Are We Building Sustainable Software? Adoption, Challenges, and Early-Stage Strategies

Thalita Reis[a], André Araújo[b], Rodrigo Gusmão[c], Artur Farias[d], José Silva[e] and Alenilton Silva[f]

Computing Institute, Federal University of Alagoas, Av. Lourival Melo Mota, S/N, Cidade Universitária, Maceió, Brazil

Keywords: Green Software Engineering, Sustainability, Software Life Cycle, Energy Efficiency, Best Practices.

Abstract: The growing environmental impact of digital systems has brought sustainability to the forefront of software engineering research and practice. Green Software Engineering proposes a set of principles and practices aimed at reducing the energy consumption and carbon footprint of software systems. This study investigates the extent to which sustainable development practices are being adopted in the software industry and identifies the software life cycle stages in which they are applied. A structured literature review was conducted to analyze empirical evidence based on a set of design and coding practices focused on sustainability. The results reveal a fragmented and predominantly reactive adoption of these practices, with an emphasis on the development and maintenance phases. In contrast, earlier stages such as requirements elicitation and prototyping remain largely unexplored. The study also identifies key challenges, including the lack of standardization, limited real-world validation, and the absence of practical frameworks aligned with organizational processes. These findings highlight the need for comprehensive strategies and tools to support the integration of sustainability into all phases of the software development life cycle.

1 INTRODUCTION

The rising environmental impact of digital technologies has drawn global concern, especially regarding the energy use and carbon emissions of software systems (Patel et al., 2024). As software drives innovation, it increases the demand for computational resources and ecological footprints (Gupta et al., 2021). In response, Green Software Engineering (GSE) focuses on integrating sustainability into the software lifecycle (Atadoga et al., 2024). By embedding eco-conscious decisions in design, development, deployment, and maintenance, GSE aims to reduce ecological harm while preserving performance and usability (Atadoga et al., 2024).

The push for GSE adoption stems from the urgent call for sustainable digital transformation (van Gils and Weigand, 2020). Governments, stakeholders, and society are pressuring organizations to meet environmental goals, optimizing data centers, hardware, and software processes. GSE offers benefits such as reduced costs, greater energy efficiency, and stronger corporate responsibility. Increasing climate awareness reinforces the need to treat energy and carbon as key non-functional requirements.

Academic and industry initiatives have promoted GSE through frameworks, metrics, tools, and methodologies for sustainable software development (Abur, 2024) (Calero and Piattini, 2015). The Green Software Foundation, for instance, provides principles and resources to guide practice (Chandrasekaran and Subburaman, 2023). However, despite this progress, literature still lacks empirical evidence on the real-world adoption of GSE practices (Danushi et al., 2024), highlighting the need for further study.

The limited adoption of Green Software Engineering (GSE) presents key challenges. Without sustainability, software may become inefficient, consume excessive resources, and incur higher environmental costs. Development teams often lack awareness, tools, and guidelines to assess and improve energy efficiency. This absence of green practices hampers environmental goals and affects scalability, cost-

[a] https://orcid.org/0009-0002-4024-0394
[b] https://orcid.org/0000-0001-8321-2268
[c] https://orcid.org/0000-0003-1993-5044
[d] https://orcid.org/0009-0005-5576-124X
[e] https://orcid.org/0009-0001-0225-2696
[f] https://orcid.org/0009-0008-2989-3996

Reis, T., Araújo, A., Gusmão, R., Farias, A., Silva, J. and Silva, A.
Are We Building Sustainable Software? Adoption, Challenges, and Early-Stage Strategies.
DOI: 10.5220/0013656100003985

In *Proceedings of the 21st International Conference on Web Information Systems and Technologies (WEBIST 2025)*, pages 113-120
ISBN: 978-989-758-772-6; ISSN: 2184-3252

efficiency, and maintainability. Neglecting sustainability in software contributes to broader digital infrastructure issues.

This article examines the current adoption of GSE practices in the software industry. It reviews and synthesizes literature to determine whether GSE principles are being applied in practice. The central question is: Is there documented evidence in the literature of adopting Green Software Engineering practices in the software industry? The goal is to enhance understanding of sustainability in software development and guide future research and practice, identifying which practices are adopted and at what stages of the development life cycle.

The article is organized as follows: Section 2 defines key GSE principles, providing the theoretical foundation. Section 3 explains the review methodology, including study selection, analysis, and main findings. Section 4 discusses research gaps and challenges. Section 5 proposes ways to embed sustainability in requirements engineering. Section 6 concludes by summarizing contributions and suggesting directions for future work.

2 BACKGROUND

This section presents the theoretical background of this study. It first outlines the core principles of Green Software Engineering, followed by a set of best practices proposed by the Green Software Foundation, which will serve as the basis for analyzing their adoption in the software industry.

2.1 Foundations of Green Software Engineering

Green Software Engineering (GSE) integrates environmental sustainability into the development, deployment, and maintenance of software systems (Penzenstadler et al., 2014). As the demand for computing resources grows, the environmental impact of software, particularly regarding energy consumption and carbon emissions, has become a critical concern (Andrae and Edler, 2015) (Belkhir and Elmeligi, 2018). GSE addresses this challenge by embedding sustainability as a core non-functional requirement across the software lifecycle (Moreira et al., 2024).

Sustainable software is defined not just by the code but also by its interaction with hardware, networks, and cloud infrastructure (Andrikopoulos and Lago, 2021). Inefficient algorithms and poor deployment practices can result in excessive energy consumption (Maryam et al., 2018). Thus, GSE promotes a holistic approach throughout the entire software lifecycle, from requirements engineering to decommissioning (Raisian et al., 2017).

Sustainable design considers the environmental impact from the start, using lean development methods and reusable components to reduce resource waste (Ibrahim et al., 2023). Modular software facilitates updates, extends lifespan, and reduces electronic waste (Te Brinke et al., 2013). Lightweight applications and responsive design enable software to run efficiently on various devices, minimizing the need for energy-intensive hardware (Turan and Şahin, 2017).

Adopting established guidelines and best practices is essential to supporting sustainable software development. These principles form the foundation for strategies that promote responsible and environmentally conscious engineering (Matthew et al., 2024), focusing on both technical aspects like code optimization and organizational approaches that foster a culture of sustainability.

2.2 Best Practices for Green Software Development

The Green Software Foundation has proposed a set of principles to guide the development of sustainable software systems, aiming to reduce energy consumption and carbon emissions throughout the software lifecycle (Green Software Foundation, 2021). This section presents seven key practices that developers and organizations can adopt to align software engineering processes with environmental sustainability goals.

- **BP1 - Focus on high-consumption features and common usage scenarios:** Identify and optimize energy-intensive and frequently used features to maximize environmental benefits.
- **BP2 - Reduce data usage:** Minimize unnecessary data exchange by using efficient caching, compressing and aggregating data, and reducing media and image sizes.
- **BP3 - Remove or refactor unused features:** Eliminate obsolete features to improve energy efficiency, reduce execution overhead, and simplify future updates.
- **BP4 - Eliminate ineffective loops and idle computations:** Remove code that consumes energy without functional benefits, such as unnecessary attempts to connect to unreachable servers.
- **BP5 - Adapt application behavior to power modes and device conditions:** Adjust software behavior based on device energy state, e.g., reducing background activities in power-saving mode.

- **BP6 - Limit computational accuracy to operational needs:** Avoid excessive precision when unnecessary, such as approximating geolocation for locating nearby friends to save energy.
- **BP7 - Monitor real-time energy consumption:** Track energy usage during runtime to identify and optimize high-impact components.

These practices provide a framework for analyzing their adoption by the software industry throughout the development lifecycle. The Green Software Foundation's guidelines are relevant, clear, and influential in both academic and industrial contexts, making them a suitable reference to evaluate current practices and identify gaps in adopting environmentally responsible software development approaches.

3 METHODOLOGY

The methodological approach of this study was structured in three stages, as shown in Figure 1. The first stage was an exploratory study, which aimed to understand the context and select sustainable software guidelines for the research. The guidelines chosen were those proposed in (Green Software Foundation, 2021), focusing on software design and coding practices, which served as the basis for analyzing sustainable practices in the reviewed studies.

The second stage involved defining a strategy for the literature review, including selection criteria and the search string shown in Figure 1. The temporal scope covered 2013-2024 to include up-to-date research, as green software engineering is an evolving field. The search was conducted in ACM, IEEE Xplore, ScienceDirect, Scopus, and Springer, retrieving 802 articles. After applying inclusion and exclusion criteria, 87 relevant articles remained.

In the second phase of analysis, the articles were further refined based on the application of green software engineering practices in the software development life cycle or in real or simulated environments. This resulted in a sample of 18 studies, which were used for data extraction and analysis. The selected articles focused on the practical development of sustainable solutions, identifying both contributions and challenges in the field. The results are presented in the following section.

4 LITERATURE REVIEW

This section comprises three parts: state-of-the-art analysis, discussion, and identified challenges. The first presents key research and approaches aligned with the study's guiding questions. The discussion contrasts findings from selected works, highlighting thematic intersections. Finally, the challenges outline gaps and opportunities for future research.

4.1 State of the Art Analysis

This study analyzed eighteen articles on green software development, outlining practices for energy efficiency across the software life cycle. These practices focus on code analysis and optimization, architectural decisions, and tool integration, addressing the increasing demand for computational resources amid limited energy availability.

A key strategy found is the correlation between code metrics and energy consumption. Studies like (Hindle, 2015) and (Sahar et al., 2019) show that object-oriented attributes (e.g., coupling, complexity, inheritance) influence energy use. Predictive models using machine learning, as in (Beghoura et al., 2014) (Alvi et al., 2021), estimate energy consumption early, guiding eco-conscious development.

Code refactoring and managing energy debt also reduce energy usage over time. (Maia et al., 2020) and (Sehgal et al., 2022) show that addressing code smells lowers energy consumption, especially in mobile apps. The concept of energy debt quantifies the cost of poor code quality, helping prioritize sustainable maintenance.

Studies like (Procaccianti et al., 2016) and (Melfe et al., 2018) provide evidence-based guidelines for energy efficiency. The former evaluates MySQL and Apache, showing reduced energy use; the latter assesses Haskell data structures, emphasizing DRAM and compiler optimizations, both using robust metrics and benchmarks.

Tool integration into development environments supports systematic adoption of green practices. HADAS (Munoz, 2017) (Munoz et al., 2017), FEETINGS (Mancebo et al., 2021), PortAuthority (Ford and Zong, 2021), and E-Debitum (Maia et al., 2020) enable energy-aware architecture choices and consumption analysis. Their integration with IDEs streamlines eco-conscious workflows.

Architectural choices also impact energy performance. (Khomh and Abtahizadeh, 2018) and (Zhan et al., 2014) show that using microservices, load balancing, and cloud offloading affects energy efficiency. Selecting suitable patterns improves both sustainability and service quality, especially at scale.

Testing and coding practices also contribute. (Jabbarvand et al., 2016) proposes test suite minimization that maintains fault detection while reducing test

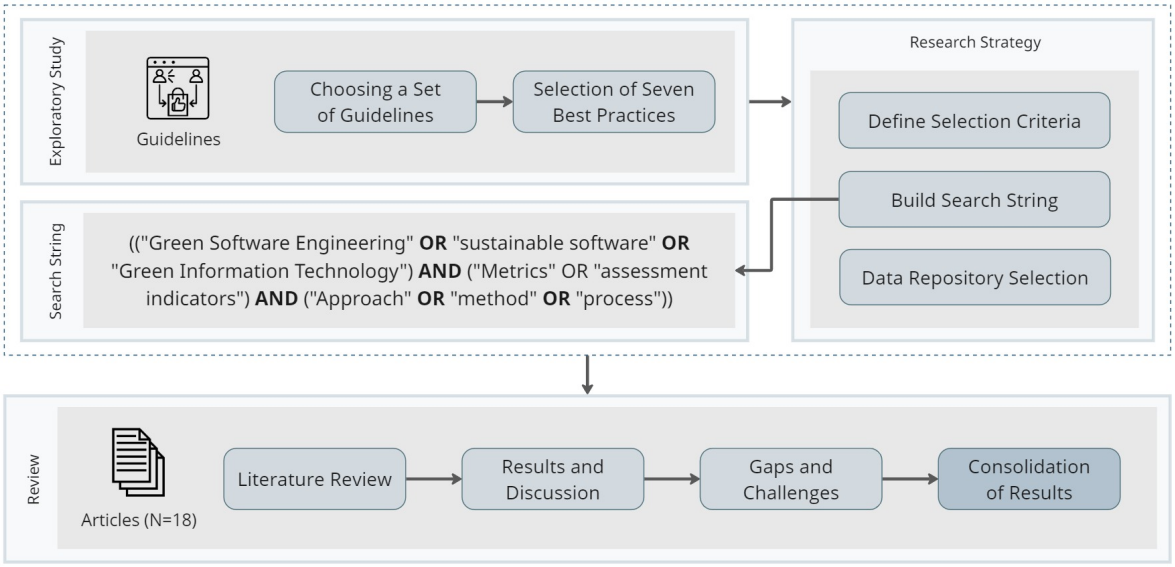

Figure 1: Methodological Approach.

runs. (Rocheteau et al., 2014) confirms empirically that Java best practices enhance energy efficiency, offering practical eco-design guidance.

Sustainability in real-world and legacy systems is feasible. (Nguyena and Chitchyan, 2013) shows that reengineering legacy systems with sustainability in mind improves performance and reduces CO2 emissions. (Oyedeji et al., 2019) validates a framework applying the Karlskrona Manifesto, aligning software with SDGs and promoting incremental, cultural integration of sustainability.

4.2 Discussion

The articles analyzed were examined using the seven core design and coding practices outlined in Section 2.2 to identify practical applications of Green Software Engineering guidelines. Figure 2 illustrates the best practices referenced. This investigation mapped not only the adoption of sustainability-oriented practices but also the challenges in implementing them across the software development lifecycle. The analysis reveals both successful green practices and barriers limiting their broader adoption in the industry.

Table 1 presents a detailed analysis of sustainable practices in software development. It organizes information on which practices are adopted and their presence across the software life cycle phases defined by SWEBOK (Washizaki, 2024). The data reveal a limited and fragmented application of the proposed guidelines. While the table includes recurring practices, such as managing energy-intensive features, refactoring unused resources, and adapting applications to usage context, few studies apply multiple practices simultaneously.

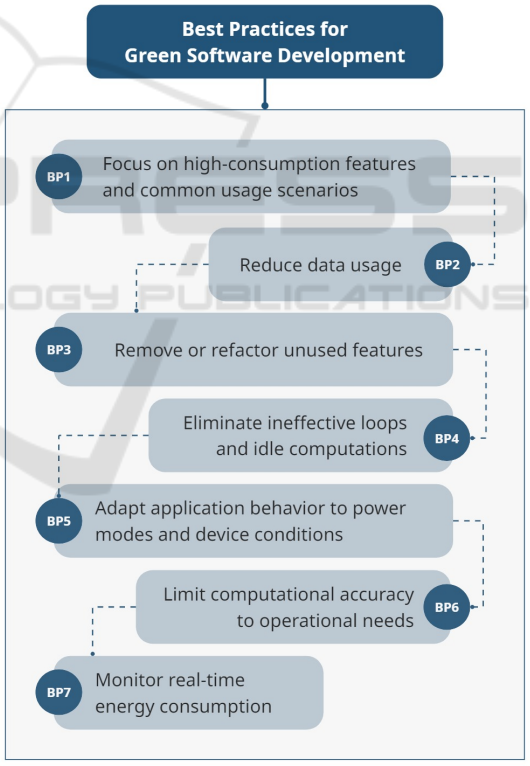

Figure 2: Best Practices for Green Software Development.

Most studies focus on improving code structure, particularly by removing unnecessary loops, refactoring, and feature control. In contrast, practices involving runtime environment integration, like real-time energy monitoring and context-aware adaptation, are

Table 1: Comparative Analysis of Green Software Engineering Practices and Lifecycle Phases.

Related Work	Best Practices							Lifecycle Phases			
	BP1	BP2	BP3	BP4	BP5	BP6	BP7	P1	P2	P3	P4
(Hindle, 2015)	●	○	○	○	○	○	○	○	○	●	●
(Sahar et al., 2019)	●	○	○	●	○	○	○	○	○	●	●
(Beghoura et al., 2014)	●	○	○	○	○	○	●	●	○	○	●
(Zhan et al., 2014)	●	○	○	○	○	○	●	○	○	●	○
(Munoz et al., 2017)	●	●	○	○	○	○	○	●	○	●	●
(Ford and Zong, 2021)	●	○	○	●	○	○	●	○	○	●	●
(Khomh and Abtahizadeh, 2018)	●	○	○	○	○	○	●	○	●	●	○
(Munoz, 2017)	●	●	○	○	○	○	○	●	●	●	●
(Mancebo et al., 2021)	●	●	○	○	○	○	●	●	●	●	●
(Alvi et al., 2021)	●	○	○	○	○	○	○	○	○	●	●
(Procaccianti et al., 2016)	●	○	●	○	●	○	●	○	○	●	○
(Melfe et al., 2018)	○	○	○	○	○	○	●	○	○	●	○
(Jabbarvand et al., 2016)	●	○	○	●	○	○	○	○	○	●	●
(Rocheteau et al., 2014)	●	○	○	○	○	○	●	○	○	●	○
(Oyedeji et al., 2019)	●	○	○	○	○	○	●	●	○	●	●
(Maia et al., 2020)	●	○	○	○	○	○	○	○	○	●	●
(Sehgal et al., 2022)	●	○	●	○	○	○	●	○	○	●	●
(Nguyena and Chitchyan, 2013)	●	○	○	○	○	○	●	●	○	●	●

Note: **P1** = Requirements elicitation and specification; **P2** = Prototyping; **P3** = Development; **P4** = Maintenance and evolution.

less common. This reflects a dominant focus on internal software elements, overlooking how software interacts with runtime conditions and hardware, which limits the impact of green practices.

Regarding life cycle coverage, sustainable practices are mostly applied during development and maintenance phases, suggesting a reactive approach. Early phases like requirements elicitation and prototyping are rarely addressed, despite their influence on later stages. This gap neglects strategic decisions that could shape sustainability from the outset.

The findings highlight a need to integrate sustainability earlier in the life cycle, including defining green requirements, energy-aware prototyping, and designing architectures with built-in sustainable principles. Additionally, the lack of standardization across studies hinders comparison, replication, and the consolidation of a solid theoretical and practical foundation.

4.3 Challenges and Gaps Identified

The analysis of the state of the art revealed gaps hindering the adoption of sustainable practices throughout the software life cycle. Many studies focus on isolated aspects without considering integrated approaches that align with software project realities. Additionally, the lack of standardized metrics makes it difficult to compare findings and replicate results. Most experiments are conducted in controlled environments, which differ from real-world usage conditions. These challenges highlight the need for further research addressing all stages of the development process, from the early phases to the creation of operational frameworks applicable within organizational contexts.

Based on this analysis, six key challenges were identified and are discussed below.

- **Limited focus on the early stages of the software life cycle:** Most studies focus on development and maintenance phases, neglecting early stages like requirements elicitation and prototyping, which are crucial for defining sustainable strategies from the start.
- **Fragmented adoption of sustainable best practices:** Studies often apply one or two best practices in isolation, without considering integrated approaches combining multiple green practices across development.
- **Limited attention to contextual and adaptive practices:** Best practices like adapting to device contexts and monitoring real-time energy consumption are rarely explored, even though they are vital for reducing energy waste in dynamic systems.
- **Lack of standardization in terminology, application, and measurement:** The absence of uni-

form definitions and evaluation methods hinders comparison and replication of studies, underscoring the need for standardized terminology and benchmarks.

- **Shortage of longitudinal studies in real usage environments:** Most studies are conducted in controlled environments, not reflecting real-world energy behavior. Longitudinal studies are needed to validate sustainable practices in realistic settings.
- **Lack of operational frameworks aligned with organizational realities:** There is a shortage of practical models to incorporate sustainability principles into software development processes, particularly in agile and DevOps environments, which hinders institutionalization.

These challenges emphasize the need for a comprehensive approach to sustainable software development. Adopting best practices consistently across all stages of the software life cycle, alongside applied research, standardized metrics, and frameworks aligned with organizational realities, will help integrate sustainability into the development process.

5 INTEGRATING SUSTAINABILITY INTO THE REQUIREMENTS PHASE

One of the main gaps identified in the literature concerns the limited attention given to incorporating green software engineering practices during the early stages of the software development life cycle. In particular, the requirements engineering phase, which is responsible for capturing and specifying stakeholder needs, is rarely explored as a space for integrating sustainability concerns. This represents a missed opportunity, as many strategic decisions directly affecting energy consumption and environmental impact are defined at this stage.

To address this gap, we propose a set of alternatives to support the inclusion of sustainability from the early stages of the software development process. These alternatives are illustrated in Figure 3.

First, it is essential to treat sustainability as a non-functional requirement (NFR) to be elicited, documented, and validated alongside performance, security, and usability. This includes requirements such as "the system shall minimize energy consumption during idle states" or "the application shall allow the user to select energy-saving modes."

Second, adopting goal-oriented requirements engineering (GORE) (Uysal, 2022) approaches can be

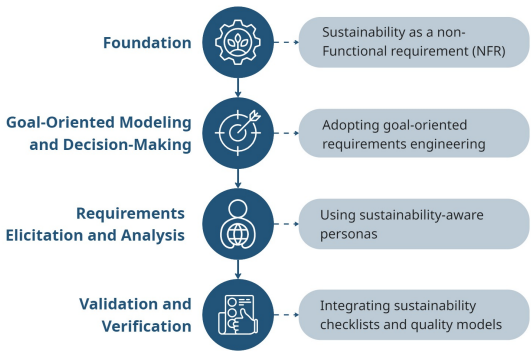

Figure 3: Strategies for Integrating Sustainability into the Requirements Engineering Process.

helpful to model sustainability goals explicitly, linking them to functional and technical decisions. Approaches such as *i** (Cabot et al., 2009) and KAOS (Zardari and Bahsoon, 2011) can support the representation of trade-offs between sustainability objectives and other system goals, promoting more informed decision-making in the early phases.

Third, using sustainability-aware personas and scenarios during requirements elicitation can help stakeholders consider the system's broader environmental impact. For example, specifying usage contexts with limited energy availability, such as mobile or embedded systems in remote areas, can help anticipate adaptation needs in the design phase.

Finally, integrating sustainability checklists and quality models, such as ISO/IEC 25010 (Qiang et al., 2024) extended with environmental attributes, into the validation process can ensure that sustainability is considered systematically. These practices can assist development teams in identifying and prioritizing green practices that are both feasible and aligned with project goals and constraints.

6 CONCLUSIONS

Green Software Engineering (GSE) has emerged in response to growing environmental concerns in the software industry. As software systems become critical across sectors, integrating sustainability into development is key for responsible digital transformation. This study explores whether sustainable practices are adopted in industry contexts and their application throughout the software life cycle.

The analysis showed that while several promising initiatives exist, sustainability practices are still limited and fragmented. Code optimization, feature control, and refactoring are more commonly applied during development and maintenance, while early

stages like requirements elicitation and prototyping are largely overlooked despite their potential impact on sustainability decisions.

This study contributes by proposing alternatives to integrate sustainability into the often-neglected requirements elicitation phase, including treating sustainability as a non-functional requirement, using goal-oriented modeling, sustainability-aware personas, and validation tools adapted to environmental attributes.

The findings also reveal challenges, such as a lack of context-aware approaches, limited standardization of metrics and terminology, and a shortage of empirical studies in real-world environments. There is also a gap in practical frameworks for incorporating sustainability into established development processes like agile and DevOps, hindering the development of a consistent knowledge base.

Future work should focus on creating adaptable frameworks and tools to support sustainability across all software development stages, alongside empirical studies, standardized metrics, and improvements in real-time energy monitoring techniques.

REFERENCES

Abur, V. (2024). The dimension of green coding in software quality control processes. In *2024 9th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6. IEEE.

Alvi, H. M., Majeed, H., Mujtaba, H., and Beg, M. O. (2021). Mlee: Method level energy estimation—a machine learning approach. *Sustainable Computing: Informatics and Systems*, 32:100594.

Andrae, A. S. and Edler, T. (2015). On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157.

Andrikopoulos, V. and Lago, P. (2021). Software sustainability in the age of everything as a service. *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future: Essays Dedicated to Michael Papazoglou on the Occasion of His 65th Birthday and His Retirement*, pages 35–47.

Atadoga, A., Umoga, U. J., Lottu, O. A., and Sodiy, E. (2024). Tools, techniques, and trends in sustainable software engineering: A critical review of current practices and future directions. *World Journal of Advanced Engineering Technology and Sciences*, 11(1):231–239.

Beghoura, M. A., Boubetra, A., and Boukerram, A. (2014). Green applications awareness: nonlinear energy consumption model for green evaluation. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 48–53. IEEE.

Belkhir, L. and Elmeligi, A. (2018). Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*, 177:448–463.

Cabot, J., Easterbrook, S., Horkoff, J., Lessard, L., Liaskos, S., and Mazón, J.-N. (2009). Integrating sustainability in decision-making processes: A modelling strategy. In *2009 31st International Conference on Software Engineering-Companion Volume*, pages 207–210. IEEE.

Calero, C. and Piattini, M. (2015). *Introduction to green in software engineering*. Springer.

Chandrasekaran, S. and Subburaman, S. P. (2023). Greening the digital frontier: A sustainable approach to software solutions. *International Journal of Science and Research (IJSR)*, 12(3):1820–1823. Fully Refereed — Open Access — Double Blind Peer Reviewed Journal.

Danushi, O., Forti, S., and Soldani, J. (2024). Environmentally sustainable software design and development: a systematic literature review. *arXiv preprint arXiv:2407.19901*.

Ford, B. W. and Zong, Z. (2021). Portauthority: Integrating energy efficiency analysis into cross-platform development cycles via dynamic program analysis. *Sustainable Computing: Informatics and Systems*, 30:100530.

Green Software Foundation (2021). 10 recommendations for green software development. Acesso em: 30 mar. 2025.

Gupta, U., Kim, Y. G., Lee, S., Tse, J., Lee, H.-H. S., Wei, G.-Y., Brooks, D., and Wu, C.-J. (2021). Chasing carbon: The elusive environmental footprint of computing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 854–867. IEEE.

Hindle, A. (2015). Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, 20:374–409.

Ibrahim, S. R. A., Sallehudin, H., and Yahaya, J. (2023). Exploring software development waste and lean approach in green perspective. In *2023 International Conference on Electrical Engineering and Informatics (ICEEI)*, pages 1–6. IEEE.

Jabbarvand, R., Sadeghi, A., Bagheri, H., and Malek, S. (2016). Energy-aware test-suite minimization for android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 425–436.

Khomh, F. and Abtahizadeh, S. A. (2018). Understanding the impact of cloud patterns on performance and energy consumption. *Journal of Systems and Software*, 141:151–170.

Maia, D., Couto, M., Saraiva, J., and Pereira, R. (2020). E-debitum: managing software energy debt. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 170–177.

Mancebo, J., Calero, C., García, F., Moraga, M. Á., and de Guzmán, I. G.-R. (2021). Feetings: framework for energy efficiency testing to improve environmental goal of the software. *Sustainable Computing: Informatics and Systems*, 30:100558.

Maryam, K., Sardaraz, M., and Tahir, M. (2018). Evolutionary algorithms in cloud computing from the perspective of energy consumption: A review. In *2018 14th international conference on emerging technologies (ICET)*, pages 1–6. IEEE.

Matthew, U. O., Asuni, O., and Fatai, L. O. (2024). Green software engineering development paradigm: An approach to a sustainable renewable energy future. In *Advancing Software Engineering Through AI, Federated Learning, and Large Language Models*, pages 281–294. IGI Global.

Melfe, G., Fonseca, A., and Fernandes, J. P. (2018). Helping developers write energy efficient haskell through a data-structure evaluation. In *Proceedings of the 6th International Workshop on Green and Sustainable Software*, pages 9–15.

Moreira, A., Lago, P., Heldal, R., Betz, S., Brooks, I., Capilla, R., Coroamă, V. C., Duboc, L., Fernandes, J. P., Leifler, O., et al. (2024). A roadmap for integrating sustainability into software engineering education. *ACM Transactions on Software Engineering and Methodology*.

Munoz, D.-J. (2017). Achieving energy efficiency using a software product line approach. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*, pages 131–138.

Munoz, D.-J., Pinto, M., and Fuentes, L. (2017). Green software development and research with the hadas toolkit. In *Proceedings of the 11th European conference on software architecture: companion proceedings*, pages 205–211.

Nguyena, N. and Chitchyan, R. (2013). Systems re-design for sustainability: Petshop case study.

Oyedeji, S., Adisa, M. O., Penzenstadler, B., and Wolf, A. (2019). Validation study of a framework for sustainable software system design and development. *sustainable development*, 3(4):5.

Patel, U. A., Patel, S., and Nanavati, J. (2024). Towards a greener future: Harnessing green computing to reduce ict carbon emissions. In *2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)*, volume 1, pages 960–969. IEEE.

Penzenstadler, B., Raturi, A., Richardson, D., and Tomlinson, B. (2014). Safety, security, now sustainability: The nonfunctional requirement for the 21st century. *IEEE software*, 31(3):40–47.

Procaccianti, G., Fernández, H., and Lago, P. (2016). Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117:185–198.

Qiang, Y., Pa, N. C., et al. (2024). Sustainable software solutions: A tool integrating life cycle analysis and iso quality models. *International Journal on Advanced Science, Engineering & Information Technology*, 14(5).

Raisian, K., Yahaya, J., and Deraman, A. (2017). Sustainable software development life cycle process model based on capability maturity model integration: A study in malaysia. *Journal of Theoretical & Applied Information Technology*, 95(21).

Rocheteau, J., Gaillard, V., and Belhaj, L. (2014). How green are java best coding practices? In *International Conference on Smart Grids and Green IT Systems*, volume 2, pages 235–246. SCITEPRESS.

Sahar, H., Bangash, A. A., and Beg, M. O. (2019). Towards energy aware object-oriented development of android applications. *Sustainable Computing: Informatics and Systems*, 21:28–46.

Sehgal, R., Mehrotra, D., Nagpal, R., and Sharma, R. (2022). Green software: Refactoring approach. *Journal of King Saud University-Computer and Information Sciences*, 34(7):4635–4643.

Te Brinke, S., Malakuti, S., Bockisch, C., Bergmans, L., and Akşit, M. (2013). A design method for modular energy-aware software. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1180–1182.

Turan, B. O. and Şahin, K. (2017). Responsive web design and comparative analysis of development frameworks. *The Turkish Online J. Des. Art Commun*, 7(1):110–121.

Uysal, M. P. (2022). Goal-oriented requirements engineering approach to energy management systems. In *International Symposium on Energy Management and Sustainability*, pages 221–230. Springer.

van Gils, B. and Weigand, H. (2020). Towards sustainable digital transformation. In *2020 IEEE 22nd conference on business informatics (CBI)*, volume 1, pages 104–113. IEEE.

Washizaki, H., editor (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0*. IEEE Computer Society.

Zardari, S. and Bahsoon, R. (2011). Cloud adoption: a goal-oriented requirements engineering approach. In *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, pages 29–35.

Zhan, K., Lung, C.-H., and Srivastava, P. (2014). A green analysis of mobile cloud computing applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 357–362.