Practical Adoption of Green Coding: A Comparative Study across Organizations in Finland and Globally

Ichchha Moktan[1], Muhammad Asif Khan[2], Shola Oyedeji[3], Mikko Puonti[4], Adisa Mikhail Ola[5], and Jari Porras[6]

[1] Ichchhamoktan07@gmail.com LUT University, Finland,
[2] Asif.khan@lut.fi LUT University, Finland,
[3] Shola.oyedeji@lut.fi LUT University, Finland,
[4] Puonti@iki.fi Solita, Finland,
[5] Mikhail.adisa@lut.fi LUT University, Finland,
[6] Jari.porras@lut.fi LUT University, Finland,

Abstract. As the environmental impact of digital technologies gains increasing attention, green coding has emerged as a critical aspect of green software engineering. This study investigates the practical adoption of green coding practices in organizations in Finland and globally to understand how they approach green coding in their software-development workflow. This study adopts a qualitative research methodology, combining a systematic literature review and semi-structured interviews. Key findings indicate that while organizations adhere to academic-recommended coding practices such as caching, lazy loading, asynchronous processing, microservices, and content/UI optimization, industry prioritization often depends on business needs, with less emphasis on energy-efficient programming languages or priority in green coding, especially in organizations outside the EU. Interestingly, in larger organizations, such as banks, green coding initiatives are mostly driven from the bottom up and led by software engineers. This reflects a positive change. However, innovative academic proposals, such as large language model (LLM)-based assistants for recommending green libraries, remain prototypes without industrial adoption. While LLM-based assistants show potential in recommending greener libraries, their current energy demand raises questions about net sustainability. Future research should evaluate their carbon trade-offs to ensure these tools contribute positively to sustainability goals. This study underscores the need for closer collaboration between academia and industry to promote the practical adoption of effective tools, along with standardized guidelines and sustainability-oriented education, to enable the wider adoption of green coding practices.

Keywords: green coding, energy efficiency, green software engineering, sustainability, Finland

1 Introduction

Green information and communication technology (ICT) is becoming increasingly popular, especially in the European Union (EU). The ICT sector currently accounts for approximately 2-7% of global greenhouse gas emissions and is projected to reach 14% by 2040 [1]. Many countries, including Finland, have pledged to become carbon neutral by 2050, as decided at the UN Climate Change Conference in Glasgow (COP26). Finland has set an even more ambitious target, aiming to achieve carbon neutrality by 2035, creating a challenging climate goal for the ICT industry [2]. Within the broader ICT industry, the software sector plays a crucial role, given its rapid growth and the increasing use of digital services in daily life. Organizations like the Green Software Foundation[7] are enhancing software sustainability in the EU by establishing standards, offering training, and collaborating with technology companies to minimize emissions associated with software usage. Similarly, in Finland, TIEKE[8] is leading efforts such as the Green ICT Ecosystem, which brings together academia, businesses, and government to promote the development of energy-efficient software.

Green coding, a subset of green software, is gaining popularity among software practitioners. Research has also shown its potential to reduce energy consumption and enhance energy efficiency through various approaches, techniques, and strategies [3]. For example, one study demonstrated that the selection of an appropriate algorithm for data collection can reduce energy consumption by 38% [4]. Similarly, another study demonstrated that energy consumption can be reduced by up to 67% through various energy-efficient architectural design strategies [5]. Although there is a growing awareness and desire among professionals to adopt green coding practices, they lack practical examples to guide them in integrating green coding into their daily workflows [6–9]. Therefore, this study aims to fill this gap by providing practical insights and real-world examples through exploring industry practices, perceptions, and experiences. Furthermore, this study offers a comparative perspective by examining how these strategies are implemented in various regions. This study focuses on the following three key research questions (RQ):

- RQ1: What green coding practices are proposed in academic literature (state-of-the-art)?
- RQ2: What are the green coding practices adopted in organizations in Finland and globally (state of practice)?
- RQ3: How do green coding practices vary across organizations in Finland, other EU countries, and non-EU regions?

The remainder of this paper is organized as follows: Section 2 outlines the background and related work. Section 3 details the study's design. Section 4 presents the results and analyses. Section 5 discusses the findings and implications. Section 6 presents the conclusions and future work.

[7] https://greensoftware.foundation/
[8] https://tieke.fi/en/green_ict_ecosystem

2 Background & Related work

2.1 Green Coding: Concept, Importance, and Impact

Green coding refers to writing and optimizing code to reduce energy use and environmental impact through efficient algorithms, minimal data transfer, and removal of redundant processes [10]. Growing regulatory and stakeholder pressures further encourage the adoption of sustainable software practices [11]. Academic and industrial discussions increasingly emphasize the role of software efficiency in mitigating the carbon footprint of digital systems, linking performance optimization with environmental responsibility [12].

Numerous green coding strategies have been proposed in academia; however, their practical adoption remains limited [8]. This study examines how organizations translate sustainability goals into coding practices by comparing current advancements with practices observed in Finland and globally.

2.2 Related Work

Melo et al. [13] conducted a qualitative study involving 21 Brazilian software professionals to investigate how sustainability is understood and practiced in real-world settings. Their findings revealed growing awareness among developers but also showed that practical implementation is limited due to limited expertise and insufficient dissemination of sustainable practices within teams. This concern is echoed by Karita et al. [6], who conducted a state-of-practice survey that revealed that developers often recognize sustainability as a quality attribute. The current literature also reveals a gap between the state-of-the-art and practice. Junger et al. [14], in their review of existing research on green software development, point out that while academic discourse increasingly explores the potential and principles of green coding, there is a notable lack of empirical studies that examine how these principles are translated into actual development workflows, tools, or organizational practices. Other researchers have examined the organizational and systemic factors influencing the adoption of green software. For example, Pathiranage and Madrika [15] explored the adoption of Green Information Systems (Green IS) in global IT organizations, finding that while these practices have the potential to improve environmental outcomes, many organizations lack the strategic commitment to adopt them. Similarly, Lohikoski [16] identified barriers such as the absence of formal policies, unclear incentives, and low organizational support through interviews with ICT professionals, even in cases where developers were personally committed to sustainable software development.

3 Study Design

This study was designed to study both the state of the art and state of practice through a systematic literature review and semi-structured interviews to gain a comprehensive understanding of the green coding practices mentioned in academia and their practical adoption in the industry across Finland and globally. The overview of the study structure is detailed in **Figure** 1 with three interconnected phases:

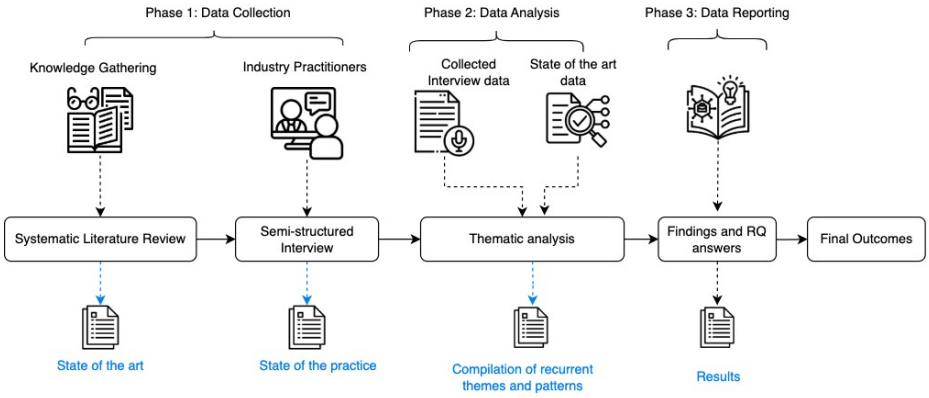


Fig. 1: Overview of the research design.

3.1 Systematic Literature Review:

The Systematic Literature Review followed a structured protocol to ensure consistency and reliability of the results. This protocol includes database searches, inclusion and exclusion criteria, and an important categorization of the selected studies into different themes. The search covered publications from 2015–2025. Four different databases, IEEE Xplore, ACM, Web of Science, and Scopus, were utilized according to their reputation for publishing a substantial number of articles related to the field of study and covering the most relevant publications. The overall data collection process for the SLR is shown in **Figure** 2. The following search string was used to search for primary studies:

"Green coding" OR "green software" OR "sustainable software" OR "energy-efficient software" OR "software sustainability") AND ("adoption" OR "implementation" OR "practice" OR "use" OR "integration") AND ("software industry" OR "software companies" OR "software development"

3.2 Semi-Structured Interviews:

To gain qualitative insights into the real-world adoption and perception of green coding practices among software professionals, 15 semi-structured interviews with software professionals from organizations in Finland (5) and outside Finland (10) were conducted.

A. Interview Design: The interviews were designed to be flexible, guided by a structured framework that ensured consistency among participants while allowing for elaboration based on individual experiences and contexts. Each interview session lasted approximately 45–60 minutes and was conducted using Microsoft Teams. Participants were asked to provide their consent for the recording of the meeting. To capture how organizations approach green coding adoption with different regulations, the sample included five organizations, each from Finland, other EU countries, including the UK, and non-EU countries.

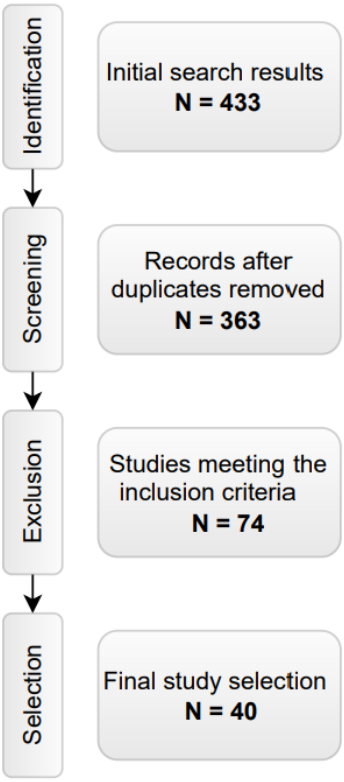


Fig. 2: Overview of the literature review process

B. Participant recruitment: Potential organizations were identified by reviewing publicly available materials, such as company sustainability reports, corporate blogs, and official websites, to identify those that explicitly referenced green coding or energy-efficient software development practices, particularly in the EU. Organizations were chosen if they were involved in software development or had an internal software development team. Eligible participants needed a solid understanding of software engineering practices, regardless of whether they were hands-on developers or in technical leadership roles (e.g., developers, architects, and tech leads). In total, 15 organizations participated in the study, with one representative from each organization interviewed. The participating organizations were based in Finland(5), as well as other EU countries, such as the Netherlands(3) and France(1), and non-EU countries, including the United Kingdom(1), Canada(1), the United States(1), and India(3). The organization names were anonymized and replaced with the company A-N placeholders, as shown in **Table** 1 before proceeding with data analysis, to protect confidentiality. The size of each company is classified as small (fewer than 50 employees), medium (50 to 249 employees), and large (250 or more employees).

Table 1: Distribution of Participating Organizations

S.N.	Organization Name	Country	Type	Size
1	Company A	France	Financial Institution	Large
2	Company B	Finland	Software Development	Medium
3	Company C	Finland	IT Services & Consulting	Small
4	Company D	Finland	IT Services & Consulting	Small
5	Company E	Finland	IT Services & Consulting	Large
6	Company F	Finland	Telecommunications	Medium
7	Company G	Netherlands	Software Development	Large
8	Company H	Netherlands	Financial Institution	Large
9	Company I	India	Software Development	Large
10	Company J	India	IT Services & Consulting	Large
11	Company K	Canada	Software Development	Small
12	Company L	USA	Financial Institution	Large
13	Company M	India	Software Development	Large
14	Company N	UK	Software Development	Small
15	Company O	Netherlands	Financial Institution	Large

3.3 Data Analysis: Thematic analysis was used to identify key patterns in both the academic and industry sectors. It followed Braun and Clarke's six-phase framework [17], which started with the extraction of interview recordings as text-based transcriptions. The transcripts were thoroughly examined and relevant data segments related to the research questions were highlighted. The codes started to emerge from the data rather than having a prepared set of codes; therefore, an inductive coding (bottom-up) approach was employed. Each relevant segment/sentence was assigned a code that reflected its core idea with the help NVivo. Thus, a codebook was iteratively developed to track and organize all the codes across the transcripts. Once all data were coded, similar codes were further grouped into broader categories, which formed the basis of preliminary themes. Based on this, potential themes such as architecture & design patterns, code-level optimization, etc., have started to emerge. Each preliminary theme was critically reviewed, and themes that were too broad, overlapped, or weakly supported were refined, combined or removed. After the review and refinement process, each theme was given a descriptive name to ensure that it communicated the core idea effectively. Sub-themes were also created wherever necessary to reflect a better description of the software development experiences and practices on green coding in software-development projects.

4 Results & Analysis

4.1 State of Art: Systematic Literature Review findings

The literature highlights a wide spectrum of practices, ranging from code-level optimizations to architectural and infrastructure-level practices, as detailed in the **Table** 2 below and explained in the subsequent subsections.

Table 2: State of the Art in Green Coding: Strategies, Focus Areas, and Emerging Trends

Category	Focus Areas	Green Coding Strategies
Code-Level Optimization	Programming Language Choices	Use of native languages (C, Rust) [1, 18, 19]; energy-efficient VM languages [18]; green compilers and optimized native code [20, 21].
	Database and Data Storage Optimization	Efficient databases (SQL vs. NoSQL) [22]; indexing, caching, and compression [21]; efficient data structures [19].
	Algorithm-wise Optimization	Dynamic programming, memoization, and merge sort [23]; refactoring energy smells [3].
	Code-Specific Optimization	Code refactoring and optimization [19, 24]; reduce algorithmic complexity; select energy-efficient algorithms [20, 25].
Content & UI Design	UI/UX	Avoid unnecessary graphics or animations [21, 23]; energy-efficient interfaces (dark mode, adaptive streaming, and compressed images) [21, 23]; lazy/asynchronous loading [21, 26].
	Content-specific	Use of compressed and responsive images [23]. adaptive video streaming [21]; efficient formats (WebP, SVG) [21].
	Device/User-Aware Operations	Push notifications instead of polling; race-to-idle mechanisms [23] dynamic retry delays and batch operations [23]; prefer WiFi over mobile data [23].
Architecture & Design Patterns	Green Software Design Patterns	Edge computing, caching, modularization [19, 21].
	Sustainable Software Architecture	Genetic Improvement (GI) methods [27]; modular and microservice architectures [19]; agile, reusable, and flexible design [24, 28].
Cloud and Infrastructure	Green DevOps Practices	Green continuous testing [29] sleep modes, and disabling unused ETL processes [30, 31]; demand shaping and workload scheduling [19].
	Virtualization and Cloud Optimization	VM scaling and consolidation [20] energy-aware infrastructure [19]; choosing green cloud providers [1].
Emerging Technologies & Trends	Advanced Technologies	AI, IoT, Blockchain integration for optimization [32]; multi-objective optimization (e.g., GEMMA for GUIs) [33].
	Tool Proposals	SPELL, GreenAdvisor, PEEK [19,34]; LLMs recommending green libraries [35]; auto-generated energy reports [36].

A. Code-Level Optimization: Research emphasizes the need to optimize the source code to improve energy efficiency. The choice of programming language plays an important role, with native languages such as C and Rust offering higher efficiency [1, 18, 19] and virtual machine languages such as C# along with the use of green compilers [18, 20, 21]. Database-specific strategies, such as indexing, caching, and compression, help reduce data storage energy consumption [21], whereas algorithmic optimizations, such as dynamic programming and memoization, reduce computational energy demands [23]. Refactoring codebases further contributes to significant savings over time [19, 24].

B. Content and UI Design: Studies have highlighted UI/UX strategies to reduce energy consumption, including minimizing unnecessary graphics and animations, adopting energy-efficient designs such as dark mode and adaptive streaming, and using compressed image formats (WebP, SVG) [21, 23]. Techniques such as lazy loading, asynchronous content loading, push notifications over polling to minimize background activity, implementing race-to-idle patterns, using dynamic retry delays and batch operations, and favoring WiFi connections over mobile data improve both energy efficiency and user experience [21, 23, 26]

C. Architecture and Design Patterns: Sustainable software architecture patterns, such as modular architectures, microservices, edge computing, and caching, promote reusability and flexibility while reducing energy consumption [19] [21]. Genetic Improvement (GI) methods further optimize architectures over time, integrating AI-based optimization for enhanced efficiency [27].

D. Cloud and Infrastructure: Green DevOps practices, such as putting idle servers to sleep, disabling unused ETL pipelines, and demand shaping, help reduce energy usage [29, 30]. Virtualization strategies, such as VM scaling and consolidation, minimize over-provisioning [20], whereas choosing green-certified cloud providers and energy-aware libraries enables developers to reduce the carbon footprint of hosted applications [1, 19].

E. Emerging Technologies and Trends: Emerging technologies such as IoT, AI, and blockchain have been studied for both their energy impacts and mitigation potential [32]. Green coding tools (e.g., SPELL, GreenAdvisor, and PEEK) have been proposed to help developers monitor and improve the energy profile of their software [19, 34]. LLMs are being explored as intelligent assistants that recommend greener library alternatives during development [35].

Although various effective strategies have been documented, the literature highlights a fragmented landscape, lacking standardized and consolidated guidelines for practitioners. This highlights the need for standard and centralized guidelines, such as the Web Content Accessibility Guidelines (WCAG)[9] to support developers in the easy adoption of green coding practices. Furthermore, innovative tools and techniques, such as LLM-based assistants for recommending green libraries and tools such as SPELL for energy leak detection in code, show promise, but they remain conceptual or underutilized in real-world development environments. This gap between academic proposals and industrial adoption underscores the importance of applied research focused on integrating these tools into real-world industrial practices. Future research should therefore focus on bridging this gap by emphasizing usability, empirical evaluation, and integration within industry-standard practices.

[9] https://www.w3.org/TR/WCAG21/

4.2 State of Practice: Interview findings

Six dominant themes emerged from the analysis that are related to green coding practices and perceptions, as shown in **Table** 2. The industry findings, as detailed in **Table** 3, show a key focus on clean coding principles, modular design, SQL query optimization, API/payload reduction, and caching as common practices.

Table 3: State of Practice: Green Coding Strategies and Focus Areas Across Organizations (bold = mentioned by all organizations [A–N]).

Theme	Category	Practices Mentioned [Organizations]
Code-Level Optimization	Clean Coding Principles	Naming conventions and modular functions [O,F]; SOLID principles [I,L]; efficient data structures (e.g., dictionaries over lists) [G,L].
	Code Efficiency	Time/space complexity considerations [O,A,L] efficient constructs (e.g., `StringBuilder` over concatenation) [A]; code reviews for quality, consistency, and security [A,F,K,L,N].
	Data Handling	**DB optimizations (indexing, query tuning, selective fetching) [A–N]**; database engine optimization; regular cleaning and maintenance [A].
	Network Load	**Lean API design, minimized payloads, batching [A–N]**; avoid redundant API calls [H,N].
Architecture & Design	Design Patterns	**Caching, lazy loading, memoization [A–N]**.
	Architecture	microservices and modular design [I,L,M] event-driven architecture [A,D].
	Concurrency	Multithreading and concurrent programming [A,J,L].
Frontend & UI	Asset Optimization	Minify and compress assets [B,H,K]; use WebP; compress media such as videos/animations [C,H].
	UI Design	Streamlined UI and short user journeys [B,C]; accessibility standards [B,L]; dark mode [H]; lazy loading [B,K].
DevOps	CI/CD Pipelines	Parallel jobs to reduce the build time [C]; scheduled builds and automated testing [I,E].
Cloud & Infrastructure	Resource Management	Automated shutdowns [O,A,E,F,H,L]; right-sizing and demand-based usage [A,D,I,K,E,H].
	Scaling / Scheduling	Serverless deployments and autoscaling [C,D,G,H,J]; Energy-aware scheduling carbon-aware datacenters [H,M,N].
	Cleanup	Delete unused resources and avoid cloud waste [D,A,F].
Tooling & Monitoring	Carbon Estimators	`CO₂.js`, CodeCarbon, Cloud Carbon Footprint → estimate emissions from code, cloud, or web workloads [A,C,O].
	Website Assessment	EcoIndex, Ecograder, Google Lighthouse, WebsiteCarbon, ElectricityMaps → website environmental and performance assessments [A,B,C,H,K].
	Monitoring Dashboards	Kepler (K8s), Azure/AWS dashboards, and custom BI tools → monitor infrastructure energy and resource usage [A,D–L,H].
	Others	DailyClean, SlimFaaS → lightweight Kubernetes automation [A]; Firefox Profiler, Playwright → resource profiling and test automation [A,B].

A. Code-Level Optimization: Most participants emphasized efficient, clean, and optimized code as a key green coding strategy for achieving green coding. Common practices include applying clean coding principles, such as clear naming conventions, modular functions, selecting appropriate data structures (e.g., dictionaries over lists), optimizing algorithms for time and space complexity, and reducing redundant application programming interface (API) calls. Batch processing, SQL query optimization, and regular database maintenance were also highlighted to improve performance and reduce resource utilization. Code reviews are also used to check for inefficient patterns, such as redundant loops or non-performant data structures.

B. Architecture & Design Patterns: Organizations employ architectural choices and design patterns to optimize processing, reduce unnecessary data handling and improve resource utilization. One participant shared: *"We use event-driven architecture to respond only when an event occurs rather than continuously processing."* Other approaches include caching and memoization to reduce redundant processing, lazy loading to defer data or resource loading until necessary, and asynchronous or concurrent programming to maximize CPU efficiency. Modularization and microservice architecture are also used, one participant shared, *"We make everything modular to ensure that each service can be scaled or updated independently."* A minimalism mindset is also embraced, a participant shared *"We try to fetch as little data as possible, only what is wise and necessary."*

C. Tooling and monitoring: Organizations use different tools to monitor, measure, and improve the efficiency of their codes, software, and infrastructure. Google Lighthouse was commonly used for web performance and environmental impact assessment, and traditional code analysis tools such as SonarQube, but with integrated 'green' rulesets or performance criteria to catch memory leaks or flag inefficient code. One participant shared *"Lighthouse provides a good baseline."* Another shared, *"We use tools like Daily- Clean to automatically shut down unused Kubernetes pods, and SlimFaaS to run only minimal functions when needed."* Some organizations are also developing in-house tools. One participant shared, *"We are building a CI/CD pipeline monitoring tool so that the developers can see whether the current deployment process is taking less or more energy."* Infrastructure-level dashboards, such as AWS CloudWatch and Azure CO_2 monitoring, are also used to monitor and reduce idle resource usage.

D. Cloud and Infrastructure Efficiency: Most organizations employ strategies to optimize cloud resources, including resource right-sizing, automated scheduling and shutdown of idle resources, dynamic scaling, and serverless architecture. One participant stated, *"For every name space in production, we check if we are over-sizing or under-sizing to maintain the right resource level."* Another participant stated, *"Our lambda-based services only run when needed."* Energy-aware scheduling, which schedules workloads when energy is green, is also practiced.

E. Content & Design: Organizations optimize their digital content and design to improve performance, user experience, sustainability, and accessibility by minimizing file sizes and network bandwidth using media compression and optimized images. They also streamline user journeys by eliminating unnecessary elements to reduce both cognitive and computational efficiency and improve overall efficiency. For example, one participant shared: *"We design the UI to minimize the number of steps users take to complete tasks."* Dynamic content loading techniques, such as lazy loading, are utilized to enhance efficiency, along with accessible design practices to ensure inclusivity and sustainability. Another participant shared, *"We use dynamic input so that only the required components*

are loaded per page."

F. DevOps practices: Four participants emphasized DevOps strategies to enhance the efficiency and sustainability of software development pipelines. Key practices include optimizing CI/CD processes, strategically deploying to avoid unnecessary builds and tests, and optimizing the size of deployment packages to reduce resource consumption during builds. For example, one participant shared: *"We managed to cut 30% of processor time used in the CI/CD build step by parallel execution of tasks."*

Interestingly, organizational culture emerged as a key enabler in some organizations in Finland and the EU. Awareness campaigns, internal guidelines, and educational initiatives are being implemented to promote sustainable thinking among developers, helping to embed green coding into long-term practice. To accelerate deliberate adoption, organizations can integrate green coding criteria into existing code review processes, as code reviews are commonly used in software development workflows.

5 Discussion

5.1 Bridging the State of Art and State of Practice

There is a clear convergence across various evidence sources regarding effective, high-leverage practices. The literature review and interviews highlighted recurring strategies such as database and query optimization, caching and memoization, minimizing API payloads, implementing lazy and asynchronous loading, utilizing modularization and microservices, and adopting auto-scaling and serverless architectures [19, 21, 23]. These techniques are appealing because they provide measurable performance improvements while reducing resource consumption and can be easily integrated into existing workflows. Similar to the findings of Wyoski [3] and Ahmed [20], this study confirms that awareness of energy-efficient coding is growing, yet adoption often follows performance or cost drivers rather than clear sustainability goals.

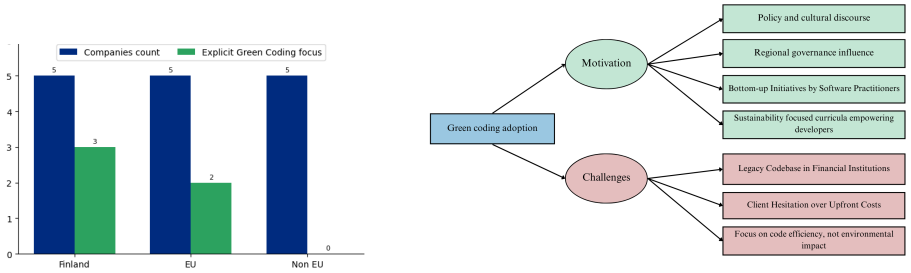
The Literature highlights the importance of programming language and compiler selection, as well as the potential of tools such as SPELL [34], GreenAdvisor [18], and LLM-based assistants. However, according to Luhikowski [16] and Pathiranage [15], our interviews show that these research tools are underutilized because of limited industrial integration and unclear incentives. It shows that the difference between theory and practice is not due to a lack of awareness but due to the absence of workflow-embedded tools and standards.

Adoption occurs where integration costs are low and benefits are visible. Common practices such as query tuning, cache design, and auto-scaling are often implemented. [19, 21]. In contrast, high-friction recommendations, such as language shifts or prototype tools, tend to be more time-consuming. These findings strengthen the earlier work of Junger et al [14], which notes that although the concepts of green coding are well articulated academically, there is little

evidence for empirical adoption. This study contributes by providing such an empirical basis in multiple areas and organizational contexts.

5.2 Regional Patterns in Adoption

The common ground of practices is evident across regions, although the importance varies with policy and cultural contexts. Finnish organizations, formed by strong sustainability goals and the CSRD frameworks [11], focus on code performance, event-driven design, and CI/CD optimization. Other EU organizations have more frequently experimented with carbon estimation tools and highlighted memoization, object pooling, concurrency, caching, and lazy loading [19, 21]. While non-EU organizations generally regard sustainability as a secondary outcome of engineering best practices, they rely on cloud-native provider tools and, in some instances, adjust deployment frequency as a strategic lever. These regional differences align with the findings of Pathiranage [15], who showed that regulatory pressure strongly drives sustainability integration, consistent with the influence of frameworks such as the CSRD [11] and the motivation patterns illustrated in Figure 3 (3a and 3b). Our findings align with Lohikoski [16], who noted that organizational structures and incentives shape sustainability adoption.



(a) Variation in green coding focus

(b) Motivations and challenges in green coding adoption

Fig. 3: Green coding practices

5.3 Organizational Dynamics: Bottom Up vs. Top Down

Interviews indicate that adoption typically occurs from the bottom up, particularly in large sectors, such as finance. Developers and technical leads incorporate sustainability checks into code reviews, promote right-sizing and idle shutdowns, and suggest energy-aware scheduling, even without formal mandates. This finding echoes Melo et al. [13], who observed that sustainability practices are often initiated by developers rather than management. Similarly, Lohikoski [16] identified the lack of organizational incentives and policies as a key barrier, a pattern consistent with our observations across several firms. Our proposed Capability-Constraint-Context (3C) model, mentioned below, builds on these earlier insights by providing a structured lens that connects individual initiative, organizational limitations, and environmental context. The 3C framing extends prior qualitative

frameworks, such as those by Pathiranage [15], by clarifying how team-level capabilities interact with constraints and contextual drivers to shape practical adoption and also aligns with the Technology–Organization–Environment (TOE) framework [37], in which capability reflects technological readiness, constraint parallels organizational inertia, and context mirrors environmental drivers. The 3C framework aligns with sociotechnical priorities and capability-based perspectives in sustainability research, linking the actions of developers at the micro-level with larger organizational structures and policy environments. For instance, a team's ability to incorporate energy-efficient CI/CD checks (Capability) relies on the tools they have access to and the priorities set by management (Constraints). Additionally, broader policy instruments, such as the CSRD, help define an enabling environment (context).

- **Capability (Team-Level)**: → When both know-how and workflow fit are strong (e.g., through PR templates and IDE/CI checks), low-friction practices can spread quickly.
- **Constraint (Organization-Level)**: → Legacy systems, compliance requirements, and the availability of skills limit the feasibility of implementing more profound interventions (e.g., language migration and advanced architectural optimization) [5, 16].
- **Context (Policy/Market)**: → In the EU, policy pressure promotes the adoption of measurement and review-embedded criteria, while non-EU contexts tend to focus more on provider efficiencies and portfolio strategies [11].

5.4 Practical Implications and Opportunities

The results highlight practical implications for key stakeholder groups and propose a clear framework for moving research into practice in academia and industry.

- **Academia (Translate Research into Workflow-integrated Tools):** Prioritize translation-based research by integrating results into workflow-native tools such as IDE plugins, pull request (PR) bots, and CI/CD processes that provide actionable insights (e.g., query selection checking, cache hit rate regression, and carbon estimation) [34,35]. Evidence from interviews confirms that workflow integration is an important factor in adopting indigenous delivery.

- **Industry (Embed Green Coding Standards in Code Review Templates:** Institutionalize effective practices by incorporating green standards into the evaluation and development workflow. Enhance code review templates and static analysis to address database integrity, caching efficiency, and payload budgets. Use performance dashboards such as *Lighthouse* and *Cloud Provider Matrix*, and integrate CO_2 estimators into CI pipelines to enable lightweight governance and continuous monitoring of sustainability goals [29].

- **Policymakers and Professionals (Develop Standardized Checklists):** The systematic literature review (SLR) highlights the need for standardized

practices similar to the Web Content Accessibility Guidelines (WCAG) [4]. Establishing a clear, checklist-style standard aligned with the Corporate Sustainability Reporting Directive (CSRD) would reduce compliance efforts, streamline procurement processes, and enable the definition of measurable key performance indicators (KPIs) [11].

– **Education (Integrate Sustainability into Core Computing/Software Engineering Curricula:** Integrate sustainability-focused content into software engineering/computing curricula and corporate training programs [8, 14, 25, 38], as developers often play a central role in driving sustainable change. This approach prepares future practitioners to apply evidence-based practical methods to develop green and energy-efficient software systems.

5.5 Challenges and Future Directions

The absence of standardized guidance remains a major barrier. Practitioners currently lack a unified framework or benchmark for green coding [8]. Legacy systems and modernization pose challenges, as technical debt and migration costs in sectors such as finance restrict the possibility of adopting new languages or toolchains, as well as implementing advanced optimization methods [5, 16]. In the short term, organizations should focus on incremental improvements, such as query tuning, cache design, payload reduction, auto-scaling, and idle shutdowns, which our data indicate are both practical and effective [19]. The lack of standardized guidance remains a major challenge. Practitioners do not have a collective reference point [8]. Creating guidelines for green coding similar to WCAG with explicit criteria, illustrative examples, and default benchmarks is a highly beneficial, low-risk measure, as noted in the SLR synthesis [4]. Many promising green coding tools are still in the prototype phase and have not yet been integrated into real workflows [34, 35]. To address this issue, future efforts should focus on conducting industrial trials of research tools embedded within IDEs and CI/CD pipelines, evaluating usability, the effort required for integration, and the trade-offs between performance and emissions [34]. Additionally, it is important to conduct quasi-experiments on legacy modernization to compare incremental refactoring against disruptive migration in terms of energy consumption, cost, and reliability [5]. Furthermore, the collaborative development of standards and incentives, such as procurement criteria and reporting requirements, that align sustainability with business objectives is essential [11]. Organizations are more likely to adopt low-friction, high-return practices that seamlessly fit into existing workflows, whereas deeper or experimental interventions necessitate stronger integration, a clearer ROI, and supportive policies. This study provides a comprehensive context-sensitive perspective on how research and practice align **(RQ1–RQ2)**, how regional and organizational factors influence adoption **(RQ3)**, and what is necessary to transform promising ideas into industry-ready solutions.

5.6 Threats To Validity

The research methodology has some limitations, as the small sample size and region-specific focus may limit the generalizability of the findings to broader

populations in a regional context. Likewise, Green coding can have varying definitions and interpretations depending on the organizational context, geography, and technical scope. Therefore, the potential threat lies in how the participants in the interviews understood and interpreted the term green coding. However, to mitigate this, a clear working definition of green coding was shown on the slides at the beginning of each interview, and participants were asked to describe it based on their technical scope within their organization.

6 Conclusion & Future Work

This study investigated the adoption of green coding practices in Finnish and global organizations through interviews with software professionals. The literature highlights a range of practices, including code-level optimization, sustainable architecture, content and UI design, and DevOps, as well as emerging tools, many of which remain conceptual or lack validation in industrial contexts. The findings indicate that while efficient coding is common globally, the explicit focus on green coding is mainly evident in European organizations, influenced by stricter corporate policies and regulations. Outside the EU, sustainability is generally an incidental outcome of prioritizing code quality and clean coding. To promote broader adoption, standardized, checklist-based guidance could provide practical and centralized directions for implementing green coding across organizations. Academia also plays a crucial role in embedding sustainability-focused education into computer science and software engineering programs. Bottom-up initiatives, especially in large organizations, demonstrate how software engineers can exemplify their voices to lead grassroots efforts, highlighting the importance of cultivating a sustainability mindset among future professionals.

References

1. R. Mehra, V. S. Sharma, V. Kaulgud, S. Podder, and A. P. Burden, "Towards a green quotient for software projects," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pp. 295–296, 2022.
2. T. Ojala, M. Mettälä, M. Heinonen, and P. Oksanen, "The ict sector, climate and the environment: Interim report of the working group preparing a climate and environmental strategy for the ict sector in finland," 2020.
3. W. Wysocki, "Why don't software companies care about software energy efficiency? a survey of software industry developers.," *Procedia Computer Science*, vol. 246, pp. 5054–5063, 2024.
4. S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proceedings of the 38th International Conference on Software Engineering*, pp. 225–236, 2016.
5. E. A. Jagroep, J. M. E. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper, "An energy consumption perspective on software architecture: A case study on architectural change," in *Software Architecture: 9th European Conference, ECSA 2015, Dubrovnik/Cavtat, Croatia, September 7-11, 2015. Proceedings 9*, pp. 239–247, Springer, 2015.

6. L. Karita, B. C. Mourão, L. A. Martins, L. R. Soares, and I. Machado, "Software industry awareness on sustainable software engineering: a brazilian perspective," *Journal of Software Engineering Research and Development*, vol. 9, pp. 2–1, 2021.
7. J. Gross and S. Ouhbi, "Clearing the path for software sustainability," *arXiv preprint arXiv:2405.15637*, 2024.
8. S. Oyedeji, M. A. Khan, P. Puhtila, O. Weerakoon, T. Mäkilä, M. O. Adisa, B. Naqvi, and S. Auvinen, "Green coding: State of practice," in *2025 11th International Conference on ICT for Sustainability (ICT4S)*, pp. 91–99, IEEE, 2025.
9. S. Oyedeji, B. Naqvi, M. O. Adisa, M. Abdulkareem, B. Penzenstadler, and A. Seffah, "The interplay between usability, sustainability and green aspects: A design case study from a developing country," 2019.
10. S. Mahmudova, "Green coding in programming and practices," *Journal of Engineering and Technology (JET)*, vol. 15, no. 2, 2024.
11. V. Abur, "The dimension of green coding in software quality control processes," in *2024 9th International Conference on Computer Science and Engineering (UBMK)*, pp. 1–6, IEEE, 2024.
12. S. Naumann, M. Dick, E. Kern, and T. Johann, "The greensoft model: A reference model for green and sustainable software and its engineering," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 4, pp. 294–304, 2011.
13. R. Melo, J. Vilela, C. Silva, M. Peixoto, and J. Araújo, "The brazilian practices for handling sustainability in software engineering: a replicated survey," in *Proceedings of the XXIII Brazilian Symposium on Software Quality*, pp. 298–308, 2024.
14. D. Junger, M. Westing, C. Freitag, A. Guldner, K. Mittelbach, S. Weber, S. Naumann, and V. Wohlgemuth, "Potentials of green coding-findings and recommendations for industry, education and science," 2023.
15. R. Pathiranage and H. Madrika, "Green information systems practices adoption in global it organisations: Exploring challenges, practices and strategies," 2024.
16. A. Lohikoski, "of thesis green software development-challenges and implementation ap," 2024.
17. V. Braun and V. Clarke, *Thematic analysis.* American Psychological Association, 2012.
18. I. Camargo-Henríquez, A. Martínez-Rojas, and G. Castillo-Sánchez, "Energy optimization in software: A comparative analysis of programming languages and code-execution strategies," in *2024 9th International Engineering, Sciences and Technology Conference (IESTEC)*, pp. 507–511, IEEE, 2024.
19. V. Majuntke and F. Obermaier, "Towards sustainable software," 2023.
20. R. Ahmad, "Software sustainability practices and awareness amongst software practitioners in malaysia: An exploratory study," in *2022 2nd International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS)*, pp. 192–197, IEEE, 2022.
21. S. Bhadane, V. Unde, A. Pathan, A. Kadam, G. R. Shinde, and P. N. Mahalle, "Sustainable programming: A new paradigm shift towards sustainable development," in *2024 IEEE Pune Section International Conference (PuneCon)*, pp. 1–6, IEEE, 2024.
22. H. S. Lella, R. Chattaraj, S. Chimalakonda, and M. Kurra, "Towards comprehending energy consumption of database management systems-a tool and empirical study," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, pp. 272–281, 2024.
23. P. Rani, J. Zellweger, V. Kousadianos, L. Cruz, T. Kehrer, and A. Bacchelli, "Energy patterns for web: An exploratory study," in *Proceedings of the 46th International*

Conference on Software Engineering: Software Engineering in Society, pp. 12–22, 2024.
24. S. R. Ahmad Ibrahim, J. Yahaya, and H. Sallehudin, "Green software process factors: a qualitative study," *Sustainability*, vol. 14, no. 18, p. 11180, 2022.
25. K. Oğuz, "Achieving sustainable software systems by reducing bloat and by promoting green practices in software engineering education," in *2024 Third International Conference on Sustainable Mobility Applications, Renewables and Technology (SMART)*, pp. 1–5, IEEE, 2024.
26. W. Wysocki, I. Miciuła, and P. Plecka, "Methods of improving software energy efficiency: A systematic literature review and the current state of applied methods in practice," *Electronics*, vol. 14, no. 7, p. 1331, 2025.
27. D. Di Pompeo and M. Tucci, "Harnessing genetic improvement for sustainable software architectures," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, pp. 248–249, IEEE, 2024.
28. Z. Taherikhonakdar and H. Fazlollahtabar, "Integrated delphi-interpretive structural modeling for sustainable green software development," *International Journal of Industrial Engineering*, vol. 35, no. 2, pp. 1–26, 2024.
29. K. Nayak, S. Route, M. Sundararajan, A. Jain, *et al.*, "Sustainable continuous testing in devops pipeline," in *2024 1st International Conference on Communications and Computer Science (InCCCS)*, pp. 1–6, IEEE, 2024.
30. K. Letrache, O. El Beggar, and M. Ramdani, "Green data warehouse design and exploitation," in *Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications*, pp. 1–6, 2018.
31. G. Procaccianti, H. Fernández, and P. Lago, "Empirical evaluation of two best practices for energy-efficient software development," *Journal of Systems and Software*, vol. 117, pp. 185–198, 2016.
32. A. Atadoga, U. J. Umoga, O. A. Lottu, and E. Sodiy, "Tools, techniques, and trends in sustainable software engineering: A critical review of current practices and future directions," *World Journal of Advanced Engineering Technology and Sciences*, vol. 11, no. 1, pp. 231–239, 2024.
33. M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "Multi-objective optimization of energy consumption of guis in android apps," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 1–47, 2018.
34. R. Pereira, T. Carção, M. Couto, J. Cunha, J. P. Fernandes, and J. Saraiva, "Helping programmers improve the energy efficiency of source code," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 238–240, IEEE, 2017.
35. S. Gupta and A. Gupta, "Advancing carbon-efficient software development: A sustainable path forward," in *Proceedings of the 26th International Conference on Distributed Computing and Networking*, pp. 325–330, 2025.
36. Z. Ournani, R. Rouvoy, P. Rust, and J. Penhoat, "On reducing the energy consumption of software: From hurdles to requirements," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–12, 2020.
37. L. Tornatzky and M. Fleischer, "The process of technology innovation, lexington, ma," 1990.
38. A. Moreira, P. Lago, R. Heldal, S. Betz, I. Brooks, R. Capilla, V. C. Coroamă, L. Duboc, J. P. Fernandes, O. Leifler, *et al.*, "A roadmap for integrating sustainability into software engineering education," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–27, 2025.