Sustainable Software Engineering

Ana Mako, Ada Turgut, Sahana Ganesh, Valantis Andreas, Wojciech Mundala

Group 6

1 Introduction

Today, nearly two thirds of the global population are connected to the internet through Information and Communication Technology (ICT) [1]. This widespread reliance on digital technologies has environmental consequences: the ICT sector's greenhouse gas (GHG) emissions are now estimated to be comparable to those of the aviation industry, representing roughly 1.5-4% of global emissions as of 2024 [2]. Software engineering, as a key enabler of the ICT sector, is therefore at the forefront of this issue and plays a critical role in shaping the industry's response. Despite this responsibility, there is currently no widely established systematic approach for incorporating sustainability into the software development lifecycle [3].

The absence of a systematic approach to sustainability in software development limits the sustainability literacy among practitioners in the field. Although developers are often interested in learning more about sustainable software practices, many lack expertise and awareness regarding the energy consumption and environmental impact of their code [4]. This knowledge gap prevents developers from effectively applying sustainability principles in practice and limits their ability to meaningfully address the problem [4].

Several tools have been developed to encourage the adoption of energy-efficient coding practices in software development, such as the Green Code Initiative [5] and Creedengo JavaScript [6]. However, these tools typically focus on specific platforms or domains and address energy efficiency at a relatively low technical level. As a result, there remains a lack of general-purpose tools that promote sustainability literacy among developers at a higher level of abstraction. This is the case even though energy usage concerns frequently surround architectural and design decisions [4].

In addition, while existing resources such as sustainability pattern catalogs and educational initiatives (e.g., Green Software Foundation training [7] or W3C guidelines [8]) provide valuable knowledge, they are typically consumed as standalone materials. Developers must actively seek, interpret, and remember these recommendations, which creates a disconnect between knowledge acquisition and practical application. As a result, sustainability considerations are often overlooked during everyday development tasks, where time constraints and cognitive load dominate decision-making.

In contrast, there is limited support for integrating sustainability guidance directly into the developer's workflow in a lightweight and continuous manner. This highlights the need for tools that not only inform but also actively reinforce sustainable thinking during routine development activities.

To address this gap in developer support, we introduce GreenTips, a cross-platform command-line interface (CLI) that provides developers with daily suggestions of general-purpose, sustainability-oriented design patterns, along with additional resources for further learning. The tool aims to communicate sustainability principles in software development through a digestible, high-level format that supports the development of sustainability literacy among developers without imposing significant additional time demands.

We outline the tool by first defining motivation and context in section 2, then summarize related work in

section 3. We follow this with our solution outline in section 4 and go through our steps of evaluation in section 5. Lastly, we put our findings in a bigger context in section 6 and discuss future implications in section 7.

2 Motivation and Context

In this section, we will discuss why sustainability and energy consumption play such major roles in today's software engineering practices.

Sustainable Software Engineering (SSE) is a developing field that shifts the focus from hardware efficiency to the environmental impact of software design and use [9]. As global data center energy consumption increases, "software-aware engineering" has become more important, requiring developers to consider energy as a primary constraint when developing software [10].

Modern SSE includes the concept of *carbon intensity*, where software is designed to respond to the real-time availability of renewable energy in the grid [11]. With this in mind, let us analyze different areas of energy consumption across SE domain and energy-aware practices that address this problem.

2.1 The Growing Footprint of the ICT Sector

The Information and Communication Technology (ICT) sector represents a significant and growing portion of global greenhouse gas (GHG) emissions [12]. Recent estimates also suggest that, in the absence of significant measures of energy-sobriety, this could have risen to more than 7% by 2025 [13]. Data centers, network grids and user devices account for a substantial portion of the sector's impact, with emissions generated during manufacturing making up to 50% of their total carbon footprint [14].

2.2 Energy-Aware Software Engineering Practices

Engineers adopting energy as a main architectural constraint can achieve significant efficiency gains through the following areas:

2.2.1 Algorithmic Efficiency and Energy Complexity

Sustainable engineering extends traditional computer science metrics, like time (O) and space complexity, by introducing energy complexity (E). Empirical data suggests that for sequential tasks, such as sorting in Java, around 94% of energy consumption variance correlates directly with time complexity [15]. However, this relationship breaks down in parallel or memory-intensive environments. In these scenarios, the energy cost of data movement and cache misses can make time-optimal algorithms less energy-efficient than those optimized for memory access patterns [16].

2.2.2 Data and Network Minimization

In addition, the movement of data over a network is a significant power draw in modern software, especially for mobile and cloud systems, where keeping the hardware active to process data transfers uses a lot of energy. In order to address this, developers are shifting to more compact options like Protobuf instead of JSON, to keep the amount of data sent as small as possible [11, 17]. Also, "carbon-aware" networking is used by timing non-urgent data updates for when the local power grid is being supplied by clean/renewable energy [11].

2.2.3 Sustainable Architectural Patterns

Another area where sustainable practices are vital is software design, where "Green Patterns" can reduce computational overhead. While research in C++ illustrates that the indirection required for patterns such as the *Visitor Pattern* can increase energy usage [18], many impactful energy-saving choices are found in specific implementation patterns. For instance, replacing inefficient list membership checks inside loops (which often result in $O(n^2)$ complexity) or avoiding unbounded file reads that load entire datasets into memory are key strategies for reducing resource waste. Consequently, sustainable architecture prioritizes strategies like aggressive caching

and lazy loading to isolate and optimize energy-heavy tasks [17, 11].

2.3 Benefits of Energy-Aware Practices

Adopting energy-aware software engineering practices offers significant benefits beyond environmental sustainability for various stakeholders. Organizations that successfully utilize green IT can reduce energy costs by 17% to 90% [20]. Developers benefit from "cleaner" codebases that are more efficient and require less maintenance, thus freeing up time for innovation and increasing job satisfaction rates [20].

Furthermore, software developed by following energy-efficient practices extends the lifespan of existing hardware and reduces e-waste, which is projected to reach 82 million tonnes annually by 2030 [21]. That is why research in this area is important due to the fact that software optimizations can achieve substantial energy reductions, depending on the application, without requiring expensive hardware investments [22]. By integrating energy metrics into the software development's life-cycle, engineers can create products that are faster, more reliable, and regulatory-compliant [20, 19].

3 Related Work

The environmental impact of software systems is increasingly recognized, and the need for energy-efficient software development has gained attention. Existing approaches that reduce software energy consumption fall into three main categories: static code analysis tools, runtime energy measurement frameworks and developer education initiatives. This section describes existing work in each category and identifies the gap our approach addresses.

3.1 Static Analysis Tools for Energy Efficiency

3.1.1 Green Code Initiative

The Green Code Initiative [5] provides a SonarQube plugin which analyses the source code and detects patterns wasting energy. This plugin integrates with existing software quality pipelines and gives reports on sustainability related issues as well as metrics for maintainability and security.

Although the plugin shows how sustainability analysis can be integrated into existing development workflows, it has several limitations. First, it requires the SonarQube ecosystem and supporting infrastructure, which may limit accessibility for smaller projects or educational settings. Second, many of the implemented rules are derived from performance optimizations rather than explicitly connected to sustainability principles.

3.1.2 Creedengo JavaScript Analyzer

Creedengo [6] is another tool from Green Code Initiative which focuses on identifying inefficient coding practices in the JavaScript ecosystem. It detects patterns that negatively affect the performance and energy efficiency in web applications.

However, this tool is limited only to the JavaScript ecosystem, targeting mainly web performance patterns. In addition, it is limited to JavaScript and not very applicable to other programming languages or broader software engineering contexts.

3.1.3 ecoCode

ecoCode [24] is another SonarQube plugin developed by the Green Code Initiative that focuses on detecting energy code smells. It analyzes source code to identify patterns that may lead to excessive energy consumption, increased resource usage, or reduced device lifespan.

Similar to other static analysis tools, ecoCode demonstrates how sustainability concerns can be embedded into existing quality analysis pipelines. However, it also shares some limitations. Its initial focus on Android and specific programming languages restricts its general applicability, and the set of detectable energy smells is still evolving, meaning coverage is not yet comprehensive.

3.1.4 Green Code Analyzer - Previous Year Project

The Green Code Analyzer project [26] proposes a static analysis approach for detecting inefficient code patterns that can negatively impact energy consumption. It identifies patterns related to inefficient memory usage, unnecessary computations, and poor resource management using static code analysis. The tool provides developers with feedback on potential improvements that may reduce energy usage in software systems.

The project follows sustainability-oriented coding recommendations and how a developer can incorporate these recommendations into their coding practices to become 'greener'. However, this project primarily focuses on data science workflows and computational pipelines, targeting patterns commonly found in data processing and machine learning code.

3.2 Runtime Energy Measurement Frameworks

3.2.1 Green Coding Framework

The Green Coding Framework [25] is a tool that provides means for measuring energy consumption and CO2 emissions of software systems during execution. This enables developers to evaluate the environmental impact of their software through these runtime measurements.

These frameworks however, operate at runtime and infrastructure level rather than analysing the source code directly. As a result, they do not provide any actionable guidance during the development phase, while coding.

3.2.2 EnergiBridge

EnergiBridge [27] is a cross-platform command-line tool that measures energy consumption of software during execution. It supports multiple operating systems and accesses hardware-level energy data through interfaces such as Intel RAPL and Apple SMC.

While it provides accurate runtime measurements, it requires executing the software and does not offer guidance at the source code level during development.

3.2.3 CodeCarbon

CodeCarbon [28] is a Python-based tool that estimates the carbon emissions of computational processes. It combines hardware energy measurements with regional carbon intensity data to provide an estimate of CO_2 emissions.

Although useful for raising awareness of environmental impact, it is primarily targeted at data science workflows and does not provide actionable insights directly within the coding phase.

3.2.4 PowerJoular

PowerJoular [29] is a monitoring tool designed to estimate energy consumption of running programs, with support for fine-grained analysis such as method-level energy usage in Java applications.

However, similar to other runtime tools, it focuses on measurement rather than prevention, offering limited support for guiding developers toward energy-efficient coding practices during development.

3.3 Developer Education Initiatives

3.3.1 Green Software Foundation Training

The Green Software Foundation [7] provides the *Green Software for Practitioners* training course, which introduces key concepts such as energy efficiency, carbon-aware computing, and sustainable software design. The course aims to improve developer awareness and understanding of the environmental impact of software systems.

However, as a standalone educational resource, it is not integrated into the developer's daily workflow or coding environment, making it difficult to directly apply the concepts during implementation.

3.3.2 W3C Sustainable Web Design Guidelines

The W3C Sustainable Web Design Community Group [8] has proposed guidelines for building environmentally sustainable web applications. These

include practical recommendations for reducing energy consumption, each accompanied by estimated impact and implementation effort.

While these guidelines provide valuable best practices, they are primarily presented as reference material and are not embedded into development tools, limiting their effectiveness in guiding real-time coding decisions.

3.4 Research Gap and Our Approach

Existing work focuses more on specific code patterns through static analysis, measuring runtime energy consumption, or providing sustainability education through external courses. There is limited work that combines these concerns into a lightweight, workflow-integrated tool that both provides sustainability knowledge and tailors it to the developer's current codebase and focuses on improving the 'sustainability literacy' of developers. Many of these tools also rely on complex infrastructure and focus on performance optimization.

Our approach aims to address this gap by developing a lightweight CLI based tool that provides sustainability advice to developers during development. The tool uses a JSON knowledge base of sustainability tips derived from literature and existing resources combined with static analysis to tailor the tips to be most relevant to the developer. Unlike existing solutions that depend on infrastructures such as SonarQube or focus solely on runtime measurements, our tool aims to be simple, accessible, and educational, helping developers become aware of sustainable coding practices directly within their workflow.

4 Solution Proposal

We address this gap by building GreenTips, a developer-facing CLI tool designed to promote energy-aware coding practices through a low-effort, non-intrusive experience. Rather than requiring infrastructure setup or targeting a specific language ecosystem, GreenTips provides support through the command line interface (CLI) directly. By implementing a CLI, our tool aligns with the recent rise of "terminal-first" interfaces in software engineering workflows [30]. Due to the widespread adoption of AI-native CLI tools such as Claude Code, this approach ensures that sustainability tips are delivered directly within the developer's workspace and offers the possibility of incorporating sustainability focused AI agents into our tool. Additionally, instead of using a website or cloud-based tool which requires the developer to share their data or code, the CLI provides a more private way for developers to receive meaningful sustainability tips.

GreenTips delivers actionable sustainability tips whenever requested. If a file is given as part of a tip request, the code is analyzed for any detectable characteristics such as simple code patterns. If a pattern cannot be found, then the algorithm searches for file majority type within the project and outputs a programming language specific tip. If there are no detectable characteristics found in the file, a general tip is given. Tips are stored in a structured JSON dataset, where each entry contains the tip content, its energy-saving rationale, a traceable academic or industry source, an impact and effort rating, which programming language it is related to and the reason for the suggestion if it is based on a detectable trait. This design keeps the dataset independently useful and easy to extend.

The tip dataset is grounded in established literature, including the Green Software Foundation's energy, hardware, and carbon-awareness pillars, and peer-reviewed research on energy-efficient algorithms, data structures, and language-level efficiency. Tips span categories including computation, memory management, I/O, network efficiency, database access patterns, and carbon-aware deployment, with language-specific entries for Python, JavaScript, Kotlin, Swift, Java, and others.

5 Methodology

In order to better depict how GreenTips can be used in real world scenarios we utilize the Scenario-Based Design (SBD) framework by Rosson and Carroll [31]. Due to the fact that GreenTips aims to educate developers about sustainable software practices, it is

important to consider how the tool is incorporated in everyday workflows and examine its usage by different types of engineers. SBD enables this type of analysis by describing specific usage scenarios and examining how different types of developers might respond to the tool. In this sense, the focus is not on whether GreenTips functions correctly, but on whether it meaningfully affects user behaviour and decision-making.

This section therefore introduces the core idea behind GreenTips and analyses it through a set of representative developer personas. For each persona, we consider a plausible usage scenario and use claims analysis to reason about the potential benefits and limitations of the proposed design.

5.1 Root Concept

The root concept serves as a starting point for the design by stating what GreenTips is intended to achieve and why such a tool is needed. The aim is to outline a set of assumptions about developer behavior, current limitations in practice, and how our tool addresses them through its three-tier delivery hierarchy.

5.2 Persona Scenarios and Claims Analysis

As depicted in Table 1, three personas are defined, each representing a different type of developer with a different way of engaging with the tool, in order to explore how these assumptions hold in practice. The goal is to capture a range of behaviors and expectations that are relevant to the design.

For each persona, a short scenario is created to describe how GreenTips might be utilized during regular development work. This is followed by a claims analysis, which examines the potential benefits and drawbacks of specific design features.

Our first persona is Maya, a mid-level full-stack developer who focuses on functionality and delivery speed, and does not typically consider energy efficiency when making design decisions.

Scenario. While working on a service interface, Maya runs `greentips tip ./src/api_handler.py` on her current file. GreenTips detects Python as the language and identifies a pattern of repeated JSON serialization in API responses. Based on this, it recommends considering Protocol Buffers instead of JSON for frequent data exchange. She briefly evaluates the suggestion and considers whether it is worth adopting in her current implementation.

Alex is a senior software engineer who evaluates design decisions at a system level and is cautious about adopting new approaches without strong justification.

Scenario: Alex is leading a new Python-based mi-

Table 1: Root Concept for GreenTips Sustainability CLI

Component	**Contribution to the Root Concept**
High-level vision	Provide developers with educational green design suggestions through a CLI interface, integrated into everyday development workflows.
Basic rationale	Developers often lack visibility into the environmental impact of design decisions. GreenTips provides context-aware suggestions ranging from general principles to specific code-level patterns.
Stakeholder groups	
Maya (Mid-level)	Uses file-level inspection to identify immediate energy-heavy patterns in her code.
Alex (Senior)	Uses project-wide language detection to guide architectural decisions based on language-specific efficiency.
Liam (Hobbyist)	Leverages quick mode to build broad sustainability literacy through general-purpose tips.
Starting assumptions	The tool uses an automated rule-matching engine for specific files and a fallback hierarchy (Language → General) for broader contexts.

Table 2: Claims Analysis for Maya

Design feature: Context-sensitive, code-aware sustainability tips delivered via CLI
+ The tip is grounded in an actual pattern detected in her code, making it more actionable than a generic recommendation. + Low effort required — she points the tool at her file and gets a relevant suggestion without leaving her workflow. – Detection relies on static pattern matching, which may miss deeper architectural context (e.g., the JSON usage might be required by an external API contract).

croservice project and wants to ensure the team follows language-appropriate green practices. He runs `greentips tip` in the root directory. The tool scans the project, identifies Python as the majority language, and returns Tip 002: "Use lazy evaluation instead of eager computation." This encourages the team to use generators for large sequences to reduce CPU and memory waste.

Table 3: Claims Analysis for Alex

Design feature: Language-specific tips via automated project detection
+ Provides high-level guidance tailored to the project's primary technology stack without requiring specific file paths. + Inclusion of research sources supports evidence-based senior-level decision making and team knowledge sharing. – In projects with mixed languages, the tool identifies the most frequent extension, potentially overlooking secondary language stacks.

Liam is a hobbyist developer who aims to build broad sustainability literacy and improve his general awareness of green software practices.

Scenario Liam is a hobbyist who wants to improve his general awareness of sustainable software practices but is currently browsing documentation rather than coding. He runs `greentips tip` without a target path. Since no project context is provided, the tool falls back to a general tip, such as Tip 029: "Schedule batch jobs during low-carbon periods."

Table 4: Claims Analysis for Liam

Design feature: General sustainability fallback mode
+ Builds broad conceptual literacy regarding carbon-aware computing and energy-efficient scheduling. + Extremely low entry barrier; provides value even when the user is not actively working within a code repository. – Tips may feel less "urgent" compared to code-aware suggestions as they are not tied to a specific line of code.

From the above claim analysis, it is evident that tips must remain concise and actionable with low effort in order to avoid being ignored by users. Also, the inclusion of cited sources is necessary to attract more experienced users, even if this introduces additional complexity.

5.3 Usability Specifications

The aforementioned scenarios showcase a common interaction pattern in which a developer briefly consults the tool during development and decides whether to act on the information provided or not. Because of that, the user-tool interaction must be lightweight since GreenTips is not intended to interrupt a developer's current workflow, but to offer a quick suggestion during ongoing tasks. Therefore, usability targets are defined with this constraint in mind.

As depicted in Table 5, the performance targets reflect the expectation that a user's interaction with the tool should remain quick and unobtrusive regardless of the interaction mode. Performance constraints are tied to ensure that the most immediate interactions, such as receiving a general tip, occur in under 1 second, while more complex operations like code-aware file inspection remain below a 2-second threshold. These strict limits are necessary because longer wait times would discourage repeated use in everyday

Table 5: Usability Specification for the GreenTips CLI

Task Context: A developer invokes the CLI to obtain a sustainability tip using code-aware inspection, language detection, or general fallback.		
Subtask	**Performance Targets**	**Satisfaction Targets**
1. Inspect specific file for patterns (Maya)	$\leq$ 2 seconds, 0 errors	4.5 on *precision*
2. Detect project language and suggest tip (Alex)	$\leq$ 1 second, 0 errors	4.0 on *relevance*
3. Provide general suggestion (Liam)	$\leq$ 1 second, 0 errors	4.5 on *convenience*
4. Access cited research link	$\leq$ 60 seconds, 0 errors	4.5 on *trustworthiness*

development environments. Furthermore, the satisfaction targets for precision and relevance emphasize that the information must be immediately understandable to fit within a developer's active workflow.

On the other hand, the satisfaction targets focus on how the interaction is perceived by the user. The targets for convenience and trustworthiness are particularly critical, as GreenTips relies on voluntary and repeated engagement. If obtaining a tip requires excessive effort or if the provided rationale is not perceived as credible, the likelihood of continued use decreases significantly. This is why the tool prioritizes high convenience for general tips and high precision for code-aware suggestions.

The subtask of accessing cited references allows for a longer interaction time of up to 60 seconds, as this action is optional and more reflective in nature. Providing a direct link to research sources is essential for establishing trustworthiness and offering deeper technical detail, especially for experienced users or those specifically seeking to adopt a more sustainable mindset.

The aforementioned specifications provide a concrete way to evaluate if GreenTips achieves its goal of being a lightweight tool that integrates into everyday development while offering sufficient depth to influence how developers think about sustainability across different levels of technical context.

6 Discussion

While the GreenTips CLI provides a lightweight approach to integrating sustaianable practices into daily software engineering, there are several open challenges and various limitations to consider in the current design.

A major open challenge for the future of the tool, which we realised after our analysis of senior developer needs, is ensuring that the tips being presented are being kept up to date and are constantly adapted by experts to fit new technology and design patterns. The software development landscape evolves rapidly with modern technologies, and static advice can age quickly and become obsolete. To adapt to this challenge, we added contributing guidelines and templates that allow users to actively add tips or features to the repository. Through open-sourcing the continued generation and maintenance of this sustainable software design knowledge base, the tool can continuously grow and adapt to modern tech stacks. Another key factor in retaining the reliability of our tool is that the tips need to be thoroughly reviewed for validity and applicability. As the dataset expands with the input of external community member contributions, maintaining the factual accuracy and efficacy of the advice is crucial. To alleviate this issue, a robust reviewing or approval system can be used to increase the recommendation of higher quality tips. Implementing a community voting mechanism would democratize the tip maintenance process, hopefully ensuring that effective and reliable sustainable practices can be integrated into the developer's workflow.

Furthermore, we acknowledge that it is difficult to engage users and initiate widespread tool use. We deliberately chose to implement the tool in a CLI format to integrate more easily with a developer's workflow. However, the developer still needs to initi-

ate tool use and specifically request a tip via a CLI command during development. This model relies on the user's pre-existing intent or memory to invoke the command, which could limit the tool's impact for developers who are not already actively thinking about sustainability. It will take more than the GreenTips CLI to switch the mindset of developers to create more sustainable code. However, developing and proliferating tools specifically created for sustainability can create greater awareness for sustainable software engineering through a gradual process. [36]

7 Future Work

In this section we will outline a few proposals that could improve our current implementation and that we envision implementing in the near future.

First of all, an important next step is incorporating different LLMs into the GreenTips CLI. With that addition, instead of just showing a random tip and personalized suggestions, the tool will work with AI coding agents in order to automatically scan the files user is currently working on. An AI coding agent could spot energy-heavy patterns, suggest more efficient strategies and implement them on the spot. This would turn GreenTips from a simple suggestion tool into a "green" pair programmer that helps engineers refactor their code-base for efficiency in real-time. Rather than replacing the need for developer literacy, this approach changes how developers learn. Instead of reading tips and figuring out how to apply them, developers learn by reviewing AI-proposed refactors on their own code, which is arguably a more effective path to understanding sustainable practices. It is also worth noting that more and more developers code primarily through LLMs rather than writing code by hand. This means that embedding sustainability awareness into the AI layer itself has the potential to become even more impactful than targeting developer literacy directly, as it ensures sustainable practices are applied regardless of how code is written.

Secondly, we will conduct an empirical evaluation with real developers. Since we currently rely on scenario-based design, our predictions are still hypothetical. In order to evaluate our tool more efficiently and precisely, we would conduct user studies or field experiments to measure:

- Whether developers actually change behavior.
- Which types of tips are ignored vs acted upon.
- A/B testing: with vs without GreenTips in real workflows.
- Conduct developer research on how much knowledge they gained over time using the tool.

Finally, we will work on linking tips to measurable impact by estimating actual energy/carbon savings from applied suggestions, bridging the gap between awareness and quantifiable impact. This would make our statement and tool even more impactful and measuring real energy savings could be a great feedback and help shape the direction tool is going towards.

8 Conclusion

The ICT sector's growing contribution to global greenhouse gas emissions highlights the need for sustainability to be considered as a core concern in software engineering. However, as illustrated throughout this paper, many developers lack the awareness and guidance needed to act on this in practice, and existing tools do not adequately address this at a high level of abstraction.

To address this gap, we proposed GreenTips, a lightweight cross-platform CLI tool that provides developers with practical and verifiable sustainability tips in their workspace. GreenTips is intentionally simple to use and does not require additional infrastructure, thus lowering the barrier to adoption for users. Our scenario-based evaluation suggests that it can support developers of varying experience levels in building sustainability awareness over time, though the actual effect on developer behaviour remains to be validated through empirical studies.

In conclusion, improving sustainability literacy among engineers is an important step toward to reducing the environmental impact of software systems, and tools like GreenTips can play a significant role in that process.

References

[1] International Telecommunication Union (ITU), "Population of global offline continues steady decline to 2.6 billion: Achieving universal and meaningful connectivity by 2030," ITU Media Centre, Sep. 12, 2023. Available: https://www.itu.int/en/mediacentre/Pages/PR-2023-09-12-universal-and-meaningful-connectivity-by-2030.aspx

[2] World Bank and International Telecommunication Union (ITU), "Measuring the Emissions and Energy Footprint of the ICT Sector: Implications for Climate Action," World Bank Group, Washington, DC, 2024, doi: 10.1596/41238.

[3] C. König, D. J. Lang, and I. Schaefer, "Sustainable Software Engineering: Concepts, Challenges, and Vision", *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, 2025, doi: 10.1145/3709352.

[4] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause, "An Empirical Study of Practitioners' Perspectives on Green Software Engineering," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, Austin, TX, USA, May 2016, pp. 237–248, doi: 10.1145/2884781.2884810.

[5] https://green-code-initiative.org/

[6] Green Code Initiative, "Creedengo JavaScript," Available: https://github.com/green-code-initiative/creedengo-javascript

[7] Green Software Foundation, "Green Software for Practitioners," Available: https://learn.greensoftware.foundation/

[8] W3C Sustainable Web Design Community Group, "Sustainable Web Design Guidelines," Available: https://w3c.github.io/sustainableweb-wsg/

[9] C. Calero and M. Piattini, *Green Software Strategies*. Springer, 2015.

[10] G. Pinto and F. Castor, "Energy Efficiency: A New Concern for Application Developers", *Communications of the ACM*, vol. 60, no. 12, pp. 68–75, 2017.

[11] Green Software Foundation, "State of Green Software 2024", *Green Software Foundation Reports*, 2024.

[12] C. Freitag, M. Berners-Lee, et al., "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations," *Patterns*, 2021.

[13] Carbon4 Finance, "ICT: a sector disconnected from the climate reality?," Jun. 2023.

[14] UK Parliament, "Energy Consumption of ICT," POST-PN-0677, 2025.

[15] K. Carter et al., "Energy and Time Complexity for Sorting Algorithms in Java," *arXiv:2311.07298*, rev. May 2024.

[16] S. Garcia-Rodriguez et al., "Energy-Efficient Code Patterns: A Systematic Mapping Study," *ACM Computing Surveys*, 2023.

[17] J. Mancebo et al., "Green Architectural Patterns for Cloud-Native Applications," *IEEE Software*, vol. 41, no. 1, 2024.

[18] D. Connolly and M. Ó Cinnéide, "Energy efficiency of the Visitor Pattern: contrasting Java and C++ implementations," *Empirical Software Engineering*, vol. 28, no. 6, 2023.

[19] R. Verdecchia et al., "Energy-Aware Software Testing," *ICSE 2025 - New Ideas and Emerging Results (NIER)*, May 2025.

[20] P. van de Kamp, Software Improvement Group (SIG), "Green IT: How green software development can reduce your carbon footprint," 2024. Available: https://www.softwareimprovementgroup.com/blog/green-software-development/

[21] Green Compute UK, "Global Footprints of ICT," 2025.

[22] A. Trefethen and J. Thiyagalingam, "Energy-aware software: Challenges, opportunities and strategies," *Journal of Computational Science*, vol. 4, no. 6, pp. 444–449, 2013, doi: 10.1016/j.jocs.2013.01.005.

[23] B. Bamford, "AI Inference Costs in 2025: The $255B Market's Energy Crisis," Tensormesh, Dec. 2025.

[24] O. Le Goaër and J. Hertout, "ecoCode: A SonarQube Plugin to Remove Energy Smells from Android Projects," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022. doi: 10.1145/3551349.3559518.

[25] Green Coding Solutions, "Green Metrics Tool," Available: https://www.green-coding.io/products/green-metrics-tool/

[26] Green Code Analyzer, 2025. Available at: `https://luiscruz.github.io/course_sustainableSE/2025/papers/g1_green_code_analyzer.pdf` (accessed April 2, 2026).

[27] H. L. Müller et al., "EnergiBridge: A Cross-Platform Tool for Energy Measurement," Available: https://github.com/tdurieux/EnergiBridge

[28] M. Lottick et al., "CodeCarbon: Track and Estimate the Carbon Footprint of Your Computations," Available: https://codecarbon.io/

[29] PowerJoular Project, "PowerJoular: Power Monitoring Tool," Available: https://github.com/joular/powerjoular

[30] O. Ural, "The CLI Renaissance: Why Developers Are Returning to the Terminal in the Age of AI and Platforms," Medium, Dec. 2025. Available: https://medium.com/mocean-labs/the-cli-renaissance-why-developers-are-returning-to-the-terminal-in-the-age-of-ai-and-platforms-8e73ceef8002

[31] M. B. Rosson and J. M. Carroll, *Scenario-based design*, in *The Human-Computer Interaction Handbook*, J. A. Jacko and A. Sears, Eds. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 2002, pp. 1032–1050.

[32] TTMS, "Growing Energy Demand of AI - Data Centers 2024–2026," Feb. 2026.

[33] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[34] M. Radu, "Spatial and Temporal Shifting in Carbon-Aware Computing," *Journal of Sustainable Computing*, Jan. 2025.

[35] J. Salminen, K. W. Guan, S.-G. Jung, and B. Jansen, "Use Cases for Design Personas: A Systematic Review and New Frontiers," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*, ACM, 2022. Available: https://dl.acm.org/doi/10.1145/3491102.3517589

[36] Ana Moreira, Patricia Lago, Rogardt Heldal, Stefanie Betz, Ian Brooks, Rafael Capilla, Vlad Constantin Coroamă, Leticia Duboc, João Paulo Fernandes, Ola Leifler, Ngoc-Thanh Nguyen, Shola Oyedeji, Birgit Penzenstadler, Anne-Kathrin Peters, Jari Porras, and Colin C. Venters. 2025. A Roadmap for Integrating Sustainability into Software Engineering Education. ACM Trans. Softw. Eng. Methodol. 34, 5, Article 139 (June 2025), 27 pages. https://doi.org/10.1145/3708526