quality, and longevity of the product. [13]

3.2 Software waste

In essence, software waste refers to resources utilized without yielding any benefit, encompassing characteristics, objects, conditions, processes, and actions within project elements. In software development, wastes serve as friction and persists throughout the entire development process until the final product is produced [16]. Software waste frequently occurs due to issues such as scope ambiguity, unclear requirements, inadequate specification and design, unnecessary features, technical challenges, team conflicts, and disorganized programming throughout the development process [18]. Building Incorrect features or products, backlog mismanagement, rework, overly complex solutions, unnecessary cognitive load, psychological stress, waiting or multitasking, knowledge loss, and ineffective communication in software development also result in software waste. Enhancements to software engineering productivity involves integrating sustainability principles, policies, and practices into Extreme Programming methodologies [19]. These include fostering knowledge sharing, maintaining a positive team attitude, and prioritizing code quality. Additionally, implementing policies like team code ownership, standardized schedules, with the aim of reducing technical debt can improve productivity. Developers can adopt practices like test - driven development, continuous refactoring, pair programming, and knowledge sharing.

3.3 Green Software Process

The Green software methodology entails efficiently utilizing resources to address software needs while considering economic, social, and environmental impacts. From a software standpoint, resources refer to the natural elements essential to sustain human needs while minimizing waste. For example, cloud computing offers an alternative approach to conserve energy and physical space. Software sustainability involves extending software lifespan and minimizing waste generation during development and operation. Agarwal, Nath, and Chowdhury suggest integrating green software development cycle and sustainability criteria [12]. Building sustainable software involves using solid principles, practices, and processes that enhance the resilience of software's technical sustainability [8]. The ‘green’ factor serves as a metric for evaluating the environmental friendliness of the software development process against set standards. Its application should guarantee environmentally conscious practices in software development that are sustainable for future generations.

4. Recommendations

The authors have consolidated a curated list of recommendations based on reference studies [20]. These suggestions can be used at various stages in the software development lifecycle to build sustainable software products.

Table 1: Recommendations for best practices while building software products and applications

Functional	• Minimize hardware usage • Empower users to manage and monitor resource consumption effectively in the application • Allow users to disable unnecessary functions • Support delay tolerance and slow connectivity • Support an offline version of the software application. • Disable unsupported features on older hardware. • Prompt users to remove outdated data or suggest data removal. • Monitor feature creep
Architecture	• Design a Micro - services - based application • Analyze performance and resource usage before building native client applications • Decouple back - end and front - end • Go for a static frontend and shift processing and storage to server - side microservices. • Operate microservices as switchable function - like services to conserve resources. • Shift operations into asynchronous background processes for flexibility and resource optimization • Enable incremental over - the - air updates for efficient updates. • Provide public APIs for all core functionalities and data export • Include resource usage measurements in integration tests for actionable insights.
Development	• Eliminate unnecessary libraries; prioritize those offering essential functionality. • Reduce hardware dependencies for client - side applications • Display resource usage for individual operations to clients. • Disable staging and test environments when not in use

	• Transfer processing and storage to the server - side to mitigate client - side obsolescence. • Ensure scalability and restart capabilities. • Prioritize "oldest machine first" principle following "mobile - first" approach. • Unblock unused resources. • Minimize resource usage when no work is being done. • Prevent code and software bloat • Embrace minimalism in development practices.

References

[1] "Green Software Foundation, " *GSF*. https: //greensoftware. foundation/.
[2] "Introduction, " *Learn Green Software*. https: //learn. greensoftware. foundation/introduction/.
[3] S. S. Shenoy and Raghavendra Eeratta, "Green software development model: An approach towards sustainable software development, " Dec.2011, doi: https: //doi. org/10.1109/indcon.2011.6139638. Available: https: //ieeexplore. ieee. org/abstract/document/6139638
[4] Zimmergren, "The Principles of Sustainable Software Engineering - Training, " *Microsoft Learn*. https: //learn. microsoft. com/en - us/training/modules/sustainable - software - engineering - overview/.
[5] U. Nations, "Net Zero Coalition, " *United Nations*. https: //www.un. org/en/climatechange/net - zero - coalition.
[6] "Measurement, " *Learn Green Software*. https: //learn. greensoftware. foundation/measurement/#software - carbon - intensity - specification.
[7] "PAS 2060 The ideal standard for carbon neutrality. " Available: https: //info. eco - act. com/hubfs/0%20 - %20Downloads/PAS%202060/PAS%202060%20fact sheet%20EN. pdf
[8] T. Sedano, P. Ralph, and C. Peraire, "Software development waste, " 2017 IEEE/ACM 39th Int. Conf. Softw. Eng., pp.130–140, 2017.
[9] S. Rohana, A. Ibrahim, J. Yahaya, H. Salehudin, U. Kebangsaan Malaysia, and M. Deraman, "The Development of Green Software Process Model A Qualitative Design and Pilot Study, " IJACSA) International Journal of Advanced Computer Science and Applications, vol.12, no.8, p.2021, Available: https: //thesai. org/Downloads/Volume12No8/Paper_69 - The_Development_of_Green_Software_Process_Model. pdf
[10] J. Yahaya, K. Raisian, S. R. Ahmad Ibrahim, A. Deraman, "Green software process based on sustainability dimensions: the empirical investigation, " in Proceedings of the 1st International Conference on Informatics, Engineering, Science and Technology (INCITEST 2019), 2019.
[11] H. Acar, Software development methodology in a Green IT To cite this version : Software development methodology in a Green IT environment, Thesis, Université de Lyon, 2017.
[12] S. Agarwal, A. Nath, and D. Chowdhury, "Sustainable approaches and good practices in green software engineering, " Int. J. Res. Rev. Comput. Sci., 2012
[13] S. S. Mahmoud and I. Ahmad, "A green model for sustainable software engineering, " Int. J. Softw. Eng. its Appl., vol.7, no.4, pp.55–74, 2013.
[14] S. S. Shenoy and R. Eeratta, "Green software development model: An approach towards sustainable software development, " in Proceedings of 2011 Annual IEEE India Conference: Engineering Sustainable Solutions, INDICON - 2011, 2011.
[15] H. Acar, G. I. Alptekin, J. P. Gelas, and P. Ghodous, "Towards a green and sustainable software, " Transdisciplinary Lifecycle Analysis of Systems, 2015.
[16] M. Kramer, "Best practices in systems development lifecycle: An analyses based on the waterfall model, " Review of Business & Finance Studies, vol 9, no.1, pp.77 - 84, 2018.
[17] O. Al - Baik and J. Miller, "Waste identification and elimination in information technology organisations, " Empir. Softw. Eng., vol.19, no.6, pp.2019–2061, 2014
[18] H. Alahyari, T. Gorschek, and R. Berntsson Svensson, "An exploratory study of waste in software development organisations using agile or lean approaches: A multiple case study at 14 organisations, " Information and Software Technology, vol.105, pp.78 - 94, 2019.
[19] T. Sedano, P. Ralph, and C. Péraire, "Removing software development waste to improve productivity, " in Rethinking Productivity in Software Engineering, Berkeley, CA: Apress, pp.221–240, 2019.
[20] E. Kern et al., "Sustainable software products—Towards assessment criteria for resource and energy efficiency, " Future Generation Computer Systems, vol.86, pp.199–210, Sep.2018, doi: 10.1016/j. future.2018.02.044.