Green software development using carbon-aware scheduling techniques and energy efficiency metrics throughout the SDLC

ARTICLE HISTORY
Received 31 July 2025
Accepted 19 November 2025
Published 6 January 2026

Mikita Piastou
University of West Georgia
School of Computing, Analytics, and Modeling
Carrollton, United States
mpiasto1@my.westga.edu
ORCID: 0009-0002-7360-2152

ISSN:1390-9266 e-ISSN:1390-9134 LAJC 2026

Green software development using carbon-aware scheduling techniques and energy efficiency metrics throughout the SDLC

Mikita Piastou
University of West Georgia
School of Computing, Analytics, and Modeling
Carrollton, United States
mpiasto1@my.westga.edu

***Abstract*—**The objective of the present paper is to systematize contemporary approaches of green software development through the prism of carbon-aware scheduling methodologies and energy efficiency metrics at all stages of the software development life cycle (SDLC). The study will analyze English-language, peer-reviewed articles published between 2020 and 2025. The following four carbon-intensive scheduling strategies have been identified: temporal task shifting, geographic load migration, electricity price consideration, and dynamic resource scaling. Experimental data indicates the potential for a 30–70% reduction in the carbon footprint of applications, with only a moderate impact on latency and cost. The metrics employed for evaluating energy efficiency span from low-level measures such as code complexity and measured power consumption to higher-level metrics addressing infrastructure and integration. It has been established that disregarding the initial phases of the SDLC results in an underestimation of the aggregate carbon footprint. The analysis showed that cutting emissions can conflict with maintaining high service quality. It also highlighted problems with standardizing metrics and ensuring accurate carbon-intensity forecasts, especially when significant task shifting is involved. Further unification of metrics, integration of energy monitoring at all stages of the SDLC, and consideration of economic factors are recommended.

Keywords*—*green development, software, application development, carbon pollution

I. Introduction

The development of software is becoming increasingly driven by the integration of environmental sustainability principles, a response to the escalating energy intensity of Information and Communication Technology (ICT) infrastructures. Today, the ICT sector accounts for roughly 2% to 4% of global carbon emissions, thus rendering green software development an urgent challenge in the fight against global warming and environmental concerns [1]. Projections indicate that ICT may account for as much as 8% of the world's energy use by 2030 [2]. The goal of green software development is to create products that minimize energy use and environmental impact at every stage of the SDLC, including design, implementation, maintenance, and eventual decommissioning. Traditionally, green computing has focused primarily on hardware.

However, in the past few years, it has become steadily clear that software architecture, algorithms, and the way systems operate also play a major role in how much energy a system uses. This study focuses on carbon-aware scheduling, which means planning computational and operational tasks with carbon intensity and energy use in mind. It also looks at ways to evaluate energy efficiency throughout the different stages of the SDLC.

Despite growing interest, the field is still emerging. Key challenges remain, such as the lack of standardized metrics, the complexity of isolating software energy consumption from hardware and operating system influences, the integration of sustainable practices within established DevOps pipelines, and clarifying the relationship between energy efficiency and broader sustainability outcomes.

The objective of this review is to analyze carbon-aware scheduling methods in software development and energy efficiency, and to identify key metrics as applied to software development and operation. A detailed analysis of existing studies will provide an up-to-date overview of the field, supporting further research on carbon-aware software development practices.

II. Material and Methods

The literature review followed a structured, multi-stage process: identification, screening, eligibility assessment, and inclusion, following the general principles of PRISMA-style reviews.

The search was conducted across major scientific databases, namely IEEE Xplore, Google Scholar, ScienceDirect, ACM Digital Library, Web of Science, and arXiv. This ensured adequate coverage of both peer-reviewed and preprint research. Searches targeted publications from 2020 to 2025 and were restricted to English-language sources.

The following keyword groups were used:

- *Primary terms:* "carbon-aware scheduling," "green software development," "energy efficiency metrics software lifecycle," "energy efficiency metrics," "sustainable software."
- *Secondary terms:* "energy-aware computing," "low-carbon software design."

Duplicates were removed first, leaving 73 unique articles for screening. Articles were then screened in two stages:

1) Title and abstract screening: 24 papers were excluded due to irrelevance (e.g., not addressing software systems, not focused on energy or carbon metrics) or ineligible study type (e.g., non-scientific sources such as blogs).

2) Full-text assessment: of the 49 remaining articles, 2 were excluded after detailed evaluation, resulting in 47 articles that met the inclusion criteria. Papers were included if they examined energy-efficiency metrics within the software development lifecycle, addressed carbon-aware or energy-aware scheduling of computational tasks, and provided either quantitative estimates (e.g., energy consumption, CO_2 savings) or qualitative evaluations (method comparisons, limitations).

Particular emphasis was placed on research that explored carbon-aware scheduling and cost-sensitive workload planning, as well as studies presenting or evaluating metrics to assess software sustainability across the SDLC.

III. Result and discussion

A. *Evolution of Key Research Themes*

The discourse on sustainable software has evolved considerably over the past two decades. Initially, the concept was tightly coupled with "performance engineering," where reducing resource consumption (CPU cycles, memory) was primarily a means to improve speed and reduce hardware costs, with energy savings being a welcome byproduct.

The first major shift occurred as researchers began to explicitly target energy as a primary optimization goal. This led to the emergence of energy-aware computing. Early work in this phase focused on the operating system and hardware abstraction layers, developing power models for CPUs and other components. The research community then moved up the stack, investigating how programming languages, compilers, and software architectures contribute to energy usage. This phase was characterized by a focus on energy efficiency, minimizing the watts consumed by a software application to perform a given task [3].

More recently, the theme has matured into carbon-aware computing. This represents a more nuanced understanding of environmental impact, recognizing that not all energy is created equal. The carbon intensity of electricity, the amount of greenhouse gas emitted per kilowatt-hour (kWh), varies significantly based on the energy source mix (e.g., renewables versus fossil fuels) of the electrical grid at a given time and location. Carbon-aware software, therefore, does not just aim to use less energy, it aims to consume energy when and where it is "cleanest." This has led to the development of sophisticated scheduling techniques that align computational workloads with periods of low carbon intensity [4], [5].

A holistic perspective is developing -one that considers all stages of the software application lifecycle, including requirements engineering, UI/UX design, deployment, maintenance, and eventual decommissioning. This view argues that sustainability must be a cross-cutting concern, integrated into every stage of software development [6].

In addition to the lifecycle-wide integration of sustainability, recent investigation has also considered the ethical and societal dimensions of green software [7]. As digital services expand globally, disparities in grid cleanliness across regions mean that software systems can inadvertently externalize environmental costs to more carbon-intensive areas. This raises important questions about environmental justice and the responsibilities of cloud providers in minimizing their overall footprint rather than merely shifting it elsewhere.

Interdisciplinary collaborations between software engineers, environmental scientists, and policy experts have begun to shape frameworks for green software governance, suggesting future regulation or certification schemes that could mandate transparency in energy use or emissions reporting. These initiatives, although still emerging, highlight the necessity of embedding sustainability not just as a technical goal but as a societal obligation within software engineering practice.

A foundational challenge in green software is measurement. The adage "you cannot improve what you cannot measure" is particularly salient. Research into energy efficiency metrics has evolved from coarse-grained, hardware-centric measures to fine-grained, software-centric approaches.

Key approaches include the following:

- Early approaches relied on physical power meters or processor-level instrumentation like Intel's Running Average Power Limit (RAPL) to evaluate the energy draw of entire systems. While accurate, these methods often struggle to attribute consumption to specific software processes or lines of code [8].
- To overcome the limitations of physical measurement, researchers developed statistical and machine learning models to estimate software energy consumption based on high-level performance indicators (e.g., I/O operations, CPU utilization, network packets). A study demonstrated a strong correlation between system-level metrics and energy consumption, paving the way for software-based power estimation models [3], [9].
- "Software-energy-label," a multi-dimensional metric that evaluates the energy efficiency of software applications, is similar to the energy labels on appliances. Other studies have focused on defining metrics relevant to specific domains, such as energy per transaction in database systems or energy per user request in web applications [6], [8].

A primary debate revolves around the trade-off between the accuracy and accessibility of metrics. Direct hardware measurement is the gold standard for accuracy but requires specialized equipment and expertise. Model-based approaches are more accessible and scalable but are subject to estimation errors and may require re-calibration for different hardware and software environments. There is currently no universally accepted standard for measuring and reporting the energy consumption of a software application, making it difficult to compare the "greenness" of different products [3].

TABLE I. EVOLUTION OF RESEARCH IN SUSTAINABLE SOFTWARE

Theme	Sustainability Perspective		
	Main Focus	***Representative Techniques***	***Maturity***
Performance Engineering	Optimize resource use for speed and cost	CPU and memory optimization, HW tuning	High
Energy-Aware Computing	Software-level energy optimization	OS power models, RAPL, efficient algorithms	Medium-High
Carbon-Aware Computing	Use energy where carbon is lowest	Time and geo shifting, carbon forecasting	Medium
Lifecycle Sustainability	Sustainability across SDLC	Green requirements, energy-aware design	Low-Medium
Ethical Sustainability	Transparency and environmental justice	Emissions accounting, governance models	Low

B. Carbon-Aware Scheduling

There is a number of methodologies that can be employed to account for carbon intensity within computational processes. These include time-based scheduling, geographic shift, price-aware scheduling, and flexible scaling of resources. Each of these factors deserves individual consideration.

Time-based scheduling is an approach that involves delaying batch and time-insensitive tasks to periods of low carbon intensity on the energy grid. It has been observed that users of cloud services frequently migrate batch tasks to periods of low carbon intensity. A comparable approach is employed within GAIA (Green Aware Instance Allocation), an environmentally oriented scheduler for batch tasks that has been demonstrated to produce substantial emission reductions while exerting a moderate impact on performance and cost. This approach finds application across a wide range of use cases, including data backup processes, machine learning tasks, data distribution, batch processing, and more [7].

The geographic shift approach entails the distribution of computational tasks across data centers or regions that exhibit a reduced carbon footprint. Souza et al. developed CASPER for distributed web services, which dynamically allocates load between geographic regions depending on local carbon intensity and network latency. A series of experiments has been conducted, yielding findings that demonstrate the potential for a carbon reduction of up to 70% while concurrently ensuring the maintenance of Service Level Objectives (SLOs) concerning latency [10]. In a similar vein, Lechowicz et al. proposed PCAPS, a scheduler for computational processes that takes into account both time-dependent carbon intensity and geographical location, as well as task prioritization and ordering. A PCAPS prototype in a cluster of 100 nodes reduced the carbon footprint to 32.9% of the baseline, with no noticeable loss of efficiency [11].

Price-aware scheduling is a price-conscious approach. It is evident from a substantial set of research publications that the importance of the prices of computing resources and services is frequently emphasized [12], [13], [14], [15], [16]. Z. Miao et al. suggest that cloud service and computing providers should take into consideration the carbon intensity of electricity costs and the fluctuations in renewable energy across different locations and times. Indeed, models such as ECMR take into account both carbon intensity and local electricity prices simultaneously, thus minimizing emissions at an acceptable monetary cost. The ECMR algorithm for distributed machine learning tasks has been demonstrated to enhance renewable energy utilization by up to 90.8% while simultaneously reducing carbon emissions by 30% in comparison with the baseline carbon-aware ML methods [17].

In the context of resource allocation, the concept of flexible scaling has been proposed by Hanafy et al. This approach involves the dynamic adjustment of the computing cluster's capacity in response to variations in carbon intensity. In circumstances where the carbon intensity is minimal, the cluster will allocate a greater quantity of resources. Conversely, in instances of elevated emissions, the cluster will allocate a reduced quantity of resources. The prediction of carbon intensity is achieved through the analysis of historical data or the utilization of machine learning techniques. This approach precludes the simultaneous preparation of all tasks, thus circumventing the “buffalo herd” effect, wherein the adoption of a similar low-carbon timeframe can surpass computational capacity, consequently leading to increased carbon emissions. CarboneFlex has demonstrated a 57% reduction in emissions in comparison with conventional task scheduling [18].

Another promising development is the integration of carbon-aware strategies into container orchestration platforms. Kubernetes, for instance, is being extended through plugins and custom schedulers to enable energy-aware and carbon-aware task placements. Research prototypes have demonstrated the feasibility of integrating carbon-intensity forecasts as a scheduling signal, allowing pods to be launched in regions or at times that minimize carbon emissions. These advances open the door to mainstreaming sustainability features in cloud-native systems, though they still face technical barriers in standardization, performance impact, and developer adoption [19].

The study emphasizes that architectural patterns and microservice granularity can substantially impact energy consumption [20]. Fine-grained microservices often result in elevated network traffic and resource duplication, thereby increasing runtime energy use. Xiao et al. show that selecting service co-location or modular reuse patterns can mitigate these inefficiencies and enhance energy performance [21].

Hybrid strategies that combine multiple scheduling techniques (time-based and geographic shifting with price-aware models) have shown superior results in experimental settings, offering flexible trade-offs across cost, latency, and emissions [7], [21], [22]. However, these models demand high-quality, real-time data pipelines for energy pricing and carbon intensity, which remain unreliable or unavailable in many regions [23]. As such, future research must also focus on data infrastructure and interoperability standards to enable wider deployment of carbon-aware systems.

Despite promising results, carbon-aware scheduling has some fundamental problems:

- Geographic shifting itself consumes energy and generates network traffic, the carbon footprint of which must be considered, according to Y. Guo et al. [24]. In some cases, the carbon cost of data transmission can negate the benefits of cleaner energy.

- The Rebound Effect refers to the phenomenon where efforts to maximize green energy efficiency result in increased overall computation, which can ultimately cause a rise in total energy consumption instead of a reduction [25].
- Designing and implementing complex process schedulers requires significant effort and changes to existing container management platforms such as Kubernetes. As noted by P. Wiesner et al., most current systems do not have built-in mechanisms to account for carbon intensity. It can be concluded that time-based shifting is a relatively simple and efficient method [26].

TABLE II. CARBON-AWARE SCHEDULING TECHNIQUES

Technique	Concept Overview		
	Core Idea	***Examples***	***Key Limitations***
Time-Based Scheduling	Delay tasks to low-carbon periods	GAIA, batch deferral	Latency, unsuitable for real-time tasks
Geographic Shifting	Run in cleaner regions	CASPER, PCAPS	Network latency, data transport cost
Price-Aware Scheduling	Use electricity price signals	ECMR	Requires accurate price and carbon data
Flexible Scaling	Scale resources by carbon intensity	CarboneFlex	Needs forecasting, throughput impact
Carbon-Aware Orchestration	Carbon signals in K8s schedulers	Custom K8s plugins	Lack of standardization
Hybrid Models	Combine time, geo, and price	Multi-factor schedulers	Complexity unreliable data streams

Geographic shifting, however, has the potential to significantly reduce emissions, but reliable data on carbon intensity in different regions and accounting for network delays are prerequisites. It is evident that the aforementioned methods frequently presuppose information regarding task duration, power price dynamics, computing resource prices, local carbon emissions, or the capacity for deferred loading of computing resources. This complicates the practical implementation for a substantial number of tasks, including those of significant importance.

C. *Energy efficiency metrics in the SDLC phases*

The assessment of software energy efficiency is a fundamental task, without which progress in the field of green engineering is impossible. At present, there is an absence of a standardized metric to assess the energy efficiency and environmental effectiveness of computational tasks, as well as software development and maintenance activities. It is acknowledged that a variety of metrics may be implemented during the different phases of software development. Each of these metrics possesses its own unique characteristics, advantages, and disadvantages. Current research focuses on developing precise metrics for the various phases of the SDLC. A review of the existing literature shows that certain approaches and metrics are far more commonly used than others.

In the initial phases of the SDLC, the direct measurement of energy consumption is often challenging. Consequently, researchers propose the use of indirect metrics. To achieve this objective, static code characteristics are analyzed and correlated with CPU and memory resource consumption. These characteristics include cyclomatic complexity, the use of specific data structures, and code length [27]. Several studies have demonstrated a strong correlation between these metrics and energy efficiency [28], [29]. However, other studies have shown that compilation and processor-level optimizations can make these dependencies non-linear and unpredictable [30], [31].

In the following section, a series of more direct approaches are proposed for the testing phase. For instance, incorporating energy profiling tools, such as Intel Power Gadget, into Continuous Integration/Continuous Delivery (CI/CD) pipelines enables the automated assessment of energy expenditure during the execution of tests. This allows for the identification of "energy regressions," i.e., code changes that inadvertently increase energy consumption [32]. The primary limitation in this context is that results depend heavily on the specific characteristics of the hardware and software environment, which hinders comparison and generalization.

As Kruglov and Succi observe, in the initial phases of development, metrics such as module complexity, coupling, and cohesion can be evaluated. The authors note that metrics of code cohesion show a stronger correlation with energy consumption than metrics of size or inheritance [33]. This helps identify "dark zones" of potential inefficiency during the code design and implementation stages. However, significant energy consumption data only becomes available at later phases, i.e., during software testing and deployment. Therefore, end-to-end tracking of metrics across the entire SDLC is necessary.

Direct metrics of power consumption use either hardware meters or software models, such as RAPL, which provide estimates of power usage by processor components. The issue with models like RAPL is that they do not account for the consumption of RAM, disks, NICs, and other components, which can lead to underestimates of total energy use [34].

In the context of software deployment, it is imperative to meticulously measure two critical metrics: the power consumption of services and the load on servers. In operational mode, the carbon footprint (CO_2-equivalent) and energy consumption in kWh per unit of workflow, such as per request or transaction, are frequently utilized. As widely acknowledged in the academic community, prevailing green infrastructure metrics, such as Power Usage Effectiveness (PUE), Data Center Infrastructure Efficiency (DCiE), and Carbon Usage Effectiveness (CUE), focus on data centers. However, these metrics do not account for software aspects or load fluctuations at the application level [35].

New initiatives propose considering the efficiency "inside" servers (SPUE) or calculating Software Carbon Intensity (SCI), the normalized carbon footprint of software per functional unit [36], [37], [38]. The trend toward standardization of SCI in ISO 14064/21031 reflects recognition of the need for software-level metrics [39].

The SCI can be calculated using the following formula:

$$SCI = (E \times I + M) / R \tag{1}$$

where E is the energy consumed by the software, I is the carbon intensity, M is the carbon footprint associated with hardware production, and R is the functional unit (e.g., number of users or API requests).

A limitation of the SCI metric is the difficulty of accurately measuring the value of M and defining a relevant functional unit R for complex systems, as previously outlined.

T. Simon et al. emphasize that their evaluation model distributes the overall impact between the “development” and “use” phases and demonstrate that the optimization of just one phase can result in a shift of the burden to the other phase [40]. In their example, the significance of the impact of the development phase was given greater weight, despite the focus traditionally being on operations. It is imperative to recognize that the evaluation of any metric at a specific stage of the SDLC is crucial. As Kruglov and Succi have highlighted, a comprehensive assessment of a project's performance and environmental impact can only be achieved through the integration of measurements across all developmental stages [33].

A considerable number of methodologies have been demonstrated to result in substantial emission reductions; however, it is crucial to acknowledge that these outcomes are frequently accompanied by trade-offs. For instance, Hanafy et al. demonstrate that as carbon savings increase, task delays and costs also rise due to idling reserves. The authors observe that their algorithms achieve a twofold increase in carbon savings for every percentage point increase in cost, concomitantly reducing the additional delay by 26% [7]. In other words, the consistent reduction in energy demands frequently necessitates the allocation of resources and additional time, thereby constraining implementation in critical systems.

The efficacy of such estimation and control methods is constrained by assumptions regarding the availability of data on load dynamics and energy sources. Algorithms frequently presuppose precise prediction of network carbon intensity and task duration. In the event of such data being inaccurate, the decisions made may be suboptimal. The issue of delayed task execution is also pertinent. It is noteworthy that not all studies account for network delay in geographic migration, although the issue is addressed in CASPER [10].

Another critical direction involves the automation of sustainability evaluation within development workflows. Upcoming tools seek to provide real-time energy feedback to developers by integrating estimators and profilers directly into IDEs and version control systems. For example, plug-ins can highlight energy hotspots in code as developers write it, allowing for just-in-time greenness corrections. While still in the early stages, such tooling has the potential to transform sustainability from a late-stage consideration to a core part of the coding process.

Additionally, recent work explores how AI-assisted refactoring tools might recommend low-energy alternatives for common patterns or inefficient loops [41]. These innovations reflect the increasing alignment between green software engineering and developer productivity ecosystems. Cross-field research is beginning to explore the psychological and behavioral factors that influence how developers respond to energy metrics, suggesting that future tools must be not only accurate but also actionable and motivating to drive change in software design practices.

Nevertheless, the identified works form the foundations of green software engineering approaches, indicating directions for further research, integrating metrics throughout the SDLC, automating code greenness control, and considering economic factors in resource scheduling.

Another rising aspect of energy efficiency in the SDLC is the integration of sustainability considerations into software architecture and design patterns. Research has shown that architectural choices, such as the adoption of microservices versus monolithic structures, can have a significant impact on energy consumption [19]. For example, microservice-based systems may increase network traffic and idle time due to container overhead and distributed communication, whereas monolithic designs, while less scalable, can result in lower baseline energy use under certain conditions. This highlights the need for sustainability-aware architecture trade-off analysis, where energy implications are considered alongside maintainability, performance, and scalability.

Similarly, the choice of programming language and runtime environment has been scrutinized in recent studies [42], [43], [44]. For instance, compiled languages such as C++ or Rust typically produce more energy-efficient executables than interpreted languages like Python or JavaScript, though the development speed and ecosystem support may differ. Benchmarks across common workloads (e.g., compression, parsing, web serving) confirm that language-level decisions are not trivial in the context of energy use. The growing interest in domain-specific languages (DSLs) for energy-constrained environments (such as IoT and edge computing) further exemplifies this direction, suggesting the co-evolution of tools, languages, and sustainable practices. The field is also beginning to explore the long-term effects of software bloat and feature creep on sustainability.

As software systems accumulate features, dependencies, and technical debt, they tend to grow in size and complexity, often requiring more resources to run, update, and maintain. This phenomenon, known as “code rot” or “software obesity”, introduces persistent overheads, especially when running on cloud infrastructure where idle resources still consume electricity [45]. Lean software engineering principles are being revisited through a sustainability lens, encouraging minimalist, modular, and refactorable designs as mechanisms for long-term energy savings.

TABLE III. ENERGY OPTIMIZATION APPROACHES

SDLC Phase	**Evaluation Framework**		
	Optimization Approach	***Methods***	***Effectiveness***
Planning	High-level energy goals	Green requirements, sustainability guidelines	Medium
Analysis	Evaluate potential energy impact	Software modeling, architectural trade-offs	Medium

SDLC Phase	**Evaluation Framework**		
	Optimization Approach	***Methods***	***Effectiveness***
Design	Energy-aware architecture	Component selection, modularity, low-power design patterns	Medium-High
Implementation	Efficient code development	Energy-efficient algorithms, linters, static analysis	Medium-High
Testing	Energy regression and monitoring	RAPL, Intel Power Gadget, automated energy tests	Medium
Maintenance	Reduce long-term energy cost	Refactoring, code bloat reduction, continuous monitoring	Low-Medium

Optimizing reuse without bloating systems is thus an active area of exploration. Finally, education and cultural change within the software engineering profession are becoming central to the green software movement. Studies have shown that many developers remain unaware of the energy implications of their design and implementation decisions, or lack the tools and incentives to prioritize sustainability [46], [47]. This has spurred the creation of educational materials, guidelines (e.g., the Green Software Foundation's principles), and even university courses on sustainable computing.

Bridging the knowledge gap between energy modeling experts and everyday developers is crucial if green practices are to become mainstream rather than niche. As sustainability becomes a shared responsibility, fostering a culture that values efficiency, transparency, and accountability will be just as important as advancing technical solutions.

IV. Conclusion

The review demonstrates that carbon-aware scheduling and energy efficiency metrics are active research areas for green software development from 2020 onwards. It is vital to employ critical scheduling strategies, including temporal shifting of tasks, geographic and price-based load balancing, and dynamic resource scaling. Experimental evidence shows that these techniques can reduce the carbon footprint of applications and computational processes by tens of percent. However, many of these methods remain at the prototype stage, and their deployment often involves trade-offs between emissions, performance, and cost.

A key finding is the necessity for end-to-end measurement across all stages of the SDLC. Current metrics inadequately capture software-level energy efficiency or account for application performance. The Software Carbon Intensity (SCI) metric, while promising, remains immature and requires rigorous validation. Future work should focus on developing standardized, cross-platform metrics that integrate energy, carbon, and performance indicators, enabling fair comparison and benchmarking of software systems.

Key strategies to promote sustainable software development include:

1) Integration of energy metrics into development workflows: Incorporate energy profiling, SCI estimators, and carbon-aware alerts directly into IDEs, CI/CD pipelines, and testing frameworks to enable developers to optimize energy use as they code.

2) Adoption of carbon-aware scheduling in cloud environments: Encourage cloud providers to expose real-time carbon intensity data and pricing signals, enabling applications to dynamically shift workloads across time and geography.

3) Standardization and benchmarking: Establish open datasets for carbon intensity, shared benchmarks for energy efficiency, and guidelines for SCI reporting to foster transparency and comparability.

4) Education and cultural change: Train software engineers on sustainable design patterns, energy-efficient programming practices, and the environmental implications of software architecture choices.

5) Policy and regulatory alignment: Encourage policymakers to incentivize sustainable software development through certifications, disclosure requirements, or carbon-aware procurement policies.

6) Interdisciplinary collaboration: Promote partnerships among academia, industry, and environmental science to co-develop frameworks that balance performance, cost, and sustainability in real-world software systems.

Looking ahead, the successful realization of green software systems will require a synergy of technical, organizational, and policy innovations. Scalable, interoperable infrastructures that operationalize sustainability without compromising functionality or accessibility must become the norm. As digital services underpin critical societal functions, software energy efficiency is no longer a niche concern and has become a crucial enabler of climate action. Only through coordinated efforts across stakeholders can the software industry make a meaningful contribution to global emissions reduction goals.

References

[1] J. C. T. Bieser and S. Peters, "A review of assessments of the greenhouse gas footprint of digital devices," *Telecommun. Policy*, vol. 47, no. 5, 2023.

[2] World Bank / ITU, *Measuring the Emissions & Energy Footprint of the ICT Sector*, World Bank, 2024.

[3] W. Wysocki, I. Miciuła, and P. Plecka, "Methods of Improving Software Energy Efficiency: A Systematic Literature Review and the Current State of Applied Methods in Practice," *Electronics*, vol. 14, no. 7, p. 1331, 2025, doi: 10.3390/electronics14071331.

[4] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, and I. Zhang, "Treehouse: A Case For Carbon-Aware Datacenter Software," in *Proc. HotCarbon '22*, 2022.

[5] Y. G. Kim, U. Gupta, A. McCrabb, Y. Son, V. Bertacco, D. Brooks, and C.-J. Wu, "GreenScale: Carbon-Aware Systems for Edge Computing," *arXiv*, Apr. 2023, doi: 10.48550/arXiv.2304.00404.

[6] S. McGuire, E. Shultz, B. Ayoola, and P. Ralph, "Sustainability is Stratified: Toward a Better Theory of Sustainable Software Engineering," in *Proc. ICSE '23*, May 2023, pp. 1996–2008, doi: 10.1109/ICSE48619.2023.00169.

[7] W. A. Hanafy, Q. Liang, N. Bashir, A. Souza, D. Irwin, and P. Shenoy, "Going Green for Less Green: Optimizing the Cost of Reducing Cloud Carbon Emissions," in *Proceedings of the 29th ACM International*

Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '24), La Jolla, CA, USA, Apr. 2024, pp. 479–496.

[8] P. Thamm, "Strategies to Measure Energy Consumption Using RAPL During Workflow Execution on Commodity Clusters," *arXiv*, May 2025, doi: 10.48550/arXiv.2505.09375.

[9] G. Raffin and D. Trystram, "Dissecting the software-based measurement of CPU energy consumption: a comparative analysis," *arXiv*, Jan. 2024, doi: 10.48550/arXiv.2401.15985.

[10] A. Souza, S. Jasoria, B. Chakrabarty, A. Bridgwater, A. Lundberg, F. Skogh, A. Ali-Eldin, D. Irwin, and P. Shenoy, "CASPER: Carbon-Aware Scheduling and Provisioning for Distributed Web Services," in *Proc. 14th Int'l Green & Sustainable Computing Conf. (IGSC '23)*, Toronto, ON, Canada, Oct. 2023, pp. 67–73, doi: 10.1145/3634769.3634812.

[11] A. Lechowicz, R. Shenoy, N. Bashir, M. Hajiesmaili, A. Wierman, and C. Delimitrou, "Carbon- and Precedence-Aware Scheduling for Data Processing Clusters," *CoRR*, vol. abs/2502.09717, Feb. 2025, doi: 10.48550/arXiv.2502.09717.

[12] J. Bader, J. Irion, J. Kappel, J. Witzke, N. Fomin, D. Sherifi, and O. Kao, "Learning Process Energy Profiles from Node-Level Power Data," *arXiv preprint arXiv:2511.13155*, Nov. 2025.

[13] M. Raeisi-Varzaneh, O. Dakkak, Y. Fazea, and M. G. Kaosar, "Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems," *Cluster Computing*, vol. 27, pp. 13,407–13,419, Dec. 2024.

[14] X. Sun, Z. Wang, Y. Wu, H. Che, and H. Jiang, "A price-aware congestion control protocol for cloud services," *J. Cloud Comput.*, vol. 10, no. 1, Art. 55, Nov. 2021.

[15] S. Tuli, G. Casale, and N. R. Jennings, "MetaNet: Automated dynamic selection of scheduling policies in cloud environments," *arXiv preprint arXiv:2205.10642*, May 2022.

[16] S. G. Ahmad, T. Iqbal, and E. U. Munir, "Cost optimization in cloud environment based on task deadline," *J. Cloud Comput.*, vol. 12, Art. 9, Jan. 2023.

[17] Z. Miao, L. Liu, H. Nan, W. Li, X. Pan, X. Yang, M. Yu, H. Chen, and Y. Zhao, "Energy and carbon-aware distributed machine learning tasks scheduling scheme for the multi-renewable energy-based edge-cloud continuum," *Science and Technology for Energy Transition*, vol. 79, Article 82, 2024, doi: 10.2516/stet/2024076.

[18] W. A. Hanafy, L. Wu, D. Irwin, and P. Shenoy, "CarbonFlex: Enabling Carbon-aware Provisioning and Scheduling for Cloud Clusters," *CoRR*, vol. abs/2505.18357, May 2025.

[19] X. Xiao, C. Gao, and J. Bogner, "On the Effectiveness of Microservices Tactics and Patterns to Reduce Energy Consumption: An Experimental Study on Trade-Offs," in *Proc. 22nd Int'l Conf. on Software Architecture (ICSA '25)*, Odense, Denmark, Apr. 2025, pp. 164–175, doi: 10.1109/ICSA65012.2025.00025.

[20] O. Poy, M. Á. Moraga, F. Garcia, and C. Calero, "Impact on energy consumption of design patterns, code smells and refactoring techniques: A systematic mapping study," *J. Syst. Softw.*, vol. 222, no. 15, p. 112303, Dec. 2024, doi: 10.1016/j.jss.2024.112303.

[21] X. Xiao, "Architectural Tactics to Improve the Environmental Sustainability of Microservices: A Rapid Review," *CoRR*, vol. abs/2407.16706, July 2024.

[22] E. Breukelman, S. Hall, G. Belgioioso, and F. Dörfler, "Carbon-Aware Computing in a Network of Data Centers: A Hierarchical Game-Theoretic Approach," in *Proc. 2024 European Control Conference (ECC)*, Stockholm, Sweden, Jun. 25–28, 2024, pp. 798–803, doi: 10.23919/ECC64448.2024.10591261.

[23] N. Asadov, V. C. Coroamă, M. Franzil, S. Galantino, and M. Finkbeiner, "Carbon-Aware Spatio-Temporal Workload Shifting in Edge–Cloud Environments: A Review and Novel Algorithm," *Sustainability*, vol. 17, no. 14, p. 6433, Jul. 2025, doi: 10.3390/su17146433.

[24] Y. Guo, A. Tomlinson, R. Su, and G. Porter, "The Effect of the Network in Cutting Carbon for Geo-shifted Workloads," CoRR, vol. abs/2504.14022, Apr. 2025.

[25] N. L. Woodruff, D. Schall, M. F. P. O'Boyle, and C. Woodruff, "When Does Saving Power Save the Planet?," in *Proc. HotCarbon '23*, Jul. 2023, pp. 1–6, doi: 10.1145/3604930.3605719.

[26] P. Wiesner, M. Steinke, H. Nickel, Y. Kitana, and O. Kao, "Software-in-the-Loop Simulation for Developing and Testing Carbon-Aware Applications," *Software: Practice and Experience*, vol. 53, no. 12, pp. 2362–2376, 2023, doi: 10.1002/spe.3275.

[27] F. Hussin, S. A. N. Md Rahim, N. S. M. Hatta, M. K. Aroua, and S. A. Mazari, "A systematic review of machine learning approaches in carbon capture applications," *Journal of CO₂ Utilization*, vol. 71, Art. 102474, May 2023, doi: 10.1016/j.jcou.2023.102474.

[28] J. Mancebo, C. Calero, and F. García, "Does maintainability relate to the energy consumption of software? A case study," *Software Qual. J.*, vol. 29, no. 1, pp. 101–127, Jan. 2021, doi: 10.1007/s11219-020-09536-9.

[29] H. M. Alvi, H. Majeed, H. M. Mujtaba, and M. O. Beg, "MLEE: Method level energy estimation — A machine learning approach," *Sustain. Comput. Inform. Syst.*, vol. 31, p. 100594, 2021, doi: 10.1016/j.suscom.2021.100594.

[30] K. Chan-Jong-Chu, T. Islam, M. M. Exposito, S. Sheombar, C. Valladares, O. Philippot, E. M. Grua, and I. Malavolta, "Investigating the Correlation between Performance Scores and Energy Consumption of Mobile Web Apps," in *Proc. 24th Evaluation & Assessment in Software Engineering (EASE '20)*, Trondheim, Norway, Apr. 2020, pp. 190–199, doi: 10.1145/3383219.3383239.

[31] N. Schmitt, J. Bucek, J. Beckett, A. Cragin, K.-D. Lange, and S. Kounev, "Performance, power, and energy-efficiency impact analysis of compiler optimizations on the SPEC CPU 2017 benchmark suite," in *Proc. 2020 IEEE/ACM 13th Int. Conf. on Utility and Cloud Computing (UCC),* 2020, pp. 292–301, doi: 10.1109/UCC48980.2020.00047.

[32] B. Prieto, J. J. Escobar, J. C. Gómez-López, A. F. Díaz, and T. Lampert, "Energy Efficiency of Personal Computers: A Comparative Analysis," *Sustainability*, vol. 14, no. 19, Art. 12829, Oct. 2022, doi: 10.3390/su141912829.

[33] A. Kruglov, G. Succi, and Z. Kholmatova, "Metrics of Sustainability and Energy Efficiency of Software Products and Process," in Developing Sustainable and Energy-Efficient Software Systems, *SpringerBriefs in Computer Science*, Cham, Switzerland, 2023, pp. 19–26, doi: 10.1007/978-3-031-11658-2_2.

[34] Z. Zhang, S. Liang, F. Yao, and X. Gao, "Red Alert for Power Leakage: Exploiting Intel RAPL-Induced Side Channels," in *Proc. 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*, Hong Kong, June 2021, pp. 162–175, doi: 10.1145/3433210.3437517.

[35] A. Safari, H. Sorouri, A. Rahimi, and A. Oshnoei, "A Systematic Review of Energy Efficiency Metrics for Optimizing Cloud Data Center Operations and Management," *Electronics*, vol. 14, no. 11, Art. 2214, May 2025, doi: 10.3390/electronics14112214.

[36] Green Software Foundation. "Software Carbon Intensity (SCI) Specification v1.0," *Green Software Foundation*, 2021. [Online]. Available: https://sci.greensoftware.foundation/

[37] UBS, "Baselining Software Carbon Emissions: UBS Use Case," *Green Software Foundation*, 2023. [Online]. Available: https://greensoftware.foundation/articles/baselining-software-carbon-emissions-ubs-use-case/

[38] A. Schmidt, G. Stock, R. Ohs, L. Gerhorst, B. Herzog, and T. Hönig, "carbond: An Operating-System Daemon for Carbon Awareness," in *Proc. 2nd Workshop on Sustainable Computer Systems, HotCarbon '23*, Boston, MA, USA, 2023, pp. 1-10, doi: 10.1145/3604930.3605707.

[39] ISO/IEC 21031:2024. *Information technology - Software carbon intensity - Measurement and reporting framework*, International Organization for Standardization, Geneva, Switzerland, 2024. [Online]. Available: https://www.iso.org/standard/86612.html/

[40] T. Simon, P. Rust, R. Rouvoy, and J. Penhoat, "Uncovering the environmental impact of software life cycle," in *Proc. 2023 IEEE Int. Conf. on ICT for Sustainability (ICT4S '23)*, 2023, pp. 176–187, doi: 10.1109/ICT4S58814.2023.00026.

[41] A. Imran, T. Kosar, J. Zola, and M. F. Bulut, "Towards Sustainable Cloud Software Systems through Energy-Aware Code Smell Refactoring," in *Proc. 2024 IEEE 17th International Conference on Cloud Computing (CLOUD '24)*, Shenzhen, China, 2024, pp. 223–234, doi: 10.1109/CLOUD62652.2024.00034.

[42] N. Marini, L. Pampaloni, F. Di Martino, R. Verdecchia, and E. Vicario, "Green AI: Which Programming Language Consumes the Most?," *arXiv*, 2024.

[43] N. van Kempen, H.-J. Kwon, D. T. Nguyen, and E. D. Berger, "It's Not Easy Being Green: On the Energy Efficiency of Programming Languages," *arXiv*, 2024.

[44] N. Fatema Reya, A. Ahmed, T. Zaman, and Md. M. Islam, "GreenPy: Evaluating Application-Level Energy Efficiency in Python for Green Computing," *Annals of Emerging Technologies in Computing*, vol. 3, 2023, pp. 92–110, doi:10.33166/aetic.2023.03.005.

[45] M. Couto, D. Maia, J. Saraiva, and R. Pereira, "On Energy Debt: Managing Consumption on Evolving Software," in *Proc. 3rd IEEE/ACM Int. Conf. on Technical Debt (TechDebt '20)*, Seoul, Republic of Korea, June 2020, pp. 62–66, doi: 10.1145/3387906.3388628.

[46] S. U. Lee, N. Fernando, K. Lee & J.-G. Schneider, "A Survey of Energy Concerns for Software Engineering," *Journal of Systems and Software*, vol. 210, article no. 111944, 2024, doi: 10.1016/j.jss.2023.111944.

[47] H. Noman, M. Freed, and S. Jain, "An Exploratory Study of Software Sustainability at Early Stages," *Sustainability*, vol. 14, no. 14, p. 8596, Jul. 2022, doi: 10.3390/su14148596

AUTHORS

Mikita Piastou

Mikita Piastou is a senior full-stack software engineer with over eight years of experience in the technology industry, specializing in both data and software development. He holds a Master's degree in Computer Science from the University of West Georgia and has contributed to several publications on artificial intelligence, software development, and broader computer science topics. Throughout his career, Mikita has designed, implemented, and maintained highly scalable and reliable software solutions, integrated cutting-edge AI technologies, and optimized complex data systems for various organizations across multiple sectors. He is also a member of IEEE, actively engaging with the professional community. Beyond his technical expertise, he serves as a judge in hackathons and provides guidance to fellow developers. Outside of work, he enjoys keeping up with the latest technology trends, experimenting with new programming frameworks, working on creative side projects that contribute to the advancement of society, and exploring entrepreneurship through startup competitions and collaborative innovation ventures.