Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Sustainable and Green Software Architecture: Principles, Frameworks, And Emerging Research Directions

Anandan Sonaimuthu

Cyma Systems Inc, USA

ARTICLE INFO

ABSTRACT

The rapid expansion of digital infrastructure and software-intensive systems has significantly increased the energy consumption and environmental footprint of information and communication technologies (ICT), catalysing the emergence of sustainable and green software architecture as a critical engineering discipline. This article offers an extensive academic examination of the fundamental concepts, architecture, performance metrics, and methodologies involved in the development of environmentally sustainable software systems. Drawing from recent peer-reviewed literature in sustainable computing and green software engineering, the study examines how architectural decisions, including service granularity, workload scheduling, and resource management, directly shape environmental outcomes across the software lifecycle. This article analyzes a multi-dimensional approach to sustainability involving environmental, economic, social, and technical aspects along with conventional reference models and green architectural approaches such as microservices, serverless architectures, edge computing, and data reduction techniques. The researchers analyze important performance metrics and findings associated with current issues, especially the lack of sustainability metrics. Attention is drawn to future research directions based on sustainable practices such as carbon-aware cloud computing, artificial intelligence-based energy management, sustainable AI systems, and circular computing. It is noted that incorporating sustainable concepts into software architectures is vital for environmentally sustainable computing environments.

Keywords: Green Software Architecture, Sustainable Software Engineering, Carbon-Aware Computing, Energy-Efficient Software Design, Green Architectural Patterns

1. Introduction

The global proliferation of software-intensive systems spanning mobile applications, cloud platforms, enterprise middleware, and large-scale data analytics pipelines has dramatically increased energy consumption across digital infrastructure. The information and communication technologies (ICT) sector contributes a growing and increasingly significant share of global greenhouse gas emissions, driven by the energy demands of data centers, wide-area networks, and the expanding base of end-user devices that collectively underpin modern digital economies [1]. This environmental impact is not incidental but built into the system architecture and requires design intervention. The traditional Green IT movement has emphasized efficiency considerations for hardware, such as energy-efficient server design and superior cooling and power management infrastructures. However, a fundamental

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

recognition has emerged: software design decisions exert a direct and often dominant influence on hardware utilization and energy consumption, frequently negating hardware-level efficiency gains when software-level inefficiencies are left unaddressed [2]. Redundancies, inefficient algorithms, unneeded data movements, and ineffective service collaborations add up to become energy costs that hardware innovations alone cannot address [3]. The concept of sustainable software architecture, however, represents a distinct discipline within software engineering that tries to resolve the mentioned problem through integration of environmental aspects in the early stages of system design. Such solutions and designs include techniques that reduce energy consumption, prevent waste of computing resources, extend the lifetime of a system, and ensure the maintainability and flexibility of these systems [4]. Sustainable software architecture considers energy efficiency and environmental concerns as additional objective functions, in contrast to performance architecture, which focuses on improving effectiveness and execution speed [5]. Recent developments demonstrate the significance of considering sustainability concerns throughout all phases of the software development lifecycle, from requirements elicitation and architecture to design, deployment, operation, and beyond. This article will address recent findings on sustainable software architecture by providing an overview of existing concepts, principles, reference architectures, patterns, and metrics and empirical studies as well as research directions. This work aims to provide an overview not only for academic researchers interested in broadening their knowledge of sustainable software architecture but also for practitioners in need of evidence-based best practices.

2. Background and Conceptual Foundations

2.1 Sustainable Software Engineering

Sustainable software engineering is the practice of making, running, and designing software in ways that have the least effect on the environment while also making sure that the system will last for a long time. Sustainability is not merely about optimizing resource consumption; instead, it is an area of study that incorporates a multifaceted perspective on sustainability, considering the environment, economy, society, and technology in the form of an engineering approach [4]. The comprehensive view offered sets sustainable software engineering apart from more limited approaches in green computing.

Two principal goals structure the discipline. The first is to make the ICT sector's direct impact on the environment smaller by using energy-efficient design, less hardware, and carbon-conscious ways of doing business. The second approach, sometimes referred to as software-enabled sustainability, is the application of software technology to enable sustainability impacts in other industries, such as power grids, transportation, and manufacturing, thus multiplying the positive environmental impact of well-engineered software systems [1]. It has been shown through research that both approaches are synergistic and not contradictory: a well-engineered software system is more capable of being used as a tool for environmental regulation.

It should be noted that sustainability in software engineering goes beyond environmental concerns and includes economic and social sustainability. Economic sustainability guarantees that the system is affordable to use and maintain throughout its expected lifetime. Social sustainability considers human aspects, such as usability, accessibility, and cognitive burden on developers and maintenance engineers. These sustainability considerations are mutually dependent: economically unsustainable software will be disposed of too early, reversing any environmental benefits, whereas socially unsustainable software will become increasingly resource-intensive due to accumulating technical debt [5].

2.2 Green Software Architecture

Software architecture occupies a foundational position in determining system sustainability because architectural decisions establish the structural framework within which all subsequent design and

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

implementation choices operate. Specifically, the architecture impacts the patterns of resource usage, scalability capability, maintenance features, and computational efficiency at the systemic level [6]. This is because decisions about these elements are made in the initial phases of development and are difficult and expensive to change, making them highly influential over the system's life cycle. Decisions related to distribution and monolithic system structure, size of services in SOA architectures, use of caching, and choice between synchronous and asynchronous communication mechanisms, among others, could lead to noticeable distinctions in energy usage and sustainable operation [7]. An architecture based on synchronous communication with minimal services can also be expensive to manage and may involve latencies in the connections. On the other hand, an architecture that is event-driven or based on asynchronous communication can help minimize coupling. The concept of green software architecture captures this notion through considering energy and environmental impact as one of the architectural qualities, in addition to performance, reliability, and security [8].

3. Dimensions of Sustainability in Software Architecture

Sustainable software architecture is a composite quality that manifests across multiple interrelated dimensions. The understanding of all of these dimensions is crucial when developing evaluation frameworks that would account for the totality of architectural sustainability as well as establishing the priorities based not only on environmental concerns but also practical organizational considerations [5].

3.1 Environmental Sustainability

The most extensively researched dimension is environmental sustainability, which concentrates on energy saving, reduction in carbon footprints, and decreasing the usage of computational resources during the software development process. The most important architectural solutions in terms of achieving environmental sustainability are applying energy-efficient algorithms, which would require fewer computational steps to perform the same function; minimizing the usage of memory space and improving access to data storage facilities; reducing unproductive communications across the networks with the help of advanced caching techniques and intelligent data locality; and applying advanced workload scheduling techniques, which ensure the synchronization of computing needs with available energy resources [1]. Empirical evidence confirms that improvements achieved at the software design stage can result in much more significant energy savings than hardware innovations, which could not be made possible at all [3].

3.2 Economic Sustainability

Economic sustainability ensures the financial viability of software products. Architectures that are economically sustainable avoid unnecessary over-provisioning of compute resources, minimize idle infrastructure, and adapt dynamically to workload variability through auto-scaling and demand-responsive design [2]. Cost-aware workload scheduling, which optimizes not only for computational throughput but also for the unit cost of computing resources at any given time, represents an important intersection between economic and environmental sustainability objectives, as lower-cost energy periods frequently correspond to higher renewable energy availability [9].

3.3 Social Sustainability

Social sustainability takes into account the human elements involved in the software architecture, including programmer efficiency, the comprehensibility of the software's architecture, and its ability to be maintained over time by different people within evolving organizational settings [5]. Socially sustainable architectures simplify unnecessarily complex tasks, establish clear separation of responsibilities, and enable efficient knowledge transfer, thus decreasing the amount of human work involved in modifying systems, which, in turn, saves energy due to the inefficiencies inherent in

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

extended development periods. In addition, socially sustainable system designs ensure usability for individuals with different skills and varying levels of connection.

3.4 Technical Sustainability

Technical sustainability provides software systems with adaptability, maintainability, and scalability through their extended lives without having to periodically replace entire systems, which is both inefficient and wasteful [6]. These important architectural aspects are modularity to allow change without causing disruption to the whole system, low coupling between the system's modules that minimize the impact of changes on other modules, and well-specified contract interfaces to maintain the substitutability of the system components. "System designs that explicitly emphasize technical sustainability show much lower accumulation of technical debt than others and are thus easier to maintain [7].

4. Architectural Principles for Green Software Systems

Research in sustainable computing has converged on a set of core architectural principles that collectively define the design philosophy of green software systems. These principles function as high-level constraints and objectives that guide architectural decision-making and trade-off resolution throughout the design process [1, 10].

4.1 Energy Efficiency

The principle of energy efficiency stipulates that the architecture of software should ensure that the least amount of computational effort is expended to deliver each unit of work produced. Such requirements translate into practical considerations, including the employment of effective database querying practices that avoid full table scans and the loading of superfluous data, the avoidance or postponement of any unnecessary background computation that does not add direct value for the user, and the application of efficient communication techniques that decrease serialization and transmission efforts [2]. Research demonstrates that architectural-level energy efficiency decisions made during initial system design are substantially less costly to implement than equivalent optimizations applied retroactively to operational systems [11].

4.2 Resource Efficiency

Maximizing hardware resource utilization is part of the concept of resource efficiency as an architectural concept, whereby all activities within systems must be designed in such a way as to avoid any wastage of compute resources, unused memory, or under-utilization of network facilities [12]. The use of container technologies facilitates the ability to allocate hardware resources in a very fine-grained manner while ensuring that each workload operates independently. Serverless architectures can be considered an extension of this concept through ensuring that only resources needed for a particular function are allocated to the task when it is being executed and released immediately afterward [15].

4.3 Carbon Awareness

The principle of carbon awareness in architecture goes beyond mere energy efficiency because the carbon impact of the electricity used to power software applications changes from one location to another, depending on time and the energy sources being utilized [9]. Architectures designed with carbon awareness enable workload scheduling to be influenced by real-time signals about grid carbon intensity, routing computationally intensive but time-flexible workloads to geographic regions or time windows where renewable energy is abundant. The time-space principle brings time and space into the architecture as design parameters that must be considered explicitly in the form of delaying, mobility, and carbon data streams [9].

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4.4 Hardware Efficiency

Efficiency in hardware dictates that the architectures of the software systems should aim to prolong the lifespan of the hardware and reduce the need for upgrading it. The software architecture of any such sustainable system should have commitments to backward compatibility and proper memory management, which involves avoiding unnecessary memory allocations and excessive garbage collections, and also avoid using computational methods that will require specific hardware generations [4]. In this case, the sustainability in the use of existing hardware helps save the embodied carbon from manufacturing new hardware, which is one of the sources of ICT's carbon footprint [13].

4.5 Measurement and Monitoring

Measurement and monitoring is a principle that states that any sustainable system should measure and monitor continuously its environmental metrics as part of its architectural requirements and not as a post-implementation afterthought [12]. Environmental metrics include, but are not limited to, power consumption per transaction/request, carbon emissions per workload, and utilization rates of CPU and memory per workload, among others. Sustainable metrics that should be measurable are required for sustainability improvement, sustainability maintenance, or for avoiding degradation of sustainability due to future changes. This principle aligns sustainable software architecture with the broader discipline of quality attribute monitoring and treats environmental impact as a measurable and governable system property [12].

5. Reference Models for Sustainable Software Architecture

5.1 The GREENSOFT Model

One of the first and most complete reference frameworks for green software engineering takes into account sustainability throughout the entire software engineering lifecycle [1]. The model is organized around four primary components: a lifecycle model that maps sustainability concerns to corresponding development phases; a set of sustainability metrics applicable at each phase; stakeholder-specific process guidelines addressing the distinct roles and responsibilities of developers, system administrators, and end-users; and a software quality assessment framework that extends conventional quality models to incorporate environmental attributes [4].

The lifecycle view of this model becomes especially relevant here due to the recognition of the fact that sustainability effects do not solely occur during the operations lifecycle of software systems but extend throughout the entire lifecycle stages, including requirements gathering, architecture design, development, testing, deployment, and support [8]. The link between sustainability metrics for each lifecycle stage allows us to implement sustainable governance through software lifecycles without needing to redesign current processes entirely. The model has provided a foundational conceptual reference for subsequent frameworks and metrics systems in the sustainable software engineering literature and continues to inform both academic research and applied methodology development [1].

Component	Description	Lifecycle Scope
Lifecycle model	Maps sustainability concerns to each development phase	All phases
Sustainability metrics	Quantifiable indicators applicable at each lifecycle phase	Design, Operation
Stakeholder processes	Role-specific guidelines for developers, administrators, and users	All phases
Quality assessment	Extended quality model incorporating environmental attributes	Evaluation, Maintenance

Table 1: GREENSOFT Model Key Components [3, 6, 7]

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5.2 Sustainable Software Capability Models

Complementing process-oriented frameworks, capability maturity models for sustainable software engineering provide a structured progression of organizational competencies for embedding sustainability governance into software development practice [8]. These models draw on systems thinking to situate software sustainability within broader organizational and societal contexts, recognizing that individual technical decisions are shaped by and must be evaluated within organizational incentive structures, governance frameworks, and industry standards. Capability models emphasize sustainability as an organizational practice rather than a property of individual systems, requiring sustained investment in developer education, tooling, process integration, and performance measurement [5]. The progression through maturity levels enables organizations to benchmark their current sustainability practices, identify capability gaps, and plan targeted improvement initiatives with clear success criteria.

6. Green Architectural Patterns

Architectural patterns refer to proven solutions to design issues that occur regularly. As far as sustainable software is concerned, there is enough empirical evidence collected by scholars regarding some architectural patterns that can contribute toward mitigating their ecological footprint [7].

6.1 Microservices Architecture

The microservices architecture pattern involves breaking down software applications into smaller components with distinct business functions. From a sustainability perspective, microservices allow for selective scaling, which means that only the services that are in high demand are scaled out, rather than scaling the whole monolithic application in response to localized load spikes [2]. The fine granularity of the scale ensures a reduction in over-provisioning and its resulting energy wastage. Furthermore, independent deployment of services facilitates the fine-tuning of computationally intensive processes without impacting the entire system, and enhanced fault isolation mitigates cascading failures that might result in computationally intensive recovery tasks [10]. In empirical studies conducted on microservices, it has been observed that there is an evident reduction in computing resource usage compared to monolithic applications in highly varying workloads [14].

6.2 Serverless Computing

Serverless architecture, also referred to as function-as-a-service, represents a paradigm in which compute resources are allocated dynamically in response to individual invocation events and released immediately upon function completion. This method of execution does not have any wastage of infrastructure, which is known to consume a large share of overall energy consumption within the framework of the traditionally implemented server architecture [15]. Scaling happens automatically based on demand in serverless computing environments; therefore, there will be no need to make any provision for the future maximum capacity requirements. Research shows that serverless computing achieves much better resource utilization efficiency than the traditional server architectures, especially in cases where there is uneven or sporadic traffic [15].

6.3 Edge Computing

Edge computing involves placing computational capabilities closer to the physical location where data is generated and interactions occur, thus avoiding unnecessary long-distance transmission to the central processing units [16]. This architectural framework, from the standpoint of energy consumption, avoids the necessity of using up too much network bandwidth and the related energy, provides less latency due to not having to send round trips to remote servers, and allows for the more efficient processing of locally available data streams. Studies show that implementing edge computing

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

in practical systems results in lower overall system energy consumption, especially for computational jobs suitable for local processing [16, 13].

6.4 Data Minimization Architecture

Data minimization in the context of architecture is based on the idea that applications should only store, process, and collect the data that is absolutely required to achieve their functional goals, thereby minimizing the burden of processing unnecessary amounts of data [12]. The approaches adopted in architectural design involve compressing data during the data collection and storage stages; caching data smartly by returning data queries from the local cache rather than processing the data from primary sources repeatedly; and fetching only relevant data fields instead of fetching entire sets of data. These strategies collectively reduce CPU cycles, memory pressure, storage I/O operations, and network traffic, producing compounded energy savings across the full data processing pipeline [3].

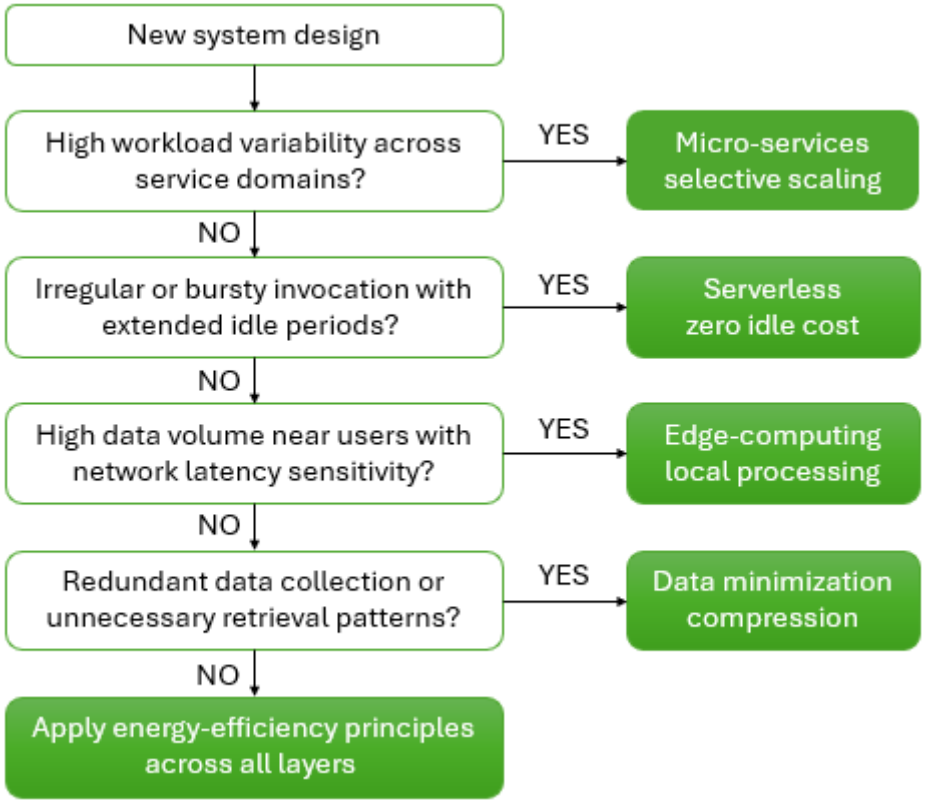

Figure 1: Green architectural pattern selection framework [10, 14, 15, 18]

7. Metrics for Sustainable Software Architecture

A comprehensive evaluation of software sustainability requires quantitative metrics that practitioners can apply practically and measure consistently and that differences in architecture can affect [12]. The lack of consistent measures for sustainability is considered one of the major obstacles hindering progress in sustainable software engineering because without them, there can be no comparison between different architectures, nor can organizations prove any improvements in sustainability [14].

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

7.1 Energy Metrics

Energy metrics measure the amount of energy used directly by software applications, making it the most tangible measure of their environmental footprint. Power consumption per transaction measured as the average electrical energy consumed per completed unit of application-defined work provides a workload-normalized measure that enables comparison across systems of different scales and utilization levels [3]. Energy usage per request, expressed in joules for each served request in network-facing systems, offers a similarly normalized metric for web services and API-based architectures. The following metrics can be gathered by means of hardware interfaces that perform power measurement, software power estimation methods, or reporting tools available on computational platforms.

7.2 Resource Utilization Metrics

Resource utilization metrics are used to measure the effectiveness of the software in exploiting the capabilities of the hardware infrastructure. CPU utilization, defined as the proportion of available processor cycles utilized during workload execution, reflects the effective use of the computational platform and highlights possibilities for workload consolidation or system rightsizing [14]. The use of memory resources is characterized by peak memory allocations, average memory working sets, and frequency of garbage collection operations; all these indicators reflect the effectiveness of memory usage and indicate unnecessary memory allocations. Usage of network bandwidth metrics measures the data traffic volume and allows communication pattern improvement.

7.3 Carbon Metrics

Carbon metrics convert energy use into environmental impact terms by taking into account the carbon intensity of the electricity that software systems use [9]. The carbon intensity, measured in grams of CO_2 equivalents per kilowatt-hour, varies significantly from one geographic region to another, as well as from one time to another, depending on the composition of energy sources utilized. The emissions per unit of work performed provide a basis for measuring the sustainability level of energy consumption, considering both the quantity and ecological footprint of energy production. Models that standardize the carbon footprint of software across platforms have been established to enable reporting on carbon impact [9, 17].

Category	Metric	Unit / Measure
Energy	Power per transaction	Joules / transaction
Energy	Energy usage per request	Joules / request
Resource utilization	CPU utilization	% of available cycles
Resource utilization	Memory usage	MB / peak working set
Resource utilization	Network bandwidth	GB / workload period
Carbon	Grid carbon intensity	gCO_2eq / kWh

Table 2: Sustainability Metrics Framework for Software Architecture [4, 11, 12, 13]

8. Case Studies and Empirical Findings

The recent development of empirical testing of the sustainability principles of architecture has been considerable, with many controlled tests and measurements in practice providing quantifiable proof of the sustainability benefits that result from certain architectural designs [1]. Research on agile-based architectural frameworks incorporating explicit sustainability checkpoints at iteration boundaries demonstrated measurable energy consumption reductions of approximately 10% compared with

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

conventional development approaches lacking structured sustainability consideration [1]. The result emphasizes the importance of integrating sustainability into the process level, as it indicates that architectural flexibility and layering techniques can improve the system's performance not just from a functional perspective but also from an environmental perspective when sustainability is recognized as a first-class architectural quality property [11]. Study of programming language and algorithm selection in large-scale data processing systems has revealed energy consumption disparities that significantly surpass those attainable through infrastructure-level optimization alone [3]. Comparative analysis across programming language implementations of equivalent computational tasks reports energy consumption variations of greater than an order of magnitude between the most and least efficient implementations, highlighting that software-level optimization is one of the most powerful tools in the sustainable architecture toolkit [3]. These results are consistent with extensive research demonstrating that the energy consumption of algorithmic and architectural choices predominantly influences software, rather than hardware specifications at comparable performance tiers. Analyses of serverless deployments in production computational environments have demonstrated idle energy elimination as a categorical benefit of the pattern, with active energy consumption depending strongly on implementation quality and invocation frequency [15]. Edge computing deployments in distributed data processing contexts show significant reductions in network-transmitted data volume compared to centralized processing architectures for sensor data aggregation tasks, with energy savings due to avoided wide-area network transmission [16]. Collectively, these empirical findings provide a quantitative foundation for architectural pattern selection decisions oriented toward sustainability objectives [10, 2].

9. Challenges in Sustainable Software Architecture

Despite the growing interest among scholars and professionals, the concept of sustainable software architecture is faced with a series of problems that hinder its implementation [14].

9.1 Lack of Standardized Metrics

The lack of universally used metrics and standardized methods for calculating sustainability continues to represent the largest obstacle standing in the way of progress in the area. There is significant heterogeneity across research groups in terms of methods of measuring power utilization, work load normalization, and system boundaries [12]. Efforts have been made to create some form of standardization, although the process seems to be disjointed, and tools used to calculate sustainability fall far behind the sophistication of regular performance profiling solutions [9].

9.2 Limited Tooling

There is very little built-in support for sustainability-related analysis in modern software development tools. There exist very few development, review, and modeling systems that incorporate any means of calculating or estimating energy consumption or greenhouse emissions [10]. This makes the process of analyzing software from this perspective one of personal responsibility, requiring a custom implementation of monitoring infrastructure, specific knowledge and skills, and time-consuming analysis processes [14]. Creating automated tools capable of measuring software's sustainability and integrating them into existing development processes would be a worthy undertaking for future research.

9.3 Trade-offs with Performance

Energy-efficient architectural designs frequently introduce trade-offs with conventional performance objectives, including latency, throughput, and availability [7]. Strategies such as workload batching to improve energy efficiency may increase response latency for interactive applications. The carbon-aware load balancing that involves deferring computation during periods when renewable energy is available

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

may not be compatible with the need for real-time processing. Such trade-offs must be carefully analyzed and settled by negotiations among stakeholders, requiring a more advanced approach to quality attribute selection than what is traditionally offered [6].

9.4 Developer Awareness

The empirical literature [14] has found that limited knowledge regarding sustainable development issues among developers is a crucial barrier. Empirical surveys of software engineers reveal that most software engineers do not consider the environmental impact when designing system architecture. They lack the means both theoretically and practically to assess whether or not their decisions can be sustainable [1]. This issue can only be addressed by spending more on sustainability-focused curriculum development in software engineering education and by providing incentives within organizations for using such a practice [8].

10. Future Research Directions

Sustainable software architecture is at a turning point, with several new trends set to increase its scope greatly, depth, and real-world impact over the next ten years [1, 10].

10.1 Carbon-Aware Cloud Computing

Carbon-aware cloud computing represents a paradigm in which computational infrastructure dynamically routes workloads to geographic regions or time windows characterized by lower grid carbon intensity [9]. Such an approach necessitates architectural capabilities for the migration of workloads, tolerance for delays in processing tasks, and real-time interoperability with carbon intensity data from the grid. Technical challenges in this domain encompass the design of optimization mechanisms for trading off between carbon reduction targets and the performance, economic, and data sovereignty factors, along with the definition of standardized interfaces for communicating carbon sensitivity to applications [9]. Empirical studies of carbon-aware scheduling approaches report carbon intensity reductions of up to 45% for batch workloads migrated to carbon-optimal scheduling regimes without performance degradation [9].

10.2 AI-Driven Energy Optimization

Artificial intelligence techniques show a lot of promise for dynamically optimizing workload placement, resource allocation, and infrastructure management in data centers [2]. Systems for optimizing using artificial intelligence are able to learn complex, nonlinear relations between features of the workload, the infrastructure, and energy consumption, which cannot be modeled by traditional systems using rule-based approaches [2]. Research challenges include ensuring that AI-driven optimization decisions are interpretable and reliable, controlling the energy used for AI model training and inference compared to the optimization benefits gained, and creating safety rules that prevent optimization decisions from violating service-level agreements or infrastructure stability requirements [18].

10.3 Sustainable AI Systems

The energy and carbon footprint of large-scale AI model training and inference has emerged as a significant and rapidly growing concern within the sustainable computing research community [19]. Training large-scale models can consume energy equivalent to substantial greenhouse gas emissions, which represents a direct architectural sustainability challenge determined by choices of model architecture, training data volume, computational hardware, and training strategy [19]. Research areas may involve creating energy-efficient architectures, using model compression/distillation for reducing inference energy usage, and incorporating sustainability criteria into AI models' design and evaluation [18, 13].

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

10.4 Circular Computing

In this area, circular economy principles are implemented in ICT infrastructure by extending the lifespan of hardware, reusing its parts, and minimizing e-waste generation [4]. From a software architecture perspective, circular computing needs to take hardware compatibility into account as an architectural constraint. This results in designs that are compatible with older hardware generations and reduce the frequency of hardware upgrades necessitated by increased software resource demands [13]. The study in this field includes investigations into architectural designs and patterns that allow software to evolve independently of increased dependency on hardware, thereby allowing software applications to continue running indefinitely on the current hardware infrastructure, resulting in less embodied carbon emissions due to the production and eventual destruction of hardware [4].

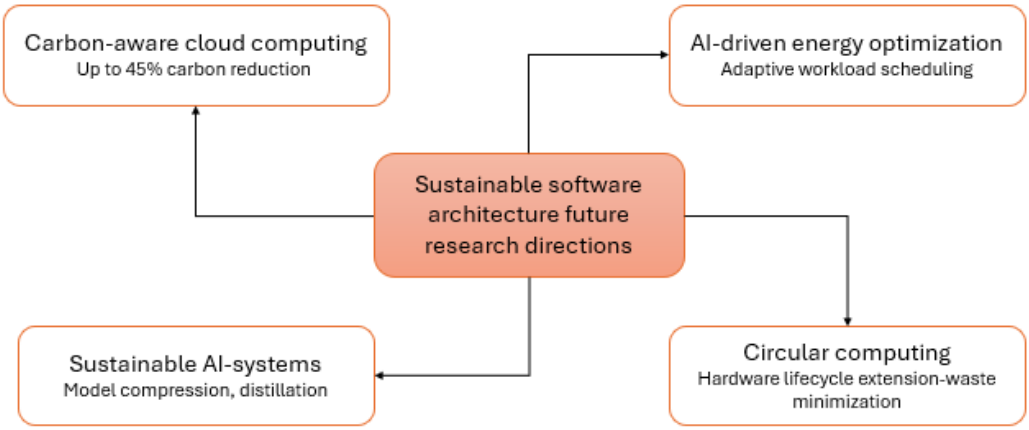

Figure 2: Emerging research directions in sustainable software architecture

Conclusion

Sustainable and green software architecture has matured from a peripheral concern of the software engineering discipline into a recognized and increasingly urgent field of research and practice. The environmental footprint of digital systems is no longer attributable solely to hardware infrastructure; software architectural decisions shape energy consumption, resource utilization, and carbon emissions in ways that are quantitatively significant and often more tractable than hardware-level interventions alone. In this essay, the theoretical underpinnings, guiding concepts, conceptual models, architectural designs, measurement approaches, and empirical findings have been discussed in an attempt to describe the present state of knowledge with regard to sustainable software architecture. The four-dimensional sustainability model, comprising environmental, economic, social, and technical sustainability, offers a very useful analysis model that allows one to look at energy and carbon issues from a larger perspective of sustainable systems design. It is pertinent to note here that this broad approach is necessary because of the fact that sustainable systems cannot be reduced to a single indicator or optimization goal. A system optimized for energy efficiency but economically unsustainable will be replaced prematurely; a system optimized for energy efficiency but technically rigid will be unable to incorporate future efficiency improvements as they emerge. Sustainability should be considered during system design and not added at a later point. There is plenty of evidence that the use of various patterns like microservices, serverless, edge, and data reduction within the domain of architecture can lead to some measurable value toward sustainability. Frameworks like GREENSOFT offer ways to systematically incorporate the principles of sustainable governance into software

engineering activities in an organization. All of these factors create a good foundation for sustainable architecture, although there are not many standardized metrics and fancy tools around yet. Persistent challenges, including the absence of standardized sustainability metrics, the immaturity of sustainability-oriented development tooling, complex trade-offs with performance objectives, and limited developer awareness, constrain the pace of adoption and the reproducibility of sustainability gains across organizations. This calls for collaborative investment in standardization efforts, tool creation, and education that reaches far beyond what single research efforts or individual institutions can achieve on their own. Cooperation between industrial consortia, standardization organizations, and research initiatives is required to establish the foundations that would enable sustainable software engineering. The following trends, including future research, carbon-aware computing, AI-enabled energy conservation, sustainable AI, and circular computing, represent an incredible future research agenda to investigate over the next decade. The software engineering community's ability to design, develop, and implement sustainable software will determine whether the ever-growing role of digital infrastructure leads to environmental problems or solutions. Sustainable software architecture should be designed as a true engineering discipline backed by standardized procedures and empirical research.

References

[1] Sandeep Kautish and Dipesh Gurung, "Advancing Sustainable Computing: A Systematic Literature Review of Software, Hardware, and Algorithmic Innovations," ICCK Transactions on Sustainable Computing, 2025. Available: https://www.icck.org/article/html/tsc.2025.767094

[2] Jianian Jin et al., "Green Software Engineering: A Study on Energy-Efficient Design and Deployment in Cloud Infrastructure," IEEE Access, 2025. Available: https://www.scirp.org/journal/paperinformation?paperid=144329

[3] Olivia Poy et al., "Impact on energy consumption of design patterns, code smells, and refactoring techniques: A systematic mapping study," Journal of Systems and Software, 2025. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121224003479

[4] Jakub Swacha, "Models of Sustainable Software: A Scoping Review," Sustainability, MDPI, 2022. Available: https://www.mdpi.com/2071-1050/14/1/551

[5] Patricia Lago et al., "Framing Sustainability as a Property of Software Quality," Communications of the ACM, 2015. Available: https://dl.acm.org/doi/epdf/10.1145/2714560

[6] Patricia Lago, "Architecture Design Decision Maps for Software Sustainability," International Conference on Software Engineering: Software Engineering in Society, 2019. Available: https://ieeexplore.ieee.org/document/8797634

[7] Carlos Paradis et al., "Architectural Tactics for Energy Efficiency: Review of the Literature and Research Roadmap," Hawaii International Conference on System Sciences, 2021. Available: https://pdfs.semanticscholar.org/6f33/a3778deb2a95fe08ec83cbd07b2f6ee31031.pdf

[8] Vasilios Andrikopoulos et al., "Sustainability in Software Architecture: A Systematic Mapping Study," arXiV, 2022. Available: https://arxiv.org/pdf/2204.11657

[9] Ana Radovanovic et al., "Carbon-Aware Computing for Datacenters," IEEE Transactions on Power Systems, 2022. Available: https://ieeexplore.ieee.org/document/9770383

[10] Sung Une Lee et al., "A survey of energy concerns for software engineering," Journal of Systems and Software, 2024. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121223003394?utm_source=chatgpt.com

[11] Roberto Verdecchia et al., "Green Software Engineering: Principles and Practices for Sustainable AI Development," Eco Tech Knowledge, 2023. Available: https://knowledge.les-enovateurs.com/resources/2023-08-15-green-software-engineering/

[12] Saurabh Kapoor, "Green Software Quality: A Comprehensive Framework for Sustainable Metrics in Software Development," International Journal of Computer Trends and Technology, 2024. Available: https://ijcttjournal.org/archives/ijctt-v72i10p118

[13] Loïc Lannelongue et al., "GREENER principles for environmentally sustainable computational science," Nature Computational Science, 2023. Available: https://doi.org/10.1038/s43588-023-00461-y

[14] Irene Manotas et al., "An Empirical Study of Practitioners' Perspectives on Green Software Engineering," International Conference on Software Engineering (ICSE), 2016. Available: https://dl.acm.org/doi/epdf/10.1145/2884781.2884810

[15] Simon Eismann et al., "A Review of Serverless Use Cases and their Characteristics," SPEC RG Cloud Working Group, 2021. Available: https://research.spec.org/fileadmin/user_upload/documents/rg_cloud/endorsed_publications/SPEC_RG_2020_Serverless_Usecases.pdf

[16] Weisong Shi et al., "Edge Computing: State-of-the-Art and Future Directions," Journal of Computer Research and Development, 2019. Available: https://www.researchgate.net/publication/330567277

[17] Ornela Danushi et al., "Carbon-Efficient Software Design and Development: A Systematic Literature Review," arXiV, 2025. Available: https://arxiv.org/pdf/2407.19901

[18] Roberto Verdecchia et al., "A Systematic Review of Green AI," WIREs Data Mining and Knowledge Discovery, 2022. Available: https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1507

[19] Emma Strubell et al., "Energy and Policy Considerations for Modern Deep Learning Research," Proceedings of the AAAI Conference on Artificial Intelligence, 2020. Available: https://doi.org/10.1609/aaai.v34i09.7123