Developer Views on Software Carbon Footprint and its Potential for Automated Reduction

Haozhou Lyu[1], Gregory Gay[1][0000−0001−6794−9585], and Maiko Sakamoto[2]

[1] Chalmers | University of Gothenburg, Gothenburg, Sweden
haozhou@student.chalmers.se, greg@greggay.com
[2] University of Tokyo, Tokyo, Japan
m-sakamoto@k.u-tokyo.ac.jp

Abstract. Reducing software carbon footprint could contribute to efforts to avert climate change. Past research indicates that developers lack knowledge on energy consumption and carbon footprint, and existing reduction guidelines are difficult to apply. Therefore, we propose that automated reduction methods should be explored, e.g., through *genetic improvement*. However, such tools must be *voluntarily adopted and regularly used* to have an impact.
In this study, we have conducted interviews and a survey (a) to explore developers' existing opinions, knowledge, and practices with regard to carbon footprint and energy consumption, and (b), to identify the requirements that automated reduction tools must meet to ensure adoption. Our findings offer a foundation for future research on practices, guidelines, and automated tools that address software carbon footprint.

Keywords: Carbon Footprint · Energy Consumption · Sustainability · Genetic Improvement · Genetic Programming

1 Introduction

The carbon dioxide emitted through development and use of software may contribute to climate change. In 2015, data centers accounted for an estimated 3.00% of global energy consumption—double that of the United Kingdom and matching the airline industry [2]. Training a single neural network can emit as much carbon as the entire lifetime of five cars [8]. That carbon footprint *must be reduced*, but this is not a straight-forward task. There are sources of emissions at multiple stages of development, produced through development and use of software [18]. Further, while carbon footprint is largely a product of energy consumption, the *quantity*, *sources*, and *location* of consumption are important.

Researchers have begun to make recommendations on how to reduce carbon footprint (e.g., [10, 19, 22]). Such guidelines are highly important, but can be difficult to apply—especially after the code has been written. Further, it is not clear that developers have a clear understanding of carbon footprint, energy consumption, or how either can be reduced during development [12–14].

Therefore, we are interested in exploring *automated* reduction of carbon footprint. There are multiple stages of development that could benefit from such reduction—e.g., design, implementation, or maintenance—and multiple practices or development artifacts that could be optimized—e.g., source code, processes, or design models. To ground this study, we focus on one example—where a tool would transform the existing source code of the project-under-development to reduce the carbon footprint caused by the usage of the software. This tool could take measurements, e.g., of energy consumption, data transfer, and other factors related to the carbon footprint—measured during test case execution—to assess the impact of the attempted transformations.

Such transformations should preserve semantic meaning while reducing the carbon footprint by, e.g., reducing energy consumption or controlling where energy is consumed. A promising technique to impose such changes is *genetic improvement* (GI) [3], where populations of program patches are scored according to qualities of interest, then evolved over many generations to maximize or minimize these scores. GI has been applied to reduce energy consumption (e.g., [3, 6, 10, 11, 20]), and other researchers have recently suggested the potential of GI with regard to carbon footprint reduction [7, 15]. We, similarly, propose that such approaches could be extended, or new approaches could be developed, specifically to target carbon footprint.

The development of such tools could improve the sustainability of the IT industry. However, such tools *must be voluntarily adopted and regularly used by developers.* The goals of this study are (a) to explore developers' existing opinions, knowledge, and practices in this area, and (b), to identify the requirements that automated tools must meet to ensure voluntary adoption. We conducted a set of initial interviews, followed by a broader survey of software developers, and performed thematic and quantitative analyses of the collected data.

This study provides a foundation for future research by exploring requirements for automated carbon footprint reduction tools. We also offer insights to those interested in the existing opinions, knowledge, and practices of developers. To help enable future research, we also make a replication package available [9].

2 Background and Related Work

Carbon Footprint and Energy Consumption of Software: Carbon footprint is the total quantity of carbon dioxide emissions associated with an individual, product, or organization's activities. This can include direct (e.g., fuel consumed during production) or indirect emissions (e.g., energy consumption) [21].

Software is a source of carbon emissions.There are sources of both direct and indirect emission at multiple phases of development, including implementation, testing, delivery, usage, and maintenance [18]. In this research, our scope is primarily restricted to indirect emissions associated with energy consumption during software *usage*—i.e., when interactions take place with the software.

Within this scope, carbon footprint is affected both by the *quantity* of energy consumed and *where and how* that energy is produced or consumed, as some energy sources have a greater carbon footprint than others. Calculating and reducing software carbon footprint is not simple, as it is affected by energy usage on both the client-side (i.e., on consumer devices) and server-side (i.e., at data centers in disparate geographic areas), as well as on network transmissions between the two [1]. Automated approaches must consider not just energy quantity, but also aspects such as the location of computing elements.

Genetic Improvement: GI is the automated improvement of non-functional qualities (e.g., performance) of software through transformations to the source code [3]. Population of patches are produced and then judged using one or more fitness functions related to the qualities of interest. The patches that yield the best scores form a new population through mutation—where stochastic changes are introduced—or crossover—where aspects of "parent" patches combine to form new "children". Carbon footprint can be considered a quality, improved through appropriate program transformations and fitness functions.

Related Work: Researchers have found that most programmers are largely unaware of energy consumption, lack knowledge on the causes or how to measure consumption, and rarely address energy issues [13, 14, 12]. Some developers even regard green software design as a "threat" that could disrupt their workflow [12]. Our study yields similar findings on energy consumption, but extends our understanding with regard to developer opinions on carbon footprint, current practices regarding both carbon footprint and energy consumption, and opinions on automated improvement tools.

Past research has offered guidelines on how to reduce energy consumption and carbon footprint (e.g., algorithm selection [10], code structure [22], considering server distribution and location [10], or controlling image quality [19]). Such guidelines are highly important, but are not always easy to apply. Nor is it simple to manually improve code after it has been written. Therefore, we are interested in automated carbon footprint reduction techniques.

To date, we are unaware of any automated tools specifically targeting carbon footprint. However, there have been several approaches targeting energy consumption, mostly based on genetic improvement (e.g., [3, 20, 11, 10, 6]). Other approaches include specialized compilers [16] and data migration strategies [5]. We hypothesize that GI can also reduce carbon footprint, potentially by extending existing approaches to consider both client and server-side components and additional fitness functions. Other researchers have also recently suggested the use of GI in this area, e.g., to improve Machine Learning frameworks [7, 15].

3 Methodology

Our study is guided by the following research questions:

- **RQ1:** What knowledge do developers have about the carbon footprint or energy consumption of software?
- **RQ2:** How do developers assess and control the carbon footprint or energy consumption of their software?
- **RQ3:** What requirements and constraints must be satisfied for developers to trust carbon footprint reduction tools?
- **RQ4:** How should a reduction tool fit into the development workflow?
- **RQ5:** How can voluntary adoption of reduction tools be encouraged?

Table 1: Demographic information on interviewees, including location, position, job responsibilities (self-described), and development experience (years).

ID	Country	Position	Responsibility	Exp.
P1	Sweden	Manager	Overlook technical road maps	25
P2	Japan	Developer	Data analysis and development	5
P3	Sweden	Student	Developer testing televisions	4
P4	Sweden	Manager	Technical strategy and development process	20
P5	Japan	Developer	Develops software	5
P6	Japan	Developer	Network operation and maintenance tools	4
P7	Japan	Developer	Service planning and development	7
P8	Japan	Researcher	AI and robotics development	6
P9	Sweden	Student	Machine Learning development	4
P10	Sweden	Developer	C software development	4

To answer these questions, we conducted semi-structured interviews, then performed thematic analysis following Cruzes and Dyba's guidelines [4], to gain an initial understanding. Then, based on the interview results, we developed a survey to gain additional insights from a broader range of participants.

We do not collect personal information, but rather data on participants' perceptions and practices. All collected information was fully anonymized. Participation in the study was voluntary. Given these considerations, obtaining ethical approval was deemed unnecessary at our institutes.

3.1 Interviews

Population Definition: Our population consists of participants with experience developing software, including professionals and university students studying a related discipline.
Sampling: We interviewed 10 participants. The sampling method was a mix of purposive and convenience sampling. The professionals were gathered from companies in Sweden and Japan using LinkedIn, as well as through personal contacts. After 10 interviews, we had achieved result saturation.
Demographics: Table 1 shows information on participants. To maintain confidentiality, we omit participants' names. These participants come from various roles, with experience ranging from 4–25 years, and experience in a variety of domains (e.g., robotics, machine learning, web applications).
Interview Guide: The interview questions can be found in our replication package [9]. The questions were open-ended, so we could ask follow-up questions if needed, while ensuring we answered the core research questions. The 13 questions were divided into three sections: (1) prior knowledge, (2) experiences and opinions, and (3), requirements for automated carbon footprint reduction tools.
Data Collection: During interviews, we introduced the background and purpose of the research. We then conducted the semi-structured interview. Following completion, we answered their questions and shared information on the project. From November to December 2022, all interviews were conducted online and lasted between 20 and 30 minutes. Participants were interviewed in English. To analyze the results, we recorded both video and audio. We transcribed our records using a denaturalism approach. Transcriptions were performed using a speech-to-text tool. We referred to the recordings to make clarifications.

Table 2: Survey respondent demographics.

ID	Country	Position	Software Domain	Experience
I1	China	Developer	Embedded system	1-3 years
I2	Japan	Developer	Sever web service	1-3 years
I3	Denmark	Researcher	Web applications	9 years+
I4	Japan	Developer	Internal tools	4-6 years
I5	Sweden	Developer	Embedded system	7-9 years
I6	Ireland	Researcher	Programming environment	9 years+
I7	Japan	Manager	Web applications	7-9 years
I8	Sweden	Student	Embedded system	4-6 years
I9	UK	Developer	Web applications	<1 year
I10	China	Student	Embedded system	1-3 years
I11	UK	Developer	Web applications	1-3 years
I12	Sweden	Student	Various	1-3 years
I13	USA	Developer	Web applications	<1 year
I14	Sweden	Manager	Analytics	9 years+
I15	Sweden	Developer	Web and desktop applications	7-9 years
I16	Romania	Developer	Web and desktop applications	1-3 years
I17	Sweden	Developer	Data analytics	1-3 years
I18	Sweden	Developer	Various applications	1-3 years
I19	Sweden	Manager	Web applications	9 years+
I20	Sweden	Developer	Data analysis application	4-6 years
I21	Sweden	Developer	Web applications	9 years+
I22	India	Manager	Web applications	9 years+
I23	Ireland	Developer	Windows and Web applications	9 years+
I24	Sweden	Manager	Enterprise software	9 years+
I25	Sweden	Manager	Business intelligence	9 years+
I26	Sweden	Developer	Visual analysis software	9 years+
I27	Sweden	Developer	Visual analysis software	9 years+
I28	Sweden	Developer	On-prem client, cloud service	9 years+
I29	France	Developer	Web applications	4-6 years
I30	Japan	Developer	Mobile mini applications	<1 year
I31	Japan	Developer	Embedded system	1-3 years
I32	Japan	Developer	System software	1-3 years
I33	Japan	Developer	Web applications	<1 year
I34	Japan	Developer	Web applications	1-3 years
I35	Japan	Developer	Cloud web applications	1-3 years
I36	Japan	Developer	Web applications	1-3 years
I37	Japan	Developer	Artificial Intelligence	1-3 years
I38	Japan	Developer	Web applications	1-3 years
I39	Japan	Unemployed	N/A	1-3 years
I40	Sweden	Developer	SAAS	1-3 years

Data Analysis: We adopted thematic coding. We familiarized ourselves with the data by reading the transcript repeatedly and identifying relevant segments (codes). These codes were organized and aggregated into themes and sub-themes. After each interview, we modified the codes and themes, and paused for discussion. We stopped when no new codes were found in transcripts. Attention was paid to ensuring that each code was accurate to the original response.

Coding was conducted by the first author. However, the second author independently coded one interview to ensure reliability. As only minor differences were observed, coding of the remaining interviews proceeded.

3.2 Survey

Population and Sampling: Our population and sampling methods are the same as in the interviews. We sent the questionnaire directly to some participants, and also distributed it on LinkedIn, Facebook, Twitter, and Mastadon. Between November–December 2022, 40 respondents completed the survey.

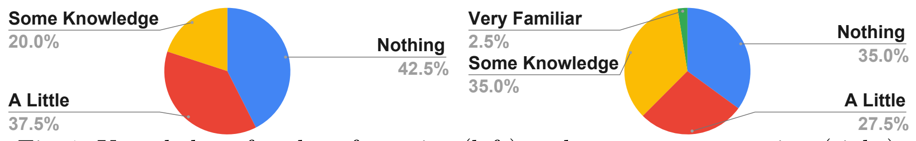


Fig. 1: Knowledge of carbon footprint (left) and energy consumption (right).

Participant Demographics: Table 2 shows demographic information. The participants come from various countries—although most are still from Japan and Sweden—with varied roles, experience levels, and development domains.
Survey Guide: The survey questions are in our replication package [9]. To ensure a high response rate, the survey was designed to be as brief—lasting between 10-15 minutes. The 26 questions consist of open-ended, multiple choice, ordinal scale, and interval scale questions [17].
Data Collection: The partially-structured questionnaire design method was adopted to ensure participants had freedom to express their opinions. The questionnaire is divided into three parts, mapped to the same topics as the interviews.
Data Analysis: We use descriptive statistics to analyze quantitative data. Qualitative data was incorporated into our previous thematic mapping.

4 Results and Discussion

In this section, we answer the research questions using data from both the interviews and survey.

4.1 Existing Knowledge (RQ1)

As shown in Figure 1, 80% of survey participants are either unfamiliar with or only have a little knowledge on carbon footprint. Participants had somewhat more knowledge on energy consumption, but 63% still had no or little knowledge. Encouragingly, however, many interviewees had—at least—basic knowledge of factors that impact energy consumption or carbon footprint. For example:

> "That's not something that I think about daily. The only thing I can think about is all the things that we store on service, in the cloud, of course, those computers need electricity. There's been a lot of talk about mining cryptocurrency. It's not a sustainable way of handling money." - **P4**

Another noted how sources of energy affect carbon footprint:

> "[Carbon footprint] depends on where the energy comes from. If the energy is carbon neutral, then the footprint is still small, even if you consume lots of energy ... Most of our customers are not in Sweden, but most of our development is in Sweden. I would say majority of the energy in Sweden is not carbon-based ... water, wind, nuclear, and so on." - **P1**

As shown in Figure 2, as the experience of developers grows, there is also some increase in their median level of knowledge on both topics. Over time, developers tend to acquire knowledge of specialized topics and be more confident in their knowledge. The median knowledge of carbon footprint remains at a relatively low level—between "little" and "some" knowledge—but does rise. The median level of knowledge on energy consumption rises more noticeably to "some" knowledge.

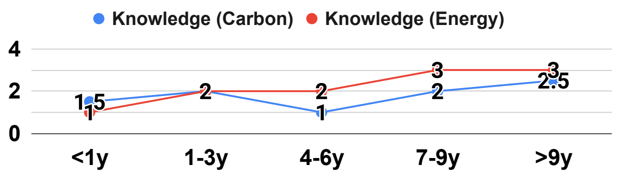


Fig. 2: Median knowledge versus experience (1="Nothing", 4="Very Familiar").

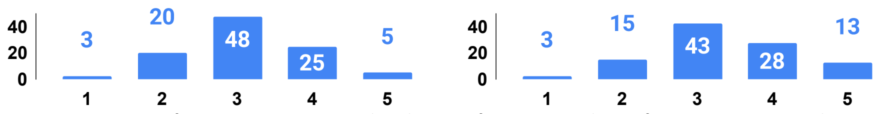


Fig. 3: Degree of agreement on whether software carbon footprint contributes to climate change (left) and whether software carbon footprint should be considered and controlled (right) (1 = "strongly disagree", 5 = "strongly agree").

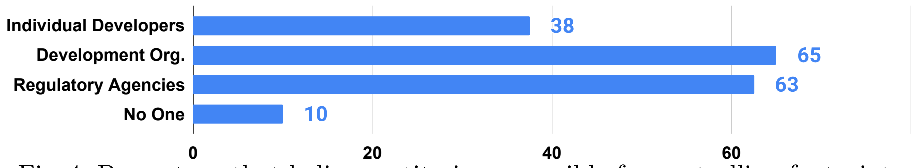


Fig. 4: Percentage that believe entity is responsible for controlling footprint.

4.2 Developer Opinions and Practices (RQ2)

Developer Opinions: Figure 3 (left) shows a largely balanced view on whether software carbon footprint contributes to climate change, with a plurality (48%) expressing a neutral view. However, somewhat more participants agree (30%) than disagree (23%). Figure 3 (right) also shows that more participants (40%) agree that carbon footprint should be considered than disagree (18%). Again, however, a plurality are neutral (43%). In both cases, there was no discernible change in opinion as developers gained experience.

Figure 4 shows participants' views on who holds responsibility for considering or controlling energy consumption or carbon footprint. Participants could select more than one option. The majority of respondents felt that both organizations (65%) and regulatory agencies (63%) should bare responsibility. Only 10% believed that no one holds responsibility. Interviewees noted that developers must comply with the company's development rules, business goals, and release criteria. Therefore, they may not have the option of reducing energy consumption.

We suggest that development organizations could take a larger responsibility by raising awareness among their staff, e.g., hosting seminars or training programs that teach sustainable development practices. Companies could also prioritize energy consumption or carbon footprint as part of project goals and impose policies to control their impact on the environment. Regulatory agencies can also formulate stronger policies regarding the IT industry.

Development Stages: Carbon footprint and energy consumption can be considered during multiple development stages. Interviewees considered both during design, testing, and maintenance. During design, developers estimate the num-

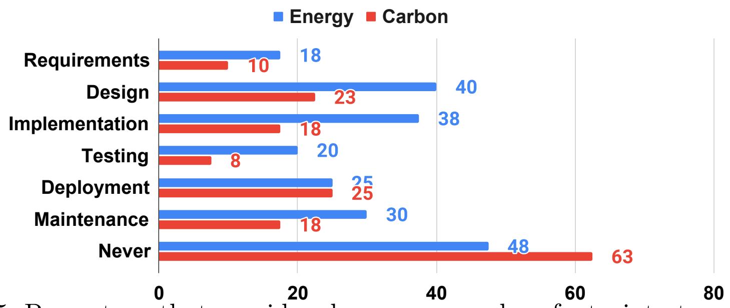


Fig. 5: Percentage that considered energy or carbon footprint at a stage.

ber of end users and specify resources (e.g., number and capability of servers):

> "During the design phase, we're going to make some estimations. We're going to decide what technologies are we going to use, how many servers we are going to have, what is the load that we are expecting." - **P2**

Additionally, problematic energy consumption can be observed and mitigated after the system is deployed:

> "... When we actually run something in production and see that ... we are overloading the systems, we need to do something about that." - **P3**

Figure 5 illustrates when survey participants considered energy consumption and carbon footprint. In both cases, the largest percentage of participants—48% for energy consumption and 63% for carbon footprint—never took action. For energy consumption, the most consideration is given during design (40%) and implementation (38%) phases. During design, decisions are made on the system architecture, which often must incorporate consideration of energy. These decisions are realized during implementation.

Many also considered carbon footprint during design (23%). However, deployment (25%) was the most common phase for carbon footprint. Gaining an accurate estimation of carbon footprint may be difficult before the system is in operation, where usage statistics can be gathered as well as knowledge of where users and data centers are located. Some energy decisions can be made without such information, so energy consumption could be considered early in development to also limit carbon footprint. Once the system is deployed, statistics on carbon footprint can be gathered and changes can be made, if needed.

Automated tools could be deployed at different stages to improve various design and implementation artifacts. The example we proposed—automated source code transformation to reduce carbon footprint from software usage—would be used during the implementation, deployment, or maintenance phases—i.e., any point where source code exists and appropriate measurements can be gathered.

Actions Taken: Most interviewees have not taken concrete actions to address energy or carbon footprint. However, many had reduced resource usage, e.g., by compressing elements or bypassing unnecessary interactions:

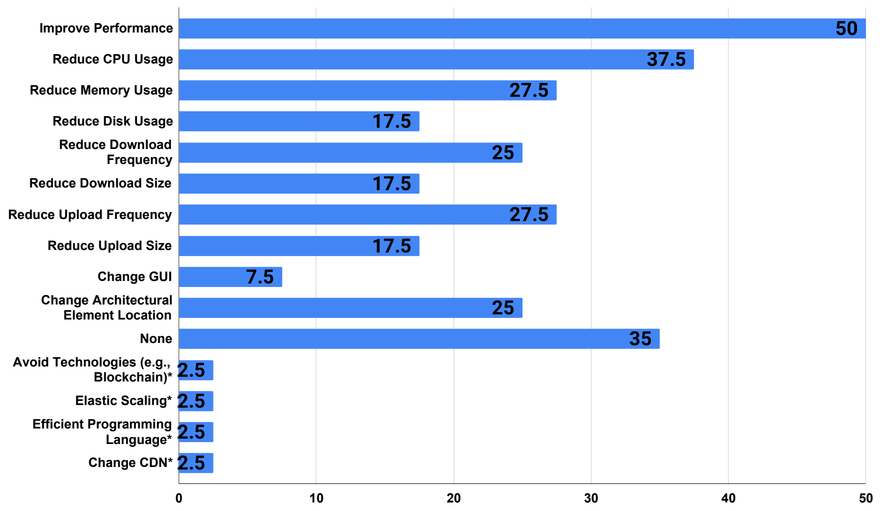


Fig. 6: Percentage of participants that took actions to reduce energy consumption (* indicates a participant-added response).

> "To make the web pages faster, or save time or resources, we reduced our resources, e.g., we compressed images or music, or [added] some easier way to click to the bottom or go to the next pages." - **P6**

These actions indirectly affect carbon footprint:

> "If I'm a good engineer ... I indirectly reduce cost, indirectly improve code, indirectly reduce carbon footprint." - **P5**

Figure 6 illustrates the percentage of participants that took certain concrete actions to reduce energy consumption. The most common practice was to improve performance (50%), followed by reducing CPU (38%) and memory consumption (28%) or data upload frequency (28%). Survey participants were also asked what actions they had taken to reduce carbon footprint. They discussed energy and resource consumption, mentioning low-power device states, elastic scaling, memory consumption, and data reuse through instances. Many of these same actions could be invoked by automated tools to improve the software. Performance improvement is already a common target of GI tools [3].

Another participant noted actions developers can take to reduce the footprint of the development process:

> "I have taken steps to reduce energy needed for developing software (by shutting down unused machines and reducing background processes and other things on development machines)." - **I21**

Measurement and Evaluation: While most interviewees have not directly evaluated energy or carbon footprint, several evaluate cost and resource usage:

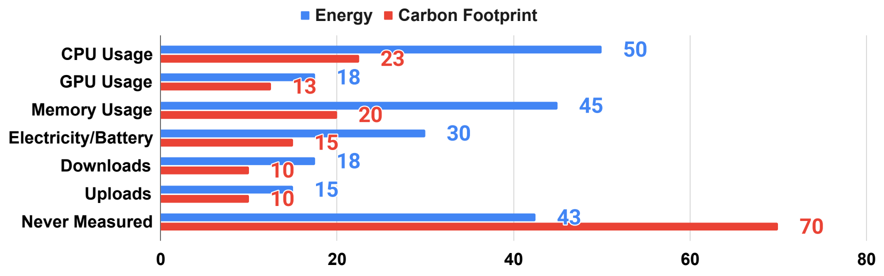


Fig. 7: Percentage that applied a particular measurement.

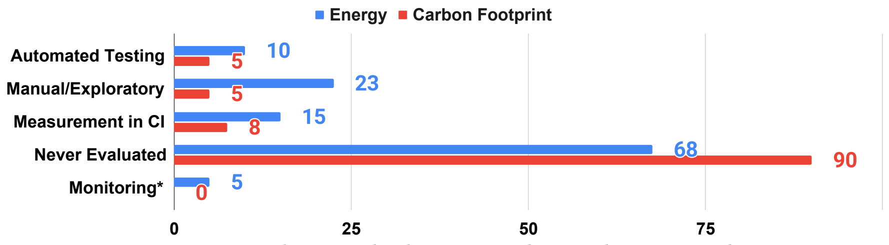


Fig. 8: Percentage that applied a particular evaluation technique.

> "We don't evaluate energy consumption, but we evaluate cost, which is directly related ... [To reduce] cost of service, we design systems ... which are more efficient, consume less cost, and improve user experience." - **P8**

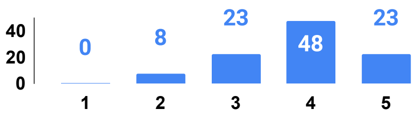
Figure 7 shows the direct or indirect measures survey participants have used to assess energy or carbon footprint. Many have never measured either (43% for energy, 70% for carbon). Among the respondents who measured either, CPU (50%, 23%) and memory usage (45%, 20%) were the most common methods. These measurements could be used by automated reduction methods to assess potential code transformations.

As shown in Figure 8, most have also never formally evaluated either (68% for energy, 90% for carbon). For energy consumption, the most common method was manual or exploratory testing (23%), followed by measurement during CI (15%). For carbon footprint, the most common method was during CI (8%). Code transformation could be implemented as part of the CI pipeline, or could be manually invoked as long as executable test cases or other ways of simulating software usage exist.

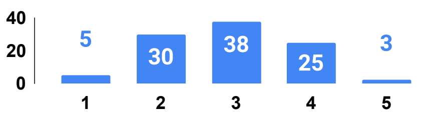
4.3 Requirements for Automated Tools (RQ3)

During the interviews, all participants expressed a positive attitude toward the idea of an automated carbon footprint reduction tool and regarded the concept as exciting and potentially helpful:

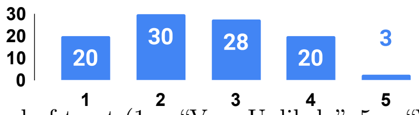
> "We want to reduce the amount of power our software requires. ... it's part of the green message we need to send to the world ... I don't really like bitcoin mining and that kind of thing, so I like what the theory is doing, changing to a much more energy-reduced algorithm for the mining." - **P1**



(a) Willingness (1 = "Strongly Unwilling", 5 = "Strongly Willing").



(b) Skepticism (1 = "Highly Skeptical", 5 = "Highly Unskeptical").



(c) Likelihood of trust (1 = "Very Unlikely", 5 = "Very Likely").

Fig. 9: Willingness, skepticism, and likelihood to trust tool results.

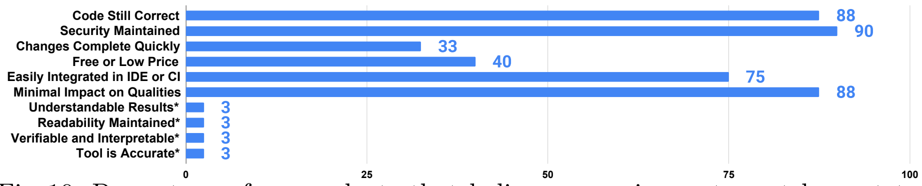


Fig. 10: Percentage of respondents that believe a requirement must be met to trust an automated tool (* indicates a participant-added response).

70% of survey participants were willing or strongly willing to try a tool (Figure 9(a)), and only 8% were unwilling. However, Figure 9(b) shows that 35% were either skeptical or highly skeptical of the ability of a tool to successfully reduce carbon footprint, and a further 38% had neutral expectations.

Figure 9(c) indicates that participants may not trust a tool to modify their code, with 50% either unlikely or very unlikely to trust the tool, and a further 28% neutral. Only one respondent indicated that they would be very likely to trust a tool. A participant's experience had little impact on willingness or skepticism. However, participants with <3 years of experience were less likely (median of 2.00) to trust than those with more experience (median 3.00).

Some skepticism seems reasonable. Even if a tool could reduce carbon footprint without supervision, the changes it made could result in incorrect behavior or reduction in other qualities. Care must be taken to prevent such an occurrence.

Interview and survey respondents were asked about the requirements that would have to be met to trust a tool. Among survey participants (Figure 10), the top requirements are that security is maintained (90%), that the code still operates correctly (88%), and that there is no—or minimal—negative impact on other qualities (88%). Security is one of the most significant qualities of software, with major legal and financial implications. Tools should also not introduce faults, and reducing carbon footprint may not a priority if it comes at the expense of qualities such as performance.

Others asked that the tool be easily integratable into a CI pipeline, be available for free or a low price, and complete changes quickly:

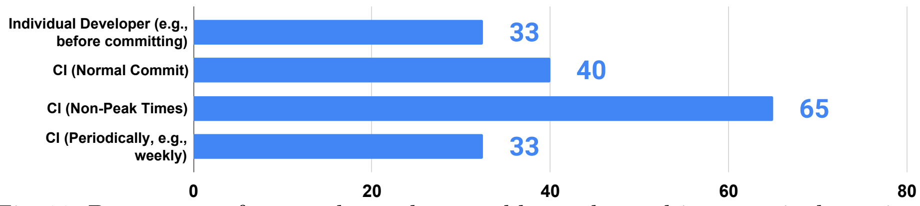


Fig. 11: Percentage of respondents that would use the tool in a particular point in the development workflow.

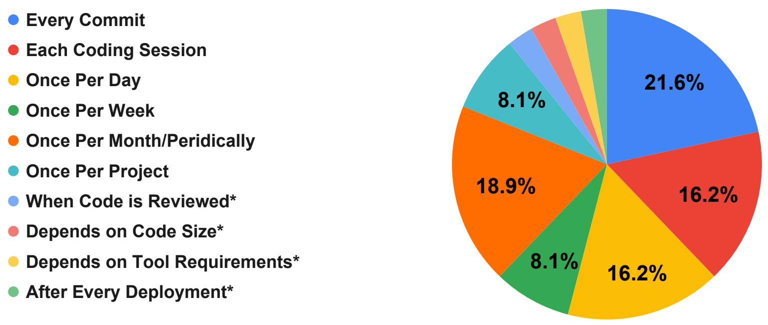


Fig. 12: Frequency that survey participants would apply the tool (* indicates a participant-added response, each 2.5%).

> "If I think I can trust it, it depends on how well it will fit into our development processes and how easy it is to integrate it ... If it's very expensive, then you have to weigh that against how much are we willing to spend on detecting energy consumption." - **P1**

To support integration into CI, the tool should offer an API or CLI, and should be installable through a package manager, (e.g., `pip`).

Further, the tool should produce understandable code and interpretable results—that is, users should understand how and why changes were made. For example, the tool could provide a report with an explanation of code changes and impact on carbon footprint. Such data and rationale can enable verification and trust.

4.4 Use in Development Workflow (RQ4)

Participants were asked about how (and how often) they would apply a tool. Many were interested in integrating this tool into an existing CI pipeline. As shown in Figure 11, 65% of participants would apply the tool in CI during non-peak times (e.g., overnight) to not interfere with normal development and to increase the likelihood that code is in a working state. The tool could also take more time if results are not needed quickly. 40% would apply the tool as part of CI after a normal commit. 33% of respondents would also apply the tool periodically in CI or before committing code.

Many indicated (Figure 12) that they would apply the tool at a relatively high frequency—split between after every commit (22%), after every coding session (16%), or daily (16%). However, a significant portion would instead apply it periodically (19%), possibly to allow code to stabilize before being improved.

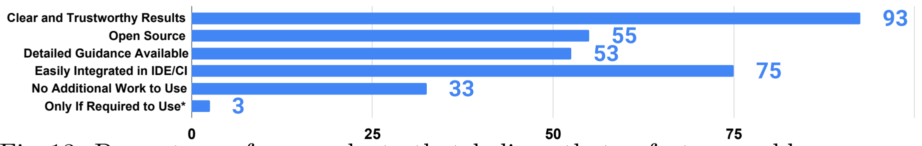


Fig. 13: Percentage of respondents that believe that a factor would encourage voluntary adoption (* indicates a participant-added response)

4.5 Voluntary Adoption (RQ5)

Participants were also asked how to encourage voluntary adoption of tools that reduce software carbon footprint. The views of survey participants are shown in Figure 13. The most important factor is that the tool shows a track record of clear and trustworthy results (93%). Testimonials from existing users could encourage adoption. One interviewee noted:

> "I will wait until the wider community accepts it. First, I will see if this is doing good. I will try to understand as much as possible what it is doing before I start using this tool."- **P6**

The tool could offer a dashboard or produce a report that visualizes changes to the source code, CPU usage, RAM usage, and carbon footprint so developers can easily understand the changes made.

Easy integration of the tool was essential for 75% of the participants. 33% asked that no additional work be needed to use the tool. An interviewee added:

> "I think the main constraint would be that it does not use a lot of resources and ... should not interrupt any development work. It's okay if it takes longer to analyze once the changes are pushed, because it can just run overnight and I don't need to worry about that. But if I use it during development, I would like it to not affect any of my development experience." - **P7**

If the tool is open-source, those who are interested can gain a better understanding of its algorithm and methods. A community could form that expands the tool's capabilities. Additionally, detailed documentation should provide clear and understandable instructions on how to install and use the tool.

5 Threats to Validity

Conclusion Validity: The number of responses may affect conclusion reliability. However, our thematic findings reached saturation. Further interviews or surveys could enrich our findings, but may not produce significant additions.
Construct Validity: The interviews or survey could have missing or confusing questions. There is also a risk that participants may not be familiar with particular terminology. We provided an introduction to reduce this risk. The use of semi-structured interviews allowed us to ask follow-up questions. We also conducted pre-testing of the interview and survey.
External Validity: Generalizability of our findings is influenced by the number and background of participants. Our participants represent many development roles, experience levels, and product domains. Therefore, we believe that our

results are relatively applicable to the software development industry. We also contrast results between Sweden and Japan.

Internal Validity: Thematic coding suffers from known bias threats. We mitigated these by performing independent coding and comparing results, finding sufficient agreement. We make our data available, increasing transparency.

6 Conclusions

This study provides a foundation for future research addressing software carbon footprint. We found that many developers lack knowledge. However, some had basic understanding of energy-related factors, and knowledge grows with experience. A plurality were neutral on whether software contributes to climate change and whether carbon footprint should be controlled. However, more agree than disagree with both. The majority feel that both development organizations and regulatory agencies bare responsibility for controlling these factors. Attention should be paid in future work to increasing developer awareness and exploring policy implications of software carbon footprint.

In current practice, energy is considered most often during design and implementation, and is reduced by improving performance or resource consumption. Carbon footprint is considered during deployment and design. Both energy and carbon footprint are most commonly measured through CPU or memory usage. Energy consumption is often evaluated using manual or exploratory testing, carbon footprint through measurements during Continuous Integration (CI).

The majority are willing to try an automated carbon footprint reduction tool. Developers would apply the tool as part of CI, generally on a regular basis (e.g., daily—possibly at non-peak times). However, many were skeptical of the potential results. To overcome such skepticism, researchers should also explore automated carbon footprint reduction solutions that meet the needs identified by the developers. Such tools must not compromise security, correctness, or other important qualities. They also must integrate into a CI pipeline, be well-documented, be reasonably priced or—preferably—open source, and offer transparent and trustworthy results.

References

1. A. S. Andrae. New perspectives on internet electricity use in 2030. *Engineering and Applied Science Letters*, 3(2):19–31, 2020.
2. T. Bawdin. Global warming: Data centres to consume three times as much energy in next decade, experts warn. *The Independent*, 2016.
3. B. R. Bruce, J. Petke, and M. Harman. Reducing energy consumption using genetic improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1327–1334, 2015.
4. D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*, pages 275–284. IEEE, 2011.
5. V. De La Luz, M. Kandemir, and I. Kolcu. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *Proceedings 2002 Design Automation Conference (IEEE Cat. No. 02CH37324)*, pages 213–218. IEEE, 2002.

6. J. Dorn, J. Lacomis, W. Weimer, and S. Forrest. Automatically exploring tradeoffs between software output fidelity and energy costs. *IEEE Transactions on Software Engineering*, 45(3):219–236, 2017.
7. S. Georgiou, M. Kechagia, T. Sharma, F. Sarro, and Y. Zou. Green ai: Do deep learning frameworks have different costs? In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1082–1094, New York, NY, USA, 2022. Association for Computing Machinery.
8. K. Hao. Training a single AI model can emit as much carbon as five cars in their lifetimes. *MIT Technology Review*, 2019.
9. H. Lyu, G. Gay, and M. Sakamoto. Replication Data for "Developer Views on Software Carbon Footprint and its Potential for Automated Reduction", Feb. 2023. https://doi.org/10.5281/zenodo.7597662.
10. I. Manotas, L. Pollock, and J. Clause. Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*, pages 503–514, 2014.
11. V. Mrazek, Z. Vasicek, and L. Sekanina. Evolutionary approximation of software for embedded systems: Median function. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 795–801, 2015.
12. 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)*, pages 1–12, 2020.
13. C. Pang, A. Hindle, B. Adams, and A. E. Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–89, 2015.
14. G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31, 2014.
15. F. Sarro. Search-based software engineering in the era of modern software systems. *Proceedings of the 31st IEEE International Requirements Engineering Conferece. IEEE*, 2023.
16. S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 409–415. IEEE, 2002.
17. H. Taherdoost. How to design and create an effective survey/questionnaire; a step by step guide. *International Journal of Academic Research in Management (IJARM)*, 5(4):37–41, 2016.
18. J. Taina. How green is your software? In *International Conference of Software Business*, pages 151–162. Springer, 2010.
19. N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery? analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50, 2012.
20. D. R. White, J. Clark, J. Jacob, and S. M. Poulding. Searching for resource-efficient programs: Low-power pseudorandom number generators. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1775–1782, 2008.
21. T. Wiedmann and J. Minx. A definition of 'carbon footprint'. *Ecological economics research trends*, 1(2008):1–11, 2008.
22. Y. Zhu and V. J. Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24. IEEE, 2013.