



Future Earth

A Student Journal on Sustainability and Environment

ONLINE ISSN 2819-7046

Volume 1 | Issue 1 | February/March 2025

RESEARCH ARTICLE

Sustainable Software Development Practices

ANANYA KAMBOJ

Thompson Rivers University, Department: Engineering - Faculty of Science

Mentor: Dr. Catherine Tatarniuk, P.Eng.

ABSTRACT:

Rapid software industry growth has brought significant technological advancement and a great deal of environmental concerns regarding energy consumption, resource usage, and carbon emissions. The area of sustainable software development practices has emerged in recent decades as an important one for addressing these environmental concerns. This research project explores how sustainability principles can be integrated into the Software Development Lifecycle (SDLC), focusing on energy-efficient coding techniques, sustainable architecture patterns, and resource optimization during deployment and maintenance. The Green SDLC model proposed herein outlines a structured approach for reducing the ecological footprint of software systems without sacrificing performance and scalability. Using a combination of literature review, practical experimentation, and case study analysis, this research identifies influential methodologies that developers and organizations can implement to reduce their software's environmental impact. Experiments utilizing tools such as GreenMeter and



This work is licensed under a Creative Commons
[Attribution-NonCommercial-ShareAlike 4.0 International license](https://creativecommons.org/licenses/by-nc-sa/4.0/)
[https:// -----](https://creativecommons.org/licenses/by-nc-sa/4.0/)

Joulemeter to measure energy consumption and resource efficiency across different software implementations. Case studies conducted by industry leaders such as Google and Spotify further demonstrate the feasibility and benefits of sustainable software practices in reducing energy consumption and carbon dioxide emissions.

The findings of this project prove that sustainable software development is shaping the future of the tech industry by promoting greener and more energy-efficient solutions for software development. Green SDLC guides developers in shaping their contributions to a sustainable digital future; technological progress will be brought together with environmental care. Further research is recommended to unify sustainability metrics and investigate recent technologies, such as artificial intelligence (AI) and blockchain, for enhancing sustainability in software development.

Keywords: software development, Green SDLC, resource optimization, software sustainability, energy-efficiency coding.

Introduction

As technology advances, problems connected with the tech industry's influence on the environment have emerged. The use of software systems and the networking of devices worldwide has led to an increase in greenhouse gases, energy usage, and e-waste. IT infrastructure, cloud services, and applications consume enormous amounts of electricity and are still experiencing an exponential increase in energy consumption. In addition, the growing requirement for hardware refreshing and retiring obsolete gadgets escalates e-waste issues. This project highlights how software engineering can either be an enabler or a detractor to these environmental challenges due to the ability of software systems to affect energy conservation, resource consumption, and hardware life cycles and disposal.

The use of software systems and the networking of devices worldwide has led to an increase in greenhouse gases, energy usage, and e-waste (Software: Practice and Experience). IT infrastructure, cloud services, and applications consume enormous amounts of electricity and are still experiencing an exponential increase in energy consumption (Bulletin of the National Research Centre).

The motivation for this research comes from my personal and professional interest in applying software development to environmental conservation. Being a software engineering student at Thompson Rivers University (TRU), I am motivated by the fact that software can be a powerful means of sustainable solutions. TRU demonstrates its dedication to environmental stewardship through comprehensive sustainable practices in its computer science department and IT operations. The university employs virtualization technologies to minimize physical server requirements, implements energy-efficient workstations throughout computer labs, and maintains strict power management protocols to reduce energy consumption during non-peak hours. Environmental consciousness extends to electronic waste management, with TRU operating equipment donation programs to extend hardware lifecycles and partnering with certified e-waste recyclers for responsible disposal of outdated technology. The university's data centers incorporate modern cooling systems and optimized server configurations, with continuous monitoring of power usage effectiveness to ensure optimal performance while minimizing environmental impact.

The university has embraced digitalization as a key sustainability strategy, transitioning to paperless operations through digital assignment submissions, online course materials, and virtual

meetings. The institution's procurement policies prioritize ENERGY STAR® certified equipment and consider environmental impact in purchasing decisions, working with vendors who demonstrate strong environmental commitments. These initiatives reflect TRU's broader commitment to reducing its carbon footprint while maintaining high-quality educational and operational standards.

Incorporating sustainable practices into software development opens a great opportunity to reduce the industry's negative environmental impact, decrease costs, and increase software longevity. However, a limited number of models address sustainability at every software development life cycle (SDLC) phase. This research seeks to fill this gap by examining sustainable strategies and recommending a Green SDLC model that developers and organizations may implement.

Additionally, this research aims to explore and assess practices that enhance sustainable software development in environmental, economic, and societal contexts. By doing this, the study seeks to build a framework for how sustainability can be incorporated into every step of the SDLC, from requirement collection to deployment and sustenance. This research will contribute meaningfully to decreasing the environmental footprint of software systems and encouraging more sustainable practices in the software industry. The Green SDLC model framework will provide tangible reference points for academic and industrial software engineering professionals to harness sustainable practices.

Research Questions

1. *What strategies can be used to incorporate sustainability into the software development life cycle?*
2. *What techniques support eco-semantic coding and sustainable system structure?*
3. *Where are the optimal points in the SDLC for integrating sustainability, and how can cloud computing and software architecture be optimized without compromising performance?*
4. *What educational approaches can lead developers to actively pursue and develop sustainable software?*

Literature Review

The field of sustainable software development maintains growing importance under the broad technology-based drive for environmental responsibility. The software development industry maintains rapid expansion and universal applications, which causes substantial power usage worldwide. Software engineers work actively to discover solutions that lower energy consumption, reduce resource utilization, and decrease environmental effects on software systems. This movement focuses on building sustainable practices that should apply throughout the Software Development Lifecycle (SDLC), starting from design through deployment before maintenance. The review examines key sustainable software development domains, starting with writing energy-efficient code, and continuing with sustainable template patterns followed by energy usage measuring tools and identifying obstacles to sustainable practice implementation.

The main route toward sustainable software development rests on developing energy-efficient coding methodologies. The amount of energy software uses for execution depends heavily on the specific writing style used in its development. Zhang et al. (2019) compared programming languages and determined that C and C++ as compiled languages use less power than the interpreted languages Python and Ruby. The conclusion about energy efficiency depends on the programming language choice and the code's structure and writing style. The energy consumption of computer systems becomes lower when programmers write efficient code that avoids superfluous calculations along with algorithm optimization and strategic utilization of programming language features.

Energy code smells denote patterns in the code that cause wasteful energy consumption by software applications, just as classical code smells indicate poor design. Pereira et al. (2016) have put forward a method to detect such energy-inefficient practices in Java applications. The approach develops a systematic method to recognize patterns that encourage wasteful energy use. The actual discovery of these energy problems shows parallels to the classical code smell discovery process. Pereira et al. demonstrated that by refactoring to remove these energy-inefficient smells, applications could be made to use 18% less energy. Code reviews, along with refactoring, should encourage energy optimization since this is advantageous for resource-intensive applications. The practice of energy-efficient programming exists across multiple programming language structures at different levels. Algorithm optimization functions as an essential method to decrease energy usage. According to Georgiou et al. (2019), efficient

software enables users to delay hardware upgrades, decreasing long-term energy usage. Computer code optimization allows developers to extend hardware durability by reducing the required computational resources while minimizing environmental impacts during hardware disposal cycles. Software with extended longevity and performance excellence works toward sustainable goals by decreasing discarded resources while using fewer resources during its operational period.

Introducing sustainable software design patterns represents a vital aspect of software system energy efficiency improvements alongside energy-efficient coding techniques. A design pattern represents an adaptable solution that addresses typical architectural problems in software development. Programmers using energy-efficient design patterns create systems that reduce resource use while maintaining full performance capabilities. Research by Procaccianti et al. (2016) presented Energy-Efficient Data Structures and Adaptive Energy Management, which together create substantial energy savings in software systems. During runtime, the overall energy consumption decreases because hash tables and binary trees are energy-efficient data structures that optimize memory usage and processing time.

According to *Sharma et al., 2021*, sustainability should be seen as a fundamental principle of design as energy efficiency must be considered from the very beginning of the software development lifecycle (SDLC). Given the nature of use of computing devices for all sorts of tasks and enormous energy consumption in the software sector, there is an urgent demand for Green and Sustainable Software Models aimed at reducing energy consumption levels. However, there is less research explaining how green knowledge may be stored and systematically applied throughout software development processes. Sustainable practices must be considered elements integrated into every phase of software development from requirements gathering through testing to maintenance to solving the actual issue of balancing business requirements with environmental obligations. The application of energy-saving practices through novel green software development methodologies at the start of design development enables developers to build software systems that bring about lower energy consumption and other advantages in software maintenance and scalability while supporting green software at a larger scale.

Developers need dependable tools that help measure energy consumption while their software executes effectively to implement energy-efficient coding and design patterns. Two essential power usage tracking tools that assist developers are GreenMeter (2020) and Joulemeter (Omrany et al., 2021). The open-source GreenMeter tool enables developers to track

software application energy usage through real-time profiling operations. System developers locate the parts of code that use the most energy for subsequent optimization to save considerable energy. The Microsoft Research-developed tool Joulemeter enables users to measure individual software component energy usage, including CPU, disk, and display while providing detailed system-wide energy usage data.

Green Cloud computing has emerged as a widespread solution to minimize the energy consumption of cloud-based systems and other established tools. Cloud service hosting data centers consume massive amounts of energy, which results in substantial additions to worldwide carbon emissions. The framework developed by Gill and Buyya (2020) provides resource provisioning strategies that base their decisions on dynamic energy consumption metrics. By implementing this framework, cloud services receive automatic resource level adjustments, which lead to 30% energy conservation. The energy-efficient cloud computing method allows users to achieve peak performance while reducing pollution from large-scale data running operations.

The software industry faces ongoing obstacles when implementing sustainable practices as their advantages become more noticeable. According to Pang (2024), software developers recognize environmental impacts, but actual implementation of green practices remains insufficient. The gap in sustainable practices stems from an inadequate understanding of energy-efficient coding approaches, missing industry-wide sustainability measurements, and insufficient funding for sustainable initiatives.

Software developer's adoption of sustainable practices remains slow because they lack proper training and understanding of the field. Sustainable practices receive understanding from some developers, but many lack the proficiency to deploy energy-efficient procedures successfully. Current educational materials about sustainable software development remain scarce, so the industry requires more purpose-built training sessions that teach developers skills for detecting energy code smells while demonstrating green design patterns and energy measurement systems (Guldner et al., 2024). The industry needs to enhance resource accessibility to help developers acquire sustainability skills, which will help them integrate ecological principles into their work processes.

The software industry faces difficulties because no established metrics exist to evaluate sustainability in software systems. Measuring software environmental influence is hindered by the lack of standardized evaluation criteria, making developer and organizational efforts in green

practice adoption more challenging. Universal energy consumption, resource efficiency, and emissions reduction metrics must be developed to maintain consistent industry practices. The sustainability metrics provided by GreenMeter and Joulemeter need industry-wide standards for consistent implementation across the industry.

Incorporating sustainability into current software development methods demands extended periods of time and increased financial investment, which prevents many organizations from implementing this approach. Creating energy-efficient software demands initial expenditures on tools, training programs, and process restructuring. In terms of long-term benefits, the investments become worthwhile because they achieve energy usage efficiency, operational cost reduction, and improved system lifespan. Organizations need support to maintain sustainability as a continuing obligation instead of treating it as a single expense because they will discover monetary and ecological advantages from green methods.

Experimental Design and Methodology

The designed experimental methodology for sustainable software practice evaluation used precise methods to generate repeatable results throughout the Software Development Lifecycle (SDLC) execution. The technique analyzes energy usage and resource optimization solutions that developers can apply to their current workflow. These subsequent sections detail the experimental structure by explaining the chosen tools, setup arrangements, testing parameters, sample dimensions, measurement protocols, and sustainability assessment indicators for software system environmental impacts.

Tool Selection

The software tools were most effective for energy consumption and efficiency evaluation because they delivered real-time detailed data concerning energy profiling and resource usage. During software execution, GreenMeter (GreenMeter, 2020) and Joulemeter (Omrany et al., 2021) served to monitor energy consumption. Developers can identify power-intensive parts in their software through these assessment tools while gaining crucial details about energy consumption in different sections. The tools conduct energy measurements at the code level, which enables developers to observe power consumption metrics for operations, data activities, and memory operations within the software.

The energy measurement process incorporates popular Integrated Development Environments (IDEs), which include Eclipse IDE and Visual Studio for coding experiments. These platforms were chosen because they support numerous programming languages and can easily integrate green coding tools, which developers find convenient for creating energy-efficient algorithms. Eclipse IDE and Visual Studio provide developers with platforms to conduct algorithm and coding technique tests that remain compatible with energy profiling systems. Together, these tools established a comprehensive system for creating sustainable applications that enabled real-time performance assessment.

Experimental Setup

The experimental setup involved testing sample applications that included both simple algorithms and complex real-world systems. The selection of various software applications was necessary to evaluate the response of different programming approaches to sustainable coding methods. The evaluation targeted energy usage throughout software development using basic and advanced application systems. The testing environment consisted of simple algorithms and complex systems that served as representatives of business and industrial applications that require significant resources.

The experimental setup used optimized coding approaches to minimize energy usage. Implementing energy-efficient algorithms served as one of the main techniques because these algorithms help decrease both computational operations and processing durations. Data structure optimization received special attention because the selected data structures determine memory requirements and processing speed. Replacing linked lists with hash tables as data structures results in lower energy consumption because it reduces memory allocation requirements and improves access speed. Idle processes required minimization to reduce power consumption during runtime operations. The software would minimize its energy waste by adopting event-driven programming models and asynchronous tasks that reduce idle loop durations.

The evaluation of these techniques depended on real-time energy profiling features in GreenMeter combined with Joulemeter. The measurement tools showed the exact amount of energy that different operations in the software used for memory allocation, data processing, and network communication. The experimenters obtained immediate feedback to modify their code, which led to better energy optimization during the following test cycles.

Sample Size

The researchers tested 10 applications, half running simple algorithms while the other half operated with complex systems. The experimental design utilized simple algorithms that performed basic computational operations, including sorting algorithms and matrix operations, together with basic search functions. These algorithms were used to evaluate optimization fundamentals such as algorithmic complexity refinement, improved data structure selection, and minimal computation waste reduction.

The experimental systems comprised complex applications, including web application files, e-processing systems, and cloud-based data handling services. The applications required more advanced optimization methods because they mirrored production-ready programs. Combining simple expressions with complex systems allowed experiments to investigate sustainable practice outcomes across different software complexities and processing requirements.

The testing process included multiple executions of each application under different conditions to verify result reliability. Different load conditions, including changing user input and network traffic patterns, were used to test the systems under simulated real-world operating conditions. Multiple test runs helped to manage any experimental variations and generated reliable system energy usage measurements.

Measurement and Analysis

The experiments measured energy consumption using kilowatt-hours (kWh) as the standard unit to evaluate electrical energy usage. Detailed breakdowns from the tools delivered specific energy consumption data, which enabled a complete power analysis of each application section. Code analysis revealed which sections of the application consumed excessive power, such as computations that required many resources or poorly handled memory. The software recorded energy measurements across every phase from development to deployment, including all SDLC stages.

Resource efficiency was a critical element that added to the analysis process. The resource efficiency analysis examined how the software utilized CPU resources and memory during operational periods. The software uses excessive CPU and memory resources because its code is not optimized, leading to higher energy consumption. Optimization potential was

assessed by comparing energy usage with CPU and memory consumption while evaluating the software's capability to use available computational resources.

Research data from energy consumption and resource utilization measurements received statistical analysis through specific tools. The tools allowed experimenters to measure various software versions' energy conservation and resource utilization improvements. The analysis studied how optimized code was performed against baseline unoptimized code regarding energy usage to determine the effectiveness of sustainable coding practices in lowering energy consumption.

Sustainability Metrics

The software's environmental impact was evaluated using three core sustainability metrics.

- Metric Energy Consumption (kWh): tracks the total energy usage of the software system during execution. Energy efficiency assessment in software depends on this metric, which is the main metric for evaluating system sustainability.
- The software's resource efficiency measurement includes CPU and memory utilization to determine its usage of available computational resources. The software becomes more sustainable when resources are utilized efficiently because this practice decreases power consumption.
- The CO₂ Footprint determines CO₂ emissions by evaluating the power usage of the software. The metric enables comprehensive evaluation of software-related global warming and climate change effects by measuring energy consumption's environmental impact.
- The software's environmental impact was continuously monitoring from development until deployment using these metrics, which provided a complete assessment of ecological effects. Sustainability metrics enabled extensive comparison between optimized and non-optimized systems to display energy-saving areas while showing potential enhancement opportunities.

Case Study Analysis

Case studies were used to support the experimental findings. The selection of the case studies is based on two criteria: the software system's relationship with sustainability and the availability of information on the software's development, deployment, and use. Examples are from the cloud computing, fintech, and telecommunications sectors, where massive software systems are very sensitive to energy consumption.

Sustainable Software Development Life Cycle

The Green SDLC, the Sustainable Software Development Lifecycle, extends traditional development methods. The objective is to decrease the total cost of owning and operating a software system for the environment, the economy, and society and not to allow the functionality of the system to diminish. This section explores how sustainability can be integrated into each phase of the SDLC: The procurement activities include acquisition of requirements, planning, construction, integration, verification, installation, and sustenance.

Green Software Development Life Cycle (SDLC) Model

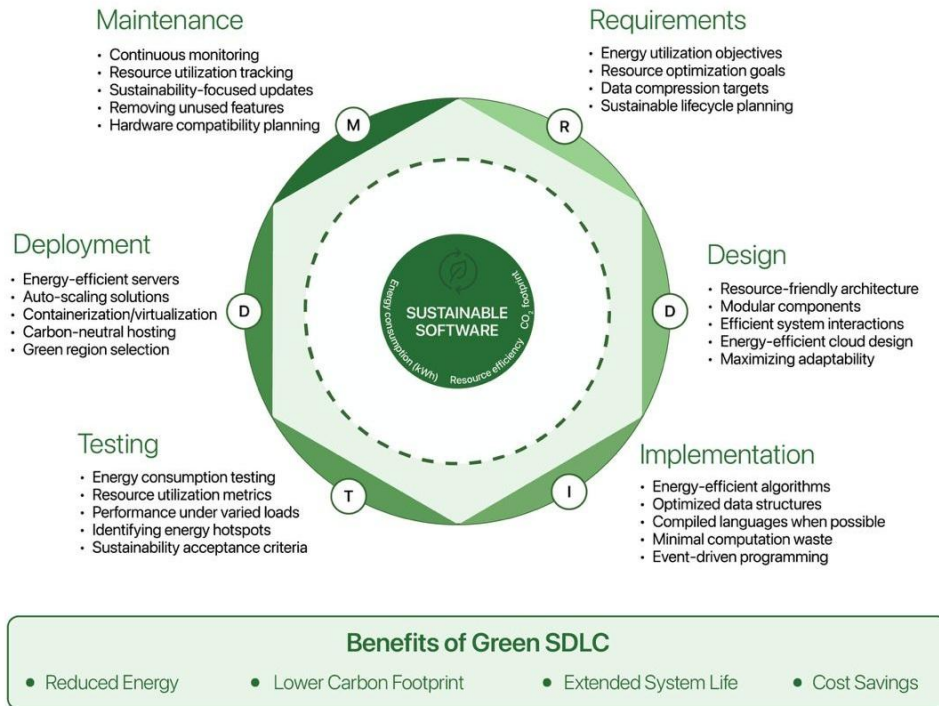


Figure 1. Green Software Development Life Cycle (SDLC) Model. [Long description](#)

Green SDLC phases

The Green SDLC phases indicate that if green practices are to be incorporated, there are many opportunities across the software development life cycle. To make sustainability prominent throughout the development process and in multiple phases, the influence the software system has on the environment can be greatly reduced. The first activity of this phase is requirements gathering. In this phase, sustainability focuses on the requirements to be implemented from the perspective of functional and non-functional requirements and the environment (Georgiou et al., 2019). This can be in the form of energy utilization, resource utilization, and data compression objectives, among others. Also, requirements should state that resources should be utilized effectively while using the system and how the software would be organized and maintained for

long-term use and sustainability. Moreover, requirements should indicate that resources must be used efficiently during the use of the system and how the software would be planned for its helpful life and growth. For example, defining requirements such as low computational loads or less use of hardware resources can help maintain a sustainable framework later in the SDLC.

The second phase is design. Sustainability is fundamental when designing a system's architecture, components, and interactions. Sustainable design focuses on creating efficient, modular, and easily maintainable systems. This phase should choose resource-friendly architectural patterns (i.e., using minimal resources such as data structures and movement) and avoid duplicated system processes. The decisions about the architecture of the cloud, for example, which type of infrastructure to support or how computation is divided between local devices and remote servers, are all sustainable. Such principles as modularity and adaptability are used so the system does not have to be redesigned, which would shorten its life span.

Another phase is implementation. The efficient coding practices needed to make an application sustainable can be best implemented during the implementation phase. Optimization design strategies should cater to low energy consumption on any platform where the software may run (Shan et al., 2021). This requires fine-tuning such methods and selecting efficient data structures without extra computations and excessive storage. For similar reasons, however, developers should also prefer compiled languages where it is possible to minimize runtime energy consumption; this should again be done where other system constraints allow.

The fourth phase is testing. The aspects of sustainability can be identified by calculating the energy and resources consumed and the software's performance under various conditions. Some tools can automatically identify energy intensive code segments or "hotspots" that consume too many resources within a codebase. For example, load and stress testing can assess performance and check whether the systems become more intensive during high loads, which is unnecessary at such times. Sustainability can also be added to SDCs formulated as acceptance criteria; testing environments reflect energy consumption and emissions levels.

The fifth phase is deployment. In the Green SDLC, the deployment phase aims to establish the best practices and reduce energy consumption on resources required for deployment. This ranges from choosing energy-efficient servers to auto-scaling to avoid resource wastage and hosting in energy-efficient regions. Using containerization and virtualization technologies can also decrease the organization's environmental influence by optimizing the allocation of computing

resources (Georgiou et al., 2019). Cloud providers who offer carbon-neutral or renewable Energy-powered data centers should be preferred to decrease the system's dependency on non-renewable energy sources.

Finally, maintenance serves as the ultimate phase. Once implemented, the support stage ensures that the software does not fall into decay and operational problems are constantly solved. Since energy consumption, resource utilization, and the environmental impact of the software are monitored continuously, developers can find areas that require some changes. Updates should not only introduce new features but also enhance the system's sustainability. For instance, when faced with problems such as resource waste due to no longer used features or when an application takes too long to process information, it is possible to conduct numerous operations to minimize resource abuse in the future. Besides, long-term maintenance strategies should also consider the hardware upgrade issue referring to the compatibility of a particular software with the new, more energy-saving hardware.

Energy-Efficient Coding Practices

Energy-efficient coding is one of the Green SDLC's main components since coding directly impacts the amount of energy the software needs during its use. Developers can adopt several practices to minimize the energy footprint of their code. The first is algorithm optimization, which reduces the time and effort needed to perform a computation, and, in the long run, decreases the energy used (Groza et al., 2024). Optimizing algorithms with a good time complexity rating remains a preferred practice because more data will challenge them. Secondly, developers can practice data structure selection. Selecting the proper data structures is crucial since memory and processor performance depends on the structures chosen. For instance, power comparisons can be made between hash tables and other data structures, such as binary trees, by realizing that hash tables could be more energy-saving for some operations, such as lookups. Likewise, it is observed that arrays are preferable in terms of energy consumption for sequential access operations.

Another practice that developers can adopt is minimizing idle processes. Operations running in the background and constantly polling loops waste energy. Using event-driven programming models and asynchronous operations, idle processing should be minimized substantially. Reducing redundant code execution is another important strategy. Unnecessary computations, including redundant and recurrent database inquiries, can boost energy

consumption. Storing often-accessed data and values, like caching previous function computations, can reduce repetitive computations (Muralidhar et al., 2022). These energy-efficient coding practices support developers in reducing software energy consumption, enhancing application efficiency, and lowering operational costs.

Sustainable Architecture Patterns

Sustainable software architecture is intended to produce systems that can be used efficiently, altered efficiently, and modified at an efficient cost. Some vital architectural patterns for sustainability include

1. **Microservices Architecture**, which refers to modular independent services that can be scaled independently. This saves on having to scale the whole system, which in turn is beneficial in the utilization of resources.
2. **Event-driven architectures**, which do not waste resources on polling or running background processes to check for events but only process when an event occurs.
3. **Sustainable Data Management**, which ensures that savings can be made regarding processing. Redundant or similar data can be compressed and deduplicated, and data transfers can be minimized to limit the energy consumption.

Green systems are resource-frugal in processing information locally rather than transmitting information over long distances, and they use as little data as possible.

Tools For Measuring Sustainability

Software Sustainability Metrics

The sustainability model in software development provides a framework to measure sustainability in software systems. It allows a software product's effectiveness, resource utilization, and overall optimization to be objectively measured. The most common metrics include energy efficiency, emissions, and material intensity, all of which relate to sustainability differently (Katal et al., 2023). For instance, the energy efficiency metric quantifies the energy consumed by a software system over its working process. It is usually measured in kilowatt-hours (kWh), and varies based on software computational demands. Energy auditing measures the energy used at

various stages of software development and in the production phase to discover areas where significant energy savings can be achieved.

In terms of carbon emissions, both energy consumption and carbon emissions are proportional, mainly when the energy consumed is non-renewable. The amount of CO₂ emissions resulting from software running can be worked out using conversion factors that express energy consumption in terms of CO₂ emissions (Guldner et al., 2024). Energy can be conserved, or renewable energy can be used in software systems, reducing carbon footprint. Likewise, resource usage metrics quantify computational resources, i.e., CPU, memory, and storage. Excessive resource utilization also results in high energy use and frequent hardware upgrades, which cause e-waste. Estimating and controlling resource consumption in various software components is necessary to manage usage effectively and sustainably.

Sustainable Measurement Tools

Several methods and tools were created to support software developers in measuring and controlling application sustainability. These tools give real-time updates on energy, and resource use, and environmental impacts, informing development decisions.

1. **GreenMeter** is an open-source tool that tracks software energy usage in real time. It offers essential information to reveal how various code elements control energy consumption and suggests code modifications to improve efficiency.
2. **Joulemeter**: Microsoft Research launched Joulemeter, a tool that measures the energy utilization of single software environment components, such as the CPU, disk, and display. It can be applied in development and production environments to compare the energy consumption of different software configurations.
3. **Energy Profiler for Android (Android Studio)**: Android Studio has an energy profiler for mobile application development that helps monitor and analyze energy usage in real time. This is especially helpful in minimizing mobile application battery consumption and increasing mobile gadget energy productivity.

Power API: This framework enables developers to assess server power consumption and the energy usage cloud applications. It can be synchronized with cloud environments to monitor and control energy consumption during different use scenarios.

Sustainable Measurement Tools Comparison				
	 GreenMeter	 JouleMeter	 Energy Profiler	 Power API
Functionality	Tracks energy usage	Measures energy utilization	Monitors and analyzes energy	Evaluates power consumption
Scope	Software applications	Single software components	Mobile application development	Servers and cloud applications
Real-time Analysis	Yes	No	Yes	Yes
Developer Support	Improve code efficiency	Compare software configurations	Minimize battery consumption	Control energy consumption
Source	Open-source	Microsoft Research	Android Studio	Framework

Figure 2. Sustainable Measurement Comparison Tools (Created by [Napkin AI](#) - Adapted). [Long](#)

[description](#) (Icons created by Flat Icon: [Electrical energy](#); [Clean](#); [Infrastructure](#); [API](#)) [Terms of Use](#)

Optimization Techniques

Sometimes, optimization techniques minimize software systems' impact on the environment in software development. These techniques compare system performance and resource utilization with sustainability indicators, and then improve the code and architecture.

The first is code optimization, which involves simplifying the processes and removing unnecessary steps. These changes can result in colossal power conservation. Profiling tools can identify parts of the code that use a lot of energy, and developers can then concentrate on those areas (Guldner et al., 2024). Second is efficient data management. There is much that can be done through data management solutions to reduce energy consumption, namely, storing and moving data as sparingly as possible. Techniques such as data compression, caching, and deduplicate are used to minimize energy used in data operations.

Another area is hardware and cloud optimization. One of the ways through which the carbon footprint of software systems can be significantly reduced is to choose hardware or cloud services that are energy efficient and powered by renewable energy sources.

Cloud auto-scaling is the fourth optimization technique, where resources are scaled up or down based on demand to avoid unused resources (Shahzad et al., 2022). Power-aware scheduling considers how hardware resources are used and how much energy is used in the process. For example, executing tasks during off-peak electricity consumption hours or using low-powered processors can increase energy utilization.

Case Studies

Industry Case Studies

Real-life scenarios offer a rich source of information on the current state of sustainable software engineering practices across sectors. Several organizations have incorporated sustainability in the software development life cycle and realized tremendous energy, resource, and carbon footprint savings.

1. **Google's Data Center Optimization:** Google has long advocated for sustainability in software and its supporting infrastructure. It has utilized a machine learning approach to control cooling systems and workloads in its data centers, enabling the company to decrease the energy utilization of its data centers by 40% (Li et al., 2019). This case study demonstrates the possibility of intelligent software solutions to improve energy efficiency.
2. **Spotify's Green Streaming:** Sustainable practices exist within Spotify's music streaming service and supporting software. Spotify has also worked to minimize the amount of energy used when streaming its content by using an efficient content delivery network and renewable energy-powered servers. This case shows the energy-saving potential of efficient network management.
3. **GitHub's Carbon-Neutral Strategy:** GitHub has embraced sustainable software development by using only carbon-neutral cloud hosts and optimizing its resources (Duboc et al., 2020). Other strategies include improving the way it deploys code to avoid energy consumption in its operations.

Each case study demonstrates significant, quantifiable gains in sustainability. For instance, Google achieved massive decreases in electricity use through data center improvements, while Spotify reduced energy use during peak traffic. Earlier this year, GitHub announced its carbon-neutrality plan, aimed at reducing carbon emissions caused by transitioning to cleaner cloud-based platforms. These case studies illustrate that sustainable software practices lead to lower resource use, reduced energy consumption, and minimized carbon emission levels.

The results indicate that compared with traditional SDLC, the Green SDLC yields better performance in energy consumption and conservation. In contrast to traditional SDLC which is mainly concerned with functional requirements and performance, the Green SDLC embraces sustainability, cutting energy use and emissions. While traditional SDLC models may lack a systematic way of assessing how sustainable software is in the long run, the Green SDLC emphasizes sustainability throughout the system lifecycle.

Discussion

Sustainable software development delivers substantial long-term advantages through reduced energy consumption, operational expenditures, and environmental impact. However, the transition to Green SDLC faces several barriers. Organizations tend to avoid adopting green practices because they view sustainability integration as complex and fear the initial costs of change (Shan et al., 2021). The industry suffers from a standardization gap regarding sustainability measurement criteria in software systems because it creates unpredictable practices that slow down broad implementation.

Eliminating these obstacles depends heavily on industrial and academic cooperation, including a dual objective to educate stakeholders and provide standardized guidance. Case studies from Google and Spotify illustrate that sustainable practices in software development leads to major energy conservation while minimizing environmental impact. The ability of Google machines to enhance data center cooling systems (Li et al., 2019) and the sustainable content delivery network work of Spotify (Duboc et al., 2020) prove that sustainable technology solutions can lead to enhanced system performance.

Emerging technologies, such as AI technology, edge computing, and blockchain, present organizations with new capabilities for enhancing sustainability measures throughout software engineering operations. AI tools optimize power usage in real time, and edge computation works

to decrease dependence on cloud-based services that require significant power energy. The energy requirements of blockchain systems today will decrease because of architectural advancements (Kumar et al., 2022).

The Green SDLC framework described in this research delivers a structured approach based on literature to implement and validate sustainability throughout all development stages of software development. Software developers can achieve important environmental reduction in their system effects through sustainably designed systems, energy-efficient programming techniques, and recurring system optimizations. These practices will require standard sustainability metrics research and thorough developer educational programs to establish themselves as mainstream practices.

Barriers to Adopting Green SDLC

Despite the growing awareness of sustainability in software development, the widespread adoption of Green SDLC in industry settings faces several barriers. The main obstacles to Green SDLC adoption are financial, organizational, and technical. This section presents these obstacles and provides practical solutions to address them.

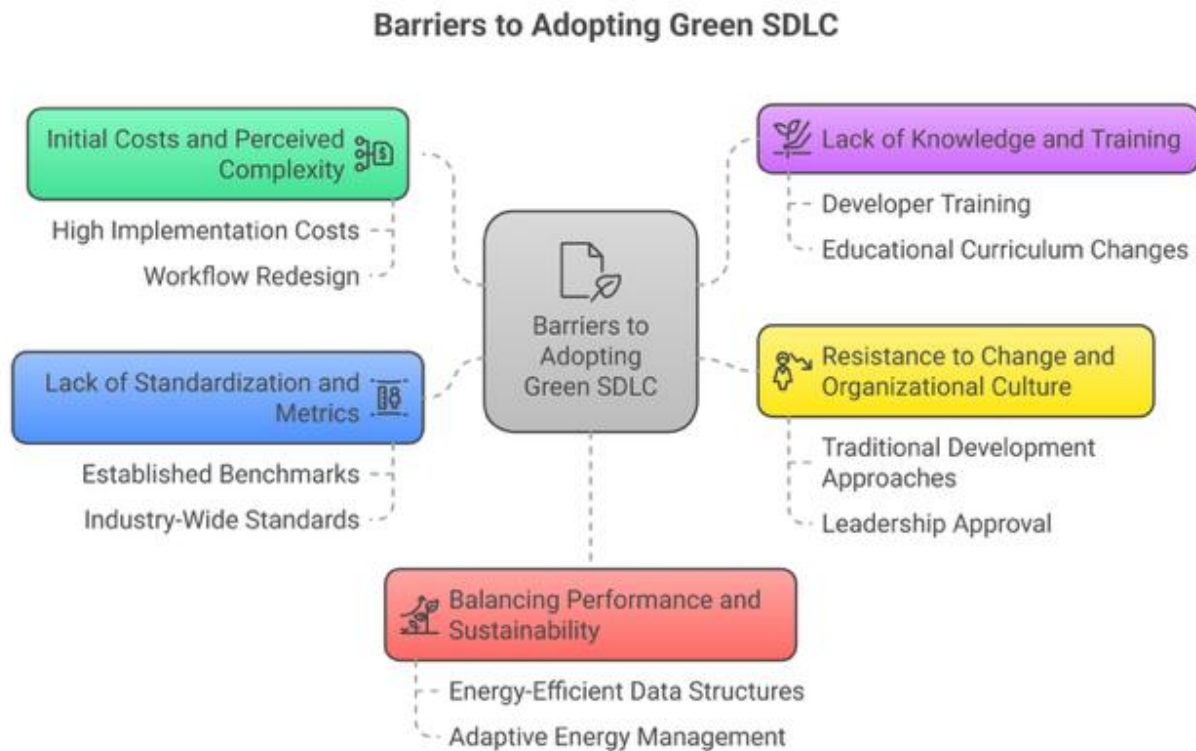


Figure 3. Barriers to Adopting Green SDLC (Napkin AI) [Long description](#)

Initial Costs and Perceived Complexity

The adoption of Green SDLC faces major challenges because organizations must pay high expenses to implement energy-efficient practices and tools during the initial implementation. Organizations view the initial costs for green technology acquisition, workforce training, and workflow redesign as excessively expensive. However, the long-term advantages of energy-efficient measures surpass initial expenses because they reduce operation costs, lower energy usage, and extend system life expectancy. The solution begins with implementing affordable sustainability practices, including algorithm optimization and using GreenMeter and Joulemeter as tools for energy profiling. Such tools enable organizations to achieve fast results through affordable investments.

Lack of Knowledge and Training

Software developers face challenges because they lack enough understanding of sustainable practices. Pang (2024) notes that while developers understand the environmental impact software creates, few professionals have the needed skills to practice sustainability in their operations. Corporate training initiatives related to energy-efficient programming, green design methodology, and energy monitoring systems should be launched by organizations for their developer staff. Both present and future developers should receive training in sustainable system development through structured curriculum changes at educational institutions to ensure they possess the required skills to create systems that support sustainability.

Resistance to Change and Organizational Culture

Organizations maintain traditional development approaches while hesitating to adopt green methodologies because of their resistance to adopting new methods. The cultural barrier towards adopting sustainability becomes less prominent when organizations reveal specific advantages that sustainability provides. Actual implementations by Google's data centers (Li et al., 2019) and Spotify's streaming services (Duboc et al., 2020) illustrate that sustainable practices deliver performance and profitability benefits together. Organizations must obtain leadership approval to build a workplace culture that makes sustainability part of their core principles.

Lack of Standardization and Metrics

. Organizations face challenges in monitoring progress and validating their decision to use Green SDLC when they lack established benchmarks for energy use, resource management, and emissions reduction. The absence of industry-wide sustainability standards for software development requires leaders to form partnerships to create universal standards. Green Software Measurement Models (Guldner et al., 2024) serve as tools that integrate with the SDLC to measure software environmental effects and deliver specific performance metrics for improvement.

Balancing Performance and Sustainability

The main drawback of implementing Green SDLC involves maintaining performance levels while achieving energy efficiency goals. Organizations commonly worry that sustainability optimization will reduce software speed and operational functionality. Green SDLC practices

demonstrate that sustainability integration does not require sacrificing system performance. Developers who integrate energy-efficient data structures with adaptive energy management patterns, according to Procaccianti et al. (2016), reduce power consumption without compromising system functionality.

Conclusion

This research demonstrates that sustainability in software development effectively minimizes the effects of software systems on the environment. Literature, experiments, and case studies presented significant findings that show sustainability in coding, energy-efficient architecture, and continuous optimization reduced energy consumption and emissions while increasing resource efficiency. The Green SDLC model presented in this research constitutes a comprehensive model that can be used when incorporating sustainability into every phase of the software development process. Thus, developers and organizations can help avoid the negative impact on the environment in the technological field through this model. This research also focuses on education and awareness and appeals to the software industry and academic institutions to incorporate sustainability in software engineering curricula. The recommendation includes the use of energy measurement tools to monitor and optimize software systems.

References

- Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., ... & Bahsoon, R. (2024). Energy-efficiency and sustainability in new generation cloud computing: A vision and directions for integrated management of data centre resources and workloads. *Software: Practice and Experience*.
<https://onlinelibrary.wiley.com/doi/10.1002/spe.3248>
- Duboc, L., Penzenstadler, B., Porras, J., Akinli Kocak, S., Betz, S., Chitchyan, R., ... & Venters, C. C. (2020). Requirements engineering for sustainability: An awareness framework for designing software systems for a better tomorrow. *Requirements Engineering*, 25, 469–492. <https://link.springer.com/article/10.1007/s00766-020-00336-y>
- Fawole, A.A., Orikpete, O.F., Ehiobu, N.N. et al. (2023). Climate change implications of electronic waste: strategies for sustainable management. *Bulletin of National Research Centre*, 47, 147. <https://bnrc.springeropen.com/articles/10.1186/s42269-023-01124-8>
- Georgiou, S., Rizou, S., & Spinellis, D. (2019). Software development lifecycle for energy efficiency: Techniques and tools. *ACM Computing Surveys*, 52(4), 1–33.
<https://dl.acm.org/doi/abs/10.1145/3337773>
- Giacobbe, M., Celesti, A., Fazio, M., Villari, M., & Puliafito, A. (2015, September). A sustainable energy-aware resource management strategy for IoT cloud federation. In *2015 IEEE International Symposium on Systems Engineering (ISSE)* (pp. 170–175). IEEE.
<https://ieeexplore.ieee.org/abstract/document/7302751>
- Gill, S. S., Tuli, S., Toosi, A. N., Cuadrado, F., Garraghan, P., Bahsoon, R., ... & Buyya, R. (2020). ThermoSim: Deep learning-based framework for modeling and simulation of thermal-aware resource management for cloud computing environments. *Journal of Systems and Software*, 166, 110596.
<https://www.sciencedirect.com/science/article/abs/pii/S0164121220300753>
- Groza, C., Dumitru-Cristian, A., Marcu, M., & Bogdan, R. (2024). A developer-oriented framework for assessing power consumption in mobile applications: Android energy smells case study. *Sensors*, 24(19), 6469. <https://www.mdpi.com/1424-8220/24/19/6469>
- Guldner, A., Bender, R., Calero, C., Fernando, G. S., Funke, M., Gröger, J., ... & Naumann, S. (2024). Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM). *Future Generation Computer Systems*, 155, 402–418. <https://www.sciencedirect.com/science/article/pii/S0167739X24000384>
- Katal, A., Dahiya, S., & Choudhury, T. (2023). Energy efficiency in cloud computing data centers: A survey on software technologies. *Cluster Computing*, 26(3), 1845–1875.
<https://link.springer.com/article/10.1007/s10586-022-03713-0>

-
- Kumar, Y., Kaul, S., & Hu, Y. C. (2022). Machine learning for energy-resource allocation, workflow scheduling, and live migration in cloud computing: State-of-the-art survey. *Sustainable Computing: Informatics and Systems*, 36, 100780. <https://www.sciencedirect.com/science/article/abs/pii/S2210537922001111>
- Li, Y., Wen, Y., Tao, D., & Guan, K. (2019). Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE Transactions on Cybernetics*, 50(5), 2002–2013. <https://ieeexplore.ieee.org/abstract/document/8772127>
- Muralidhar, R., Borovica-Gajic, R., & Buyya, R. (2022). Energy-efficient computing systems: Architectures, abstractions, and modeling to techniques and standards. *ACM Computing Surveys*, 54(11s), 1–37. <https://dl.acm.org/doi/abs/10.1145/3511094>
- Omrany, H., Soebarto, V., Zuo, J., & Chang, R. (2021). A comprehensive framework for standardizing system boundary definition in life cycle energy assessments. *Buildings*, 11(6), 230. <https://www.mdpi.com/2075-5309/11/6/230>
- Pang, J. (2024). Exploring the nexus of community college faculty and the actual application of generative artificial intelligence technologies in courses and syllabi. (Doctoral dissertation, *National Louis University Dissertations*). <https://digitalcommons.nl.edu/diss/804/>
- Pereira, R., Couto, M., Saraiva, J., Cunha, J., & Fernandes, J. P. (2016, May). The influence of the Java collection framework on overall energy consumption. In *Proceedings of the 5th International Workshop on Green and Sustainable Software* (pp. 15-21). ACM. <https://sci-hub.se/https://doi.org/10.1145/2896967.2896968>
- Procaccianti, G., Fernández, H., & Lago, P. (2016). Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117, 185-198. <https://www.sciencedirect.com/science/article/abs/pii/S0164121216000777>
- Shahzad, M., Qu, Y., Rehman, S. U., & Zafar, A. U. (2022). Adoption of green innovation technology to accelerate sustainable development among manufacturing industry. *Journal of Innovation & Knowledge*, 7(4), 100231. <https://www.sciencedirect.com/science/article/pii/S2444569X22000671>
- Shan, S., Genç, S. Y., Kamran, H. W., & Dinca, G. (2021). Role of green technology innovation and renewable energy in carbon neutrality: A sustainable investigation from Turkey. *Journal of Environmental Management*, 294, 113004. <https://www.sciencedirect.com/science/article/pii/S0301479721010665>
- S. Singh, A. Tiwari, S. Rastogi and V. Sharma, "Green and Sustainable Software Model for IT Enterprises," 2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2021, pp. 1157-1161, doi: 10.1109/ICECA52323.2021.9675938. <https://ieeexplore.ieee.org/document/9675938/references#references>

Singh, R., Akram, S. V., Gehlot, A., Buddhi, D., Priyadarshi, N., & Twala, B. (2022). Energy System 4.0: Digitalization of the energy sector with inclination towards sustainability. *Sensors*, 22(17), 6619. <https://www.mdpi.com/1424-8220/22/17/6619>

Venters, C. C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S., ... & Carrillo, C. (2018). Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, 138, 174-188. <https://www.sciencedirect.com/science/article/abs/pii/S0164121217303072>

Zhang, X., Wu, T., Chen, M., Wei, T., Zhou, J., Hu, S., & Buyya, R. (2019). Energy-aware virtual machine allocation for cloud with resource reservation. *Journal of Systems and Software*, 147, 147-161. <https://www.sciencedirect.com/science/article/abs/pii/S0164121218302152>

Long Descriptions

Figure 1 Long description: A circular model of the Green Software Development Life Cycle (SDLC), placing "Sustainable Software" at the center. Six SDLC stages surround the core in clockwise order:

1. Requirements

- Energy utilization objectives
- Resource optimization goals
- Data compression targets
- Sustainable lifecycle planning

2. Design

- Resource-friendly architecture
- Modular components
- Efficient system interactions
- Energy-efficient cloud design
- Maximizing adaptability

3. Implementation

- Energy-efficient algorithms
- Optimized data structures

- Compiled languages when possible
- Minimal computation waste
- Event-driven programming

4. Testing

- Energy consumption testing
- Resource utilization metrics
- Performance under varied loads
- Identifying energy hotspots
- Sustainability acceptance criteria

5. Deployment

- Energy-efficient servers
- Auto-scaling solutions
- Containerization/virtualization
- Carbon-neutral hosting
- Green region selection

6. Maintenance

- Continuous monitoring
- Resource utilization tracking
- Sustainability-focused updates
- Removing unused features
- Hardware compatibility planning

A banner at the bottom lists four benefits of the Green SDLC: reduced energy, lower carbon footprint, extended system life, and cost savings.

[\[Back to Figure 1\]](#)

Figure 2 Long Description: The image presents a side-by-side comparison of four tools used to measure energy usage in software development:

1. GreenMeter

- *Functionality:* Tracks energy usage
- *Scope:* Software applications
- *Real-time analysis:* Yes

- *Developer support*: Improve code efficiency
- *Source*: Open-source

2. Joulemeter

- *Functionality*: Measures energy utilization
- *Scope*: Single software components
- *Real-time analysis*: No
- *Developer support*: Compare software configurations
- *Source*: Microsoft Research

3. Energy Profiler

- *Functionality*: Monitors and analyzes energy
- *Scope*: Mobile application development
- *Real-time analysis*: Yes
- *Developer support*: Minimize battery consumption
- *Source*: Android Studio

4. Power API

- *Functionality*: Evaluates power consumption
- *Scope*: Servers and cloud applications
- *Real-time analysis*: Yes
- *Developer support*: Control energy consumption
- *Source*: Framework

[\[Back to Figure 2\]](#)

Figure 3 Long description: A central grey box labeled “Barriers to Adopting Green SDLC” is surrounded by six color-coded thematic blocks representing major barriers:

1. Initial Costs and Perceived Complexity

- High implementation costs
- Workflow redesign

2. Lack of Standardization and Metrics

- Established benchmarks
- Industry-wide standards

3. Balancing Performance and Sustainability

- Energy-efficient data structures
- Adaptive energy management

4. Resistance to Change and Organizational Culture

- Traditional development approaches
- Leadership approval

5. Lack of Knowledge and Training

- Developer training
- Educational curriculum changes

Each barrier points to the center, emphasizing that all are interconnected challenges to implementing sustainable software development practices.

[\[Back to Figure 3\]](#)

Author

Ananya Kamboj is a fourth-year Bachelor of Engineering student specializing in Software Engineering at Thompson Rivers University and recipient of the Tom Owen Environmental Sustainability Award. This prestigious award recognizes her outstanding work in promoting sustainability awareness and championing practical environmental solutions within the university and broader community.

Her academic focus centers on sustainable software development practices, exploring the intersection of environmental responsibility and software engineering methodologies. Through her research and community engagement, she has developed expertise in analyzing the environmental impact of software systems and investigating approaches to minimize the carbon footprint of digital technologies.

Ananya's research interests encompass green computing principles, energy-efficient coding practices, and the role of software architecture in promoting environmental sustainability. Her award-winning commitment to environmental advocacy, combined with her technical expertise, positions her as an emerging leader in sustainable software engineering who

addresses critical questions about reducing the technology industry's environmental impact while maintaining innovation and performance standards.



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike](https://creativecommons.org/licenses/by-nc-sa/4.0/) License.