On Reducing the Energy Consumption of Software: From Hurdles to Requirements

Zakaria Ournani, Romain Rouvoy, Pierre Rust, Joël Penhoat

On Reducing the Energy Consumption of Software: From Hurdles to Requirements

Zakaria Ournani
Orange Labs/ Inria / Univ. Lille
zakaria.ournani@inria.fr

Romain Rouvoy
Univ. Lille / Inria / IUF
romain.rouvoy@univ-lille.fr

Pierre Rust
Orange Labs
pierre.rust@orange.com

Joel Penhoat
Orange Labs
joel.penhoat@orange.com

ABSTRACT

Background. As software took control over hardware in many domains, the question of the energy footprint induced by the software is becoming critical for our society, as the resources powering the underlying infrastructure are finite. Yet, beyond this growing interest, energy consumption remains a difficult concept to master for a developer.

Aims. The purpose of this study is to better understand the root causes that prevent the issue of software energy consumption to be more widely considered by developers and companies.

Method. To investigate this issue, this paper reports on a qualitative study we conducted in an industrial context. We applied an in-depth analysis of the interviews of 10 experienced developers and summarized a set of implications.

Results. We argue that our study delivers *i)* insightful feedback on how green software design is considered among the interviewed developers and *ii)* a set of findings to build helpful tools, motivate further research, and establish better development strategies to promote green software design.

Conclusion. This paper covers an industrial case study of developers' awareness of green software design and how to promote it within the company. While it might not be generalizable for any company, we believe our results deliver a common body of knowledge with implications to be considered for similar cases and further researches.

ACM Reference Format:
Zakaria Ournani, Romain Rouvoy, Pierre Rust, and Joel Penhoat. 2020. On Reducing the Energy Consumption of Software: From Hurdles to Requirements. In *ESEM '20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3382494.3410678

1 INTRODUCTION

The last decade witnessed several attempts to consider green software design as a core development concern to improve the energy efficiency of software systems at large [2, 3, 18, 23, 26]. However, despite previous studies that have contributed to establish guidelines and tools to analyze and reduce the energy consumption [1, 7, 12, 16, 17, 25, 32], these contributions fail to be adopted by practitioners till date [14, 28].

Concretely, both quantitative and qualitative studies [22, 28, 31] previously surveyed developers to establish assumptions about developers' knowledge of green software design. These studies highlight that developers might be aware of software energy consumption problems, but have a very limited knowledge on how to reduce the energy footprint of their software product. For example, Pinto *et al.* [31] mentioned collecting "vague" answers from developers when asked about how to deal with software energy consumption. Fang *et al.* [28] reported that, among 100 developers, a small portion are aware of the primary sources of software energy consumption. Only 10% of the participants try to measure the energy consumption of their software project, while less than 20% take energy into account in the first place. Moreover, the empirical study of Manotas *et al.* [22] reported that energy requirements are often more desires than specific targets. They highlight that developers believe they miss accurate intuitions about the energy usage of their code, and that energy concerns are largely ignored during maintenance.

However, to the best of our knowledge, none of these studies discuss *i)* the hurdles that prevent the broader adoption of green software design, and *ii)* the developers' requirements in terms of tooling in an industrial context. But, we actually believe that both aspects are critical issues to consider when aiming to reach an adoption of such tools and methods among developers.

Contribution. This paper reports on a qualitative investigation on software energy consumption considerations among experienced developers of a large company. Concretely, we conducted interviews with 10 senior/expert developers with the ambition to cover developers' opinions, problems, and requirements to promote the green software design in an industrial context. The key contributions of this paper can, therefore, be summarized as:

(1) Providing a detailed understanding of the interviewed developers' awareness and knowledge about green software design,
(2) Identifying the main constraints and challenges that developers encounter in their daily development,
(3) Building specifications for the tooling that suits developers expectations and experiences,
(4) Investigating the best ways to keep developers aware of software energy consumption and promote it within a company,
(5) Identifying the exact role and responsibilities of the company to promote green software design,

We believe it can offer a common body of knowledge for researchers, tools creators, companies, and developers, which can be considered to improve awareness and adoption of green software design. For example, our investigations highlight that adoption of green software design in an industrial context requires not only a tight integration of future tools into the software development lifecycle, but also the

central of companies to align green software design at the same level of priority as maintenance or performance concerns.

Outline. The remainder of this paper is organized as follows. Section 2 formalizes our research methodology and our experimental protocol. Section 3 analyses and discusses the observations and findings behind the interview answers. Section 4 reports on the implications and exploitations of our findings. Section 5 discusses the related works in the area of developer studies and software energy consumption, and highlights our contribution in regards to those studies. Finally, Sections 6 and 7 cover the validity threats and our conclusion, respectively.

2 METHODOLOGY

In order to achieve our objective, which is to conduct an in-depth qualitative study that encompasses developers tooling requirements and awareness, we adopted a qualitative research approach [9, 30], using straussian grounded theory concepts and components, such as: *coding*, *memoing* and *theorical saturation* [37]. Despite being complex and time consuming [4], this approach has been widely adopted by similar studies and has proved its effectiveness [10, 13].

Our methodology starts with an interview phase detailed in Section 2.1, followed by the data analysis phase explained in Section 2.3.

2.1 Interviews

Interviews are the first step and the main data source for our qualitative study. In this study, we wanted to cover 3 main research questions. The first research question is the awareness and knowledge of developers with regards to the software energy consumption. Even if this was the focus of previous papers, like [28, 31], it is still very important to investigate participant's opinions about software energy consumption, as it helps to better analyze his/her others answers, depending on what he/she thinks of the problem and how important it is. The second research question aims to investigate about the hurdles and constraints that prevent a better consideration of software energy consumption, but also to push the developer to define and describe the tools that will suit his/her experience, to promote the consideration of software energy consumption in his/her daily development. The purpose of the third research question is to identify the best ways and means to keep developers aware of software energy consumption, but also to zoom into the responsibility of developers on the one hand, and the decision-makers—or the company—on the other hand.

During the interviews, we wanted to leave our participants with the freedom to express and explain their ideas and opinions so we can gather more feedback. Thus, we went for semi-structured interviews. The following sections provide more details about participant's selection, interviews conducting protocol, and the questions we asked.

2.1.1 Participants. The main criterion for the participants' selection is their experience in software development. Experienced developers in each technology had more time to cover the details, strengths, and limitations of the technology they are using. A junior developer in a specific technology or programming language may not have enough time or experience to cover all the basics, best practices and go deep into the technical characteristics, and is less likely to include energy considerations in his/her coding routines. By experience, we do not mean the professional experience, but a decent amount of time the developer has spent on a technology/programming language, to have enough knowledge to understand and be able of criticizing this technology. Thus, our participants have experience of at least 15 years and have worked on both small/short and long projects. Moreover, our selected participants are volunteers who have expressed an big interest in our interview invitation to cover developer's understanding and requirements regarding software energy consumption considerations and are more involved in green software design activities in a major European telecommunication company of more than 100000 employees. The rationale behind choosing participants from the same company—such as in [11]—is to assess the role of the company in the practice of developers. However, our study focuses on how to promote green software design within a company and expose a detailed case study but does not ambition to create a model that could be automatically generalized to companies of different sizes, activity sectors, or policies.

Instead of rigidly fixing the number of participants, we kept on conducting our interviews until reaching a level of saturation on the collected data [35]. After 10 interviews, we noticed a convergence of the collected data and thoughts [40], even considering the difference of technologies mastered by our participants and the types of projects they usually work on. Moreover, 10 is a decent number of participants that is close to the studied population by other similar works [6, 13, 36, 38].

For privacy and confidentiality purposes, we omit the usage of our participant's names and every other sensitive information, such as teams or project names, and we rather use code names ranging from P1 to P10.

2.1.2 Protocol. The interviews were conducted in 3 steps. The first step is a narrative part where we describe the purpose of our study, what the interview is about, and how would it happen. It also includes the confidentiality agreement with the participant and some indications of the interview process.

The second step is the semi-structured interview, starting with questions about the participant profile, which cover: the participant studies, the type and examples of projects he/she worked on. Then, we continue with the interview questions that focus on the 3 research questions introduced earlier and listed in Section 2.1.3. Finally, we conclude the interview with a post-questions step, where we answer the participant questions and share some information and references if she/he is more interested in software energy consumption.

Our protocol was checked and assessed by a qualitative studies expert from the company, before being tested on two developers—whom results are not reported— to apply some adjustments on the questions and the interview scenario on the one side, and have a better duration estimation to inform every candidate of the average time before every interview on the other side. To make the interview very fluid and capture every information, we recorded (with the agreement of the participant) the second step of the interview to apply post-in-depth analyses. We also prepared a quick sheet summary that allowed us to note the key answers for each question, along with participants' key thoughts and opinions. This mainly helped us to quickly detect the data saturation, as suggested by the qualitative studies expert, before the detailed analysis phase that confirmed it.

Three of the interviews were held face-to-face. The others were conducted via a call due to the distance between the interviewer

and the interviewee sites. Also, all the interviews were done in the native tongue of the participants to avoid any misunderstanding or expression difficulties due to the language. The mean duration time of the interviews is 39 minutes and 36 seconds, with a minimum duration of 28 minutes and 13 seconds, and a maximum duration of 54 minutes and 09 seconds.

2.1.3 Questions. Using semi-structured interviews was very helpful in our case. It allows identifying the main questions defining the purposes of our investigation. It was supported by follow-up questions to adapt to the participant's answers and let explore more details and directions in their answers. The main questions have been pre-defined and structured before the interview so the process goes faster, and to keep track of our baseline questions and concerns. We gave special attention to the formulation while preparing the questions. We wanted them to be open so we do not get a "yes" or "no" answer, but also to go deep in every participant's answer with the follow-up questions, as long as we maintain the theme of the main question. The main questions we asked are the following:

(1) What do you know about software energy consumption and green software design?
(2) What importance do you give to software energy consumption?
(3) What are the software energy consumption considerations that you take in your developments?
(4) What do you think are the constraints and hurdles to a better software energy consumption consideration?
(5) How ready are you to change your used-to programming language, technology, library, for better software energy consumption?
(6) How do you describe perfect tooling that suits your coding requirements for green software design—you can go deep into technical details?
(7) How do you think we should inform about energy software consumption for a better awareness?
(8) Do you think that getting a better software energy consumption consideration is the responsibility of the developer or the company? How?
(9) How can green software design be used as a marketing argument?

The questions (1) to (3) help to cover the participant knowledge and awareness of software energy consumption. The question (3), in particular, investigates any experience with methods, tips or tools, that the participant has used for green software design purposes. While the questions (4) to (6) aim at discovering the constraints that developers encounter and their tooling requirements, for better software energy consumption considerations. The purpose of the last three questions is to learn how to improve awareness. For that, we look for the best communication channels that developers would react to, to promote their consideration of the green software design. The last question is a more open one that summarizes the participant's belief and gives him/her more freedom to evoke some points that we might have missed during the interview.

Depending on the participant's answers, we ask some follow-up questions which are guided by the theme of the main question and the content of the answer. One example of a typical follow-up question we had to ask quite often along with the question (5) is: Have you questioned the quality of a tool/method/technology you have been using for a long time during your experience? How was that?

2.2 Transcription

After each interview comes to the transcription phase and we opted for a denaturalism approach to transcribe our records. This method has been used in similar works, such as [13], and allows putting a focus on the interview content while being lighter, but as complete and trustful as other methods, like Verbatim [27].

The transcription was made in the same language of the interview, but we translated some parts in English to quote participant's opinions in Section 3.

Some of the participants agreed only on sharing the results of the study, but not the raw data (recordings and transcripts). Nevertheless, we worked on preserving the participants' privacy, by omitting project names for example.

2.3 Analysis

We based our data analysis on the Straussian grounded theory coding procedure [37, 41]. First, we started with the *open coding* phase, where we read our transcripts several times and tried to summarize every chunk of data into a label, based on the meaning interpretation of the text. These label are called "open codes". Next, we used *axial coding* to identify the connections among the previously extracted open codes. Then, we used *selective coding* to figure out the core ideas, which cover all the data we collected. Finally, we read the transcripts again and selected any data that relates to the core ideas so the content segments of the transcripts will be all assigned to a core idea.

The analysis has been independently conducted by two authors to increase the accuracy and hinder the subjective interpretation overhead. The results were then compared and discussed for a consensual decision.

3 OBSERVATIONS & FINDINGS

Table 1 summarizes the key results of our study, with the core ideas that also match our main objectives. We use "SEC" hereinafter as a reference to *Software Energy Consumption*. The check-mark (✓) in each cell indicates a positive response from the participant regarding every idea that the core idea encompasses. This section discusses our observations, each section covering a reported idea. Every single idea of Table 1 is then discussed in a dedicated paragraph. Ideas that express close meanings and purposes are grouped within the same category. We provide a discussion at the end of every category to summarize the observations and findings of the detailed ideas and to add our thoughts and recommendations.

3.1 Developers Awareness & Knowledge About Software Energy Consumption

3.1.1 Developers Knowledge About SEC. The level of knowledge of the participants on software energy consumption is disparate. Some of the interviewees reported having some knowledge about green software design, or even considered it in their projects, while others reported a complete ignorance on the topic.

I already know about SEC. Software energy consumption is a relatively recent subject that people may or may not know about. For

Table 1: Summary of our interview analysis.

Core ideas	Ideas	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Developers awareness & knowledge about SEC	I already know about SEC	✓	✓				✓	✓	✓		✓
	I already considered SEC in my projects		✓				✓	✓	✓		
	SEC is an important subject to consider		✓	✓		✓	✓	✓		✓	✓
	SEC should be a high priority			✓		✓	✓	✓			
	SEC might cause conflicts with other coding metrics/aspects		✓						✓		
Constraints & tooling problems	No time to think about SEC	✓							✓		
	No tools			✓				✓	✓	✓	
	The main problem is not at the developer level		✓	✓					✓	✓	✓
	Ignorance		✓				✓	✓		✓	✓
	Enhancing performance often enhances the SEC		✓		✓	✓					
	Need for a SEC score/KPI	✓		✓	✓	✓	✓	✓	✓	✓	✓
	Include SEC among tests/CI	✓		✓	✓	✓		✓	✓		✓
	Static code analysis	✓				✓			✓		
	Simple tool with simple outputs	✓		✓	✓		✓	✓		✓	✓
	In Favor of Moving to Other Technologies / Tools			✓		✓	✓	✓			✓
Promoting SEC	The company has most of the responsibility compared to devs	✓	✓	✓	✓	✓		✓	✓	✓	✓
	The company should put objectives around SEC	✓			✓	✓		✓	✓	✓	
	The communication about SEC should be improved	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Need for training				✓						✓
	Simple presentations are effective	✓		✓	✓	✓	✓	✓		✓	
	Let's put green labels on software	✓		✓			✓	✓		✓	

some of our participants, they heard about on many occasions lately. *"I have attended several talks on software energy consumption problems, and have seen several things"* reported P1. Some of the participants reported never hearing about software energy consumption (P3, P4, P5) with answers like *"nothing"* or *"absolutely no idea"*. Others have shown a very recent interest in the subject using expressions, such as *"I became interested in software energy consumption a couple of months ago"* (P6, P7, P10).

Moreover, even among developers who answered the question affirmatively, some had a hard time explaining what do they exactly know about software energy consumption and gave global optimization examples, which are not specifically related to energy consumption. *"I know that reducing code size is good for energy consumption"* said P6. While P7 reported *"I have some ideas about web applications, such as reducing the size of data we send to the user"*.

I already considered SEC in my projects. Among the participants who reported knowing about SEC problems, 4 claimed they have already applied software energy consumption related practices in their projects. P2, for instance, witnessed *"I try, but it's not easy [...] we avoid to do useless animations [...] limit the access to servers [...] we keep an eye on the battery so our software doesn't drain it too fast"*. P6, P7, and P8 reported on attempts to reduce the software energy consumption through enhancing the performance, *"we try to cache data and limit the transfers [...] the main focus is the performance but also the energy by chain effect"* said P6. For P7, he/she is trying to get more involved in considering green software design in his/her projects. *"I think making sure the mobile application works on old phones is a good example"* he/she shared, confirmed by P2.

Discussion. Developers confront numerous kinds of information from multiple sources. Such sources do not always constitute a valid/correct set of knowledge. In the case of our study, some of the participants heard about software energy consumption, but could not provide a correct formulation of their knowledge without diverging from the energy consumption.

We argue that developers awareness can be classified into 4 different levels: (1) not knowing about software energy consumption, (2) having wrong/incomplete knowledge about the issue, (3) stacking theoretical knowledge with no application, and (4) knowing and applying SEC considerations. We evaluate most of our participants to be at the 2^{nd} stage, while proper communication and training programs should be established to help developers reaching the 4^{th} stage.

3.1.2 SEC Importance Among Developers. Our study shows that not all the participants give the same importance to green software design. The participants reported different levels of awareness, from being not important to be one of the highest priorities in software development.

Importance of SEC. While most participants think that software energy consumption is an important issue, some think that little attention is generally given to it: *"Pretty low importance"*, *"none, absolutely none"* and *"what's important is to deliver the service to the consumer"* (P1, P4 and P8, respectively). These answers are more related to their professional environment and work, rather than their own opinion and personal considerations. P8 added *"if we are talking at the social level, then the energy consumption is important, but it is not the case for what we are producing at work"*. The other participants feel like the matter is quite important and should be more considered at work. P5 and P10 even shared that the professional environment can support the movement even better *"as we trend to sobriety"* said P5, pointing at the company's objectives towards sobriety and greenness. *"It is part of the current challenges"* reported P10 in the same context, referring to the newly announced environmental objectives of the company.

Priority of SEC. Being important is a thing, but being a priority is different. Among the participants who reported the importance of software energy consumption, P2, P9, and P10 think that it is important but, at the time of the interview, not a priority in the company. *"It's one of the main challenges but it will be utopian to think it's a priority"* argued P10. *"From a company point of view, it is not the priority"* added P2. This shows a different understanding of company strategy and priorities. P4 symbolizes it, by answering *"zero, but I might change my mind if I get persuaded that it's not the case"* when asked about his perception of the current priority level of green software design within the company. On the other hand P2, P6 and P7 see it as a priority that the company has pushed in the last decade. *"The green aspect is ubiquitous"* said P6.

Discussion. While discussing the value and the significance of software energy consumption, we got more evidence about the problem of communication. We can see that not all developers were on the same level of awareness, knowledge, and even trust towards the company [28]. While all of our participants are in favor of green considerations in their personal lives as they claimed, some developers do not think the matter of software energy consumption is important in the current development processes at the company. Some participants think it should be a high priority in daily coding tasks, while others give it the same importance as several other code-related aspects (security, maintenance, etc.).

This also highlights the differences in the trust of the company. While some participants are in favor of the strategy of the company, others think that it should be improved. For this matter, we think that the company should conduct a large transparent internal campaign, to precisely identify the objectives and requirements towards the employees, and clarify any misleading ideas, thus ensure the same comprehension inside the company.

3.1.3 SEC Might Conflict with Other Considerations. P2 and P8 reported that software energy consumption considerations might cause potential conflicts with other software metrics and cause a lower rate of maintainability, security, scalability, etc. On this matter, P2 reported *"today, if I have to choose one thing over the other, I would choose code maintainability over a lower energy consumption"*. P8 has also a similar opinion *"I would only consider software energy consumption in a third position, once we ensured that the service is well optimized, and the security is guaranteed, which is one of our biggest concerns"*. Some other participants as P6 did not evoke conflict and talked about including the new metric among the other existing metrics.

Discussion. While most of our interviewees are confident about software energy consumption integration within the daily coding considerations, some legit questions may arise on how this integration would happen. We are not safe from conflicts that might occur. Choosing a less consuming, but harder-to-maintain programming language, or substituting some consuming security methods, are examples of those potential conflicts. Hence, a need for a well-designed strategy is required within projects. This strategy could include a set of objectives and validation steps per project, that derives for a more global set of objectives, to ensure taking the same choices and achieve the same purposes, as defined in the global strategy.

3.2 Constraints & Tooling Problems

3.2.1 Work Constraints. Some challenges have been raised by our participants to express the difficulties against software energy consumption considerations.

No time to think about SEC. *"No time"*, is the answer that provided two of our participants. P1 reported *"in our projects, we don't have any free time"*, meaning that with all the considerations that a developer has to take into account in a project, he/she cannot afford any extra time to deal with software energy consumption issues, at least not if no time was specifically allocated for that purpose. P8 evoked the same issue, highlighting the potential conflict between the extra developing time required when using less energy-intensive technologies, and the allocated time: *"the time factor is never negligible in our projects, and the least consuming technologies tend to require more time"*.

No tools. P3, P7, P8 and *P9* miss the appropriate tools to dive further into software energy consumption problems. *"I never heard of such tools or what they do"* is what they reported. This lack of tooling, and the related lack of feedback on actual energy consumption, hinder the analysis of the root causes of SEC, and thus the potential actions they could do to recover from bad energy practices. For example, P7 reported: *"we don't have any feedback or indicators"*. *"The main problem for me is the lack of tools, we don't have automatic tools for green code quality"* shared also P9 to express some frustration due to the absence of tools for green software design.

The main problem is not at developer level. For some of our participants, the main problem is not at the developer level. In fact, for P8, P9 and P10 the problem is at decision-making level. The developers being only the execution unit, they do not have much impact once the decisions have been made: *"we don't have full decision power"* said P2. P8 reported *"I am not sure if this is the crux of the problem, but we should be able to provide the developers with the proper tools first so they can achieve this purpose"*. This points to a lack of tools, but more importantly, to the role of the whole chain to organize and define the work conditions, towards a green software design.

Ignorance. By "ignorance", the participants refer to the lack of knowledge, but also the lack of awareness about software energy consumption. P2, for example, reported a problem with designing mobile application interfaces. *"I have some requests to put animations all over the screen, which does consume a lot of energy and doesn't improve the user experience by much"* he/she said. This ignorance is also illustrated among some developers, *"ignorance is the first reason, I didn't know a thing myself, developers don't know what can they achieve and the impact they could have"* claimed P7. P9 and P6 share the same opinion, *"people think resources are endless"* reported P6. For P10, this ignorance problem might even dissuade good wills that want to change *"the problem is that we don't all share the same green culture at the company, if the team we work with is not on the same page, nothing will happen. We need to be all on the same level"*.

Discussion. The participants we interviewed expressed a list of constraints and hurdles that prevent software energy consumption considerations. Among these constraints, we can identify the lack of time, tools, and awareness. Indeed, the developers would feel much more comfortable about green software design if they had dedicated

tools that support that activity, and a decent awareness so they can set software energy consumption related objectives at the beginning of every project for example. A dedicated time is also required. By the time, we mean allocating a specific period so the developers would be able to set up the green software design environment and checks, but also allocating the time that developers need to get along with the potentially new aspect of energy in software development.

The inclusion of the decision-makers when talking about software energy design has also been a matter of discussion in some interviews. Some project's key decisions may have a big impact on the energy efficiency of the final product. Choosing the proper technologies and allocating the proper time can be good examples of that, where developers do not have much choice and just try to deal with the constraints and deadlines.

3.2.2 Enhancing Performance Often Enhances SEC. While most of the participants diverged into performance metrics when asked about energy, some of them reported a correlation between the performance of software and its energy consumption. *"There is certainly a direct link between performance and energy consumption, for mobile application. The more we ask the phone to do the quicker drains the battery"* claimed P2, and *"theoretically performance and energy consumption are not the same things, but in practice, they are"* reported P4. For other developers, like P3 and P9, the causality relationship is not that evident: *"sometimes we spend a lot of extra energy while trying to enhance the performance"*. They even gave some examples like allocating more servers to go faster while requiring more energy.

Discussion. Our participants have mainly experienced dealing with performance instead of energy consumption. Thus, they referred to performance several times instead and tried to replicates their knowledge on energy consumption. We think that "performance vs energy" is a mandatory topic that should be discussed when promoting software energy consumption, as all the developers should be able to distinguish the slight difference and knowingly take action that can favor performance over energy and *vice-versa* when there is a room for conflict.

3.2.3 Tooling Specifications. Gathering requirements for tools that would match developers' requirements is one of the priorities of this study. Fortunately, we identified some specifications that should help to design the next set of SEC optimization tools.

Need for a global score / KPI. This has been the most requested and discussed specification. Almost all the participants mentioned the need for a global score or *Key Performance Indicators* (KPI) for the total software energy consumption evolution. *"We need to have indicators with a ranking system"*, *"consumption summary"*, *"We don't have any KPI"*, *"We need KPIs" and "track the evolution using simple KPI"* are claims from P1, P3, P6 and P7, respectively. P4 and P9 went a bit further and assimilated it to the consumption ranking that is used for household appliances, like washing machines: *"energy consumption classes like A, B, C, etc."*

Include SEC among tests/CI. This also was a common proposition for many participants when asked about how they describe a tool that would measure and track the evolution of software energy consumption. They suggested for software energy consumption reports to be integrated within the existing system platforms that the developers are using, *"It would be a tool that is integrated with my CI chain to track the consumption evolution of my software"* reported P7. Moreover, some developers do not think software energy consumption can be measured/tracked at the code level and is dependent of the running environment that could be modeled and simulated through testing and continuous integration, *"we could imagine the usage of micro-benchmarks to test the code quality on the same execution environment, which is not possible on the development station"* stated P5.

Static code analysis. Some developers assimilated a part of green software design tooling to static code analysis tools, such as the Sonarqube tool,[1] *"we could assimilate it to a Sonarqube to apply a first static audit and check some well-known bad practices"* mentioned P8. Others think that static code analysis is quite irrelevant for this purpose. P7, for example, does not believe in static code analysis as SEC is very dependent on the execution environment. *"We cannot have generic practices, we should specify the target, the platforms, etc."* said *P2. "It's is also dependent of run-time"* reported P3. This shows that not all developers have the same trust towards static code analysis to diagnose software energy consumption issues.

Simple tools with simple outputs. Participants also asked for simple tools with simple outputs, with the use of graphical interfaces to track the software global energy consumption's evolution, *"[...] with graphical output [...] that lets me notice the* 10% *energy difference"* reported P10. *"I should not need a Ph.D. to understand the outputs of the tool"* said P7 to illustrate his/her frustration with complex tools overloaded with too many details and no single score that defines the global status.

Discussion. The participant's feedback about the tools was very rich and converged to the same main ideas, where usability seems to be the key concern. Developers expressed their requirements in terms of tooling focusing on the simplicity of the outputs, which should include global KPIs/scores. When talking about energy consumption, the participants are very used to this kind of score in their daily life, with energy labels for household appliances, bulbs, houses isolation, etc. Moreover, the same kind of scores is also routinely used in their daily development work with scores on CPU consumption, memory and disk usage, response time, etc. This global indicator should allow them to track the energy consumption evolution of their source code and would be an entry point to dive into the details about the consumption of a more specific code part, like a method or an algorithm. The static code analysis was a bigger question mark to some developers when speaking of its effectiveness. While it could be very beneficial to establish some rules about bad practices and common energy bugs [29] through linters for example, it still delivers limited feedback on the actual energy consumption at run-time. We argue that the discussed aspects should be taken into account by tools creators, where we could imagine a tool that applies static code analysis rules during the development phase (integrated with the IDE for example), an energy profiler for code tuning, followed by a run-time energy tracking, integrated with CI/CD, and a dashboard for data visualization. The displayed information could be scores calculated from run-to-run evaluation, energy details/guidelines about the used technologies, etc.

[1] https://www.sonarqube.org/

3.2.4 In Favor of Moving to Other Technologies / Tools. Considering switching to another programming language, for example, is a legit question to ask when talking about a new aspect as software energy consumption. Some of the participants (P3, P5, P6, P7 and P10) reported no resistance for change. This openness is justified by the recurrent changes of technologies in their previous projects. *"I very often move from one technology to another, I have no problem with that"*, *"it wouldn't even be a difficulty"* and *"yes, it's a good thing"* reported P7, P6 and P10 to express their confidence in being able to move to other technologies for green software design purposes. Meanwhile, some developers do more worry about this change: P9 explained that the choice of technologies is also related to the developer profile, and going for more energy-efficient but less used/famous technologies would be a bad decision for his/her career. For P1, software energy consumption is not important enough to justify such a delicate thing as changing the used technologies, *"it is hard to say, especially to re-develop the already existing software, choosing the programming language, for example, is very delicate, especially when software energy consumption is not a priority"*.

Discussion. It is not surprising that some developers will express reluctance to change, as it is a delicate process that is influenced by both technical and social factors [19]. Especially if developers do not have enough knowledge of what they can do, how they can do it, and the reasons for such a change. It was however interesting to observe that some developers are in favor of such changes. This gives hope towards applying green software design. We argue that these changes should not be done in one shot, but in a couple of steps to make the adaptation easier for developers, especially after a good awareness and teaching campaign within the company.

3.3 Promoting SEC

3.3.1 The Role Of The Company. As for the developers, the role of the company is important to establish green software design practices.

The company bear most of the responsibility compared to developers. Except for P6 who does not believe in *"forcing"* the developers, but in *"if each person is aware, the collective will thrive"* instead. All other participants assigned the major responsibility to the company, to guide the application of green software design. According to P4, software energy consumption will never be a long term consideration if the company does not take the lead, *"We cannot have a hundred ways of doing, it has to be guided"* he/she added. *"You can't stay in a corner and hope it will work, it has to be applied and guided through the whole chain"* argued P2. P3 compared the roles of the company and developers to a garbage selection process, where he/she assimilated developers role to *"putting the trash in the specific bins"* (small but important responsibility) and the company role to the recycling that is done afterward (big responsibility).

The company should include SEC objectives. One of the company's roles that were fairly repeated during the interviews is setting objectives about software energy consumption and green software design. Many participants reported that setting objectives is a good way to promote SEC considerations. P5 argued *"we should have objectives around that, this would allow developers to see what they are doing and what they can achieve"*. Moreover, having objectives will give more credibility to the task, as it has been specified by products owners, *"having objectives will give more sense to green software design, and it will be one of the aspects that developers are going to be judged on by the end of the semester, as it has been specified by product owners"* reported P7. Those objectives could be related to the KPIs, *"identify some KPIs, and set some objectives around those KPIs"* reported P9.

Discussion. Establishing green software design in the company is a matter that has to be supported by both decision-makers and developers. The company has certainly a backbone role in this process, starting with keeping the developers aware of green software design, providing them with the needed tools, and identifying global and projects-related objectives. Setting objectives is very important for various reasons. First, it shows the dedication of the company towards green software design, which will transfer to the employees afterward. Then, defining a milestone would help developers to be more motivated to unlock a new achievement every semester/year regarding software energy consumption.

3.3.2 Communication Means. Now that we have seen the relative lack of awareness and knowledge many times across the previous discussions, we asked the developers what should be done to reach them.

The communication should be improved. As pointed many times in multiple discussions in this paper, the communication around software energy consumption and green software design should be improved. All our participants brought the communication problem at some point of the interview, by evoking the ignorance of the employees on these subjects (P2, P6, P7, P9, P10), or by describing the company as the guide to raise awareness about green software design (P1, P4, P5, P7, P8, P9).

The participants also gave some examples of the kind of communication they think they will be receptive to, like training programs, presentations, and conferences. P5, for example, proposed the usage of the company social network that is *"used by most of the employees"*.

Need for training. Two of our participants reported a crucial need for training to learn the specificity of software energy consumption and how to manage it. P4 argued that training is very important if software energy consumption is a real concern today, *"a training over a couple of days to learn how to do things at developer level would be very welcome. We do training for other code aspects such as security, why not for software energy consumption"*. For P10, one solution is to train a group of developers to be experts in software energy consumption. These experts would then integrate project teams and would have for mission to help the other developers learning and applying the green software design. *"Training all the developers would cost a lot of time and money"* added P10.

Presentations are very effective. Many participants argued that presentations (informal presentations, presentations in conferences, etc) are effective to keep developers aware and inform them about software energy consumption and green software design. According to P3, *"I attended a presentation lately about green challenges and I found it very interesting, with real examples"*. The presentations should be instructive but simple *"including comparisons, pictures"* said P6. *"Ideally provide some tips with the related impacts, I have a book that enumerates* 115 *web best practices, that's huge, it should have* 15*, even* 15 *is still quite a lot"* added P7.

Discussion. Communication is the keystone of every activity inside the company, and promoting green software design is no exception to the rule. The company should amplify and refine the communication around green software design, so it can be much deeper than just announcing a global plan. It has to allow the developer to act on the project he/she is working on, and to see his/her impact whenever possible. Our participants mainly evoked two means of communication. First, the presentations should be very brief, very recurrent, and should include more examples and tips rather than flat knowledge. The presentations should be used mainly for awareness rather than knowledge transfer. Training is the second evoked way of communication, which is more knowledge-based, with more details than just tips, due to the specificity of training (much more time and fewer participants compared to presentations). We argue that a wise combination of these two means of communication would be a good start towards promoting green software design, as it allows having a recurrent communication that involves most of the employees, to keep them informed about the objectives and the main guidelines. This can be achieved through presentations, meetings, and other formal or informal communication means. The second part of communication should provide the employees with the necessary knowledge to achieve that task efficiently, through training and workshops, etc.

3.3.3 Green Software Marketing. The purpose of the last question of the interview is to summarize every participant's belief in software energy consumption. The answers were split between two main ideas. While some interviewees (P1, P3, P6, P7, P9) did not hesitate to say that putting green labels on the produced software should create another selling argument. Other participants were very cautious about that label. P4, for example, reported *"I am worried about greenwashing, we have to be very careful about that"*. In the same context, P2 reported *"I am kind of worried that it will be used a little bit too easily, we have to be green and not pretend it"*.

Discussion. The two keywords that summarize what developers—who represent the production unit—think of green software marketing, are integrity and transparency. The participants argue that selling green software should even be more attractive and could constitute a good marketing argument that differentiates the product from other providers, even if there is no direct pressure from the consumer to build more energy efficient software [5].

However, this marketing has to be very transparent towards both developers and end-users and avoid misleading communication and greenwashing.

4 IMPLICATIONS

We summarize the results of our study as sets of implications for developers, the company, tool creators, and researchers. We argue that these implications constitute a rich knowledge base that could guide understanding and promoting software energy consumption and green software design.

4.1 For Developers

Developers have been the data source of our study. Hence, we can retrieve a couple of implications for them:

- Given the observed level of awareness in our interviews, we suggest that developers should seek more information on environmental issues in general and the impact of the IT sector in particular. This would help the developers to grasp the importance of these issues and motivate them to work on them;
- Some developers seem to consider the changes implied by green software design as threats to their careers. We advocate instead that it could be an opportunity for professionals in the IT sector: skills in this emerging domain will be in short supply in the future while demands will probably increase substantially, especially with the growing concern about green technologies and energy consumption. Therefore, we encourage developers to invest time now in these key skills and argue that it will pay off quickly enough;
- Developers seem to be more receptive to messages coming from their peers. Therefore, developers with better knowledge about green software design should volunteer to help to inform and teach, both in their project team and to the whole IT community. Organizing presentations, submitting talks to developers conferences and frequently posting on the public and company social networks are effective ways of doing it;

4.2 For Decision Makers

We can extract many implications and responsibilities for decision-makers—a.k.a companies—from our results, including:

- The main role of the company is to maintain a large communication campaign about SEC that encompasses: *i)* ensuring a high level of transparency with the employees regarding the "green" vision and objectives, *ii)* running a long-term awareness program (with presentations for example), and *iii)* providing the developers with the necessary knowledge (through training programs, workshops, etc.);
- Developers described the identification of green objectives for development projects as one of the most efficient ways to motivate employees about green software design and clarify the company's position and engagements. These objectives would create additional motivation for developers, and would define entries that developers and product owners would discuss, validate and readjust, every period;
- Developers also requested for necessary resources so they can achieve green software design objectives. The resources include the tools (and/or budget) that allow monitoring the software energy consumption and the necessary time to do it. Yet, there is already exploitable resources that the company could make use of, such as *i)* developers who already know green software design who could help in the communication/teaching process and *ii)* the ability of developers to adapt to different technologies, to attribute the human resource in the most convenient project;
- Marketing the green aspect of the software is also a major sector that the company should prioritize, as it would help to get clients' feedback regarding the market and the products. This would help to readjust the objectives and product specifications if necessary.

4.3 For Tool Makers

Our study provides numerous guides and specifications for tool creators:

- Developers ask for tools that can output a global score / KPI to summarize the energetic footprint of the source code. This score can then be used for communication with other stakeholders in the projects, to check that energy efficiency targets have been met, but also to easily follow the evolution of the energy consumption of software through commits, for example using graphical representations;
- At the same time, more advanced tools are also needed, that provide more detailed information to allow low-level diagnosis of the source code and identify the exact problems and solutions. This family of tools must however still be simple to use and provide graphical representations to simplify the interaction with the displayed information. Therefore, we argue that toolmakers must pay close attention to the usability of their tools, which is paramount to ensure that developers will be confident about using them;
- While there is at this time no clear consensus on the effectiveness of static code analysis for energy efficiency purposes, developers could be persuaded to use energy-focused linters, designed to flag bad practices and common "energy bugs" [29], as long as their benefits are demonstrated. To better convince developers of using such linters, toolmakers should concentrate on demonstrating the effectiveness of this approach, and integrate them in commonly used editors and IDE;
- The developers expect these tools to integrate seamlessly with available tool-chains, especially for *Continuous Integration* and *Continuous Delivery* (CI/CD), to automate the analysis and report on the energy footprint along with other metrics like code quality, performance, and tests reports.

4.4 For Researchers

Our study confirms previous works results—such as Pang *et al.* [28]—that highlighted the lack of knowledge and awareness among developers. Moreover, we present a new set of information that could be considered and extended in further research about green software design:

- The correlation between the technologies developers use and their level of awareness on green software design is an interesting research topic, like Manotas *et al.* [22] who stated that mobile developers are more concerned about software energy consumption problems. In our study, there seems to be no consensus among developers, even those coming from the same technical background Some empirical studies with a larger population should be conducted to investigate the correlation between developers considerations and their background;
- We discuss a specific case study to understand the state of mind of a set of experimented developers in a large company of more than 100,000 employees in the telecommunication sector. However, it does not make our study automatically generalizable for other companies operating in a different sector, or companies of a different size, or even companies located in a different geographical area. Further empirical studies can be conducted on other samples, such as startups, collaborative projects, open-source projects, etc. across different regions and with numerous domains of activity. Quantitative studies are more adapted to conduct this kind of studies with much bigger samples to track the different axes that may change developers' opinions and constraints regarding green software design;
- Our study highlights the need for KPIs/scores about software energy consumption. Hence, considerable research work is still needed to build the theoretical knowledge on the right set of KPIs and visualization formats that tool creators could implement, and that would speak better to both developers and decision-makers;
- We noticed during our study that a couple of trade-offs should be considered along with green software design. Not all developers fully distinguish software energy consumption from its performance, thus multiple works that could help developers to make choices and to deal with the slight difference when it occurs can still be done. Moreover, the participants foresee a trade-off between the development time and using less consuming programming languages which should be proven or denied in further researches.

5 RELATED WORK

In this section, we review the state of the art and present some studies that have been conducted on developers consideration of green software design, and highlight the novelty that our study brings.

First, Fang *et al.* [28] surveyed 100 persons focused on highlighting programmers' knowledge of software energy consumption. Their results expressed a lack of knowledge on ways and best practices to reduce software energy consumption. The paper highlights that only 10% of the participants try to measure the energy consumption of their software project, and only 18% think about energy at all. Moreover, the authors mention an urgent need for training and education on software energy efficiency.

On the other hand, Pinto *et al.* presented an empirical study on how programmers understand software energy consumption problems [31], using more than 300 questions and 550 answers from 800 participants on StackOverflow. The authors stated that practitioners are aware of software energy consumption problems, but have limited and vague answers on how to deal with it. These two works show that developers' awareness concerning software energy consumption problems varies and that their knowledge is very limited, as shown by their collected answers.

In a later work, Manotas *et al.* [22] conducted a mixed quantitative and qualitative study, applied on 464 and 18 candidates, respectively. The study provided some interesting results, reporting for example that *i)* mobile developers are more concerned about software energy consumption problems, *ii)* energy concerns are largely ignored during maintenance, *iii)* energy requirements are more often desires rather than specific targets, *iv)* developers believe they do not have accurate intuitions about the energy usage of their code and are undecided about whether energy issues are more difficult to fix than performance issues, and *v)* 93% of the survey participants want to learn about energy issues from profiling and static code analysis, Interestingly, this last result is in contrast to Johnson *et al.* work [15], in which they interviewed 20 candidates through a qualitative study

and discussed why developers do not use static analysis to find bugs, such as false positives and the way warnings are displayed. One other similar work is [14], where the authors conducted a set of semi-structured interviews to discuss the added value of an applied energy profiling method across releases of software. Software energy consumption was also subject to many other empirical studies. Sahin *et al.* provided an empirical study on the impact of 6 commonly-used refactoring rules on software energy consumption. They found that applying some rules, such as *extract method*, can have a real impact on varying energy consumption [33]. In another work [34], the authors presented an empirical study on the overall negative effect of code obfuscations on the energy consumption of a mobile application. Pathak *et al.* investigated the kind of energy bugs or e-bugs on over 39k posts on mobile applications [29].

Progress beyond the state of the art. Our study of the state of the art revealed the lack of a full qualitative study that conducts an in-depth analysis of practitioners' requirements and comprehensions. Most of the studies we encountered focused on presenting a survey of what developers know about green software design. While this is very important, it does not deliver a better understanding of what are the developer's requirements and how to get his/her attention to the problem.

This paper rather conducts an in-depth qualitative study about software energy consumption considerations among a population of experienced and expert developers in a large company. This constitutes a solid case study that exposes key implications to be considered within the company, but also preliminary findings that could be checked across other companies' profiles. Our study investigates 3 major questions: 1) What do our participants know about software energy consumption? 2) what are the main hurdles that prevent green software design considerations? 3) What is convenient tooling (or tools set) that matches developers' expectations and experience with other software metrics, such as CPU consumption or execution time? 4) What are the most efficient ways to reach the developers and the deciders, and keep them aware of the importance of energy considerations in software development and their role to promote it? The answer to the first question confirms the results of the previous papers, such as Pinto *et al.* [28, 31], regarding the moderate awareness and lack of knowledge of developers on green software design. This also means that in at least 6 years, the situation has not really evolved and developers are still struggling to handle software energy consumption issues. Moreover, our participants reported on a very mitigated consideration of static code analysis as it does not consider the execution environment, in contrast to the results of Manotas *et al.* [22].

The novelty of this paper is mainly related to the second and third questions, in which we seek to understand developers' requirements to better digest green software design and include it in their development routines and considerations. 1) setting individual and global objectives about green software design, 2) encouraging presentations and training to raise awareness and remedy to lack of knowledge, 3) including SEC in projects planning with dedicated time, tools and budget, and 4) developing and using dedicated tools, which must facilitate the integration of SEC considerations in the developers' daily activity and whose specifications can be drafted from this work.

6 THREATS TO VALIDITY

A couple of issues may affect the validity and the generalizability of our work. First, our population might not be representative of all kinds of populations for several reasons. Starting with the sample size, which offered a certain level of data saturation [37, 40], but is still not quite large for better generalizability to other populations. Conducting, transcribing, and analyzing the interviews are tedious manual tasks that are very time and energy-consuming. Hence, considering a much bigger sample size would have massively increased the workload for a qualitative study and would have implied considering diverse company types, regions, participants' profiles, etc. This is in contrast to our study's purpose to deliver a concrete case study to understand and promote green software design within a company. Moreover, considering a population of only experts and experienced developers that are likely more interested in the topic than average is not the best representation of all scenarios, as junior developers could have caused a slower data saturation, for example. The purpose behind selecting experienced and expert developers with a certain level of awareness is however to conduct a study that delivers insightful and high-quality findings and implications. This would highlight the most relevant issues and build a decent strategy to promote green software design within the company, even if it does not make it trivially generalizable. Another possible threat is the specificity of the region. All of our participants are from the same country and the same company. While this is useful to build an understanding of developers' opinions within the same company, our study may have missed other opinions from other countries/regions.

Unfortunately, most of our participants were located on remote sites. Hence, we conducted those interviews through calls, which does not allow capturing as many details as live interviews, such as the interviewee's behavior and gestures. Moreover, our analysis and interpretation can be a threat to the validity of our findings. To alleviate this issue, we were two persons to code and analyze the data separately so we can offer more credible results.

7 CONCLUSION

This paper reports on a qualitative study in which we conducted in-depth interviews with experienced developers working in a large company. The observations and findings we present deliver *i)* a better understanding of what developers think about software energy consumption and green software design, *ii)* an identification of barriers preventing its adoption, along with *iii)* specifications of developers' requirements in terms of tooling and support from the company. Our results have multiple implications for developers, decision-makers within the company, tools creators, and researchers, to promote green software design along with the other software development priorities.

REFERENCES

[1] H. Anwar, D. Pfahl, and S. N. Srirama. 2019. Evaluating the Impact of Code Smell Refactoring on the Energy Consumption of Android Applications. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 82–86.

[2] L. Ardito, G. Procaccianti, M. Torchiano, and A. VetrÃš. 2015. Understanding Green Software Development: A Conceptual Framework. *IT Professional* 17, 1 (2015), 44–50.

[3] A. Banerjee and A. Roychoudhury. 2016. Automated Re-Factoring of Android Apps to Enhance Energy-Efficiency. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 139–150. https://doi.org/10.1109/MobileSoft.2016.038

[4] Titus Barik, Brittany Johnson, and Emerson Murphy-Hill. 2015. I Heart Hacker News: Expanding Qualitative Research Findings by Analyzing Social News Websites. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, Bergamo, Italy, 882–885. https://doi.org/10.1145/2786805.2803200

[5] A. B. Bener, A. Miranskyy, and S. Raspudic. 2014. Deploying and Provisioning Green Software. *IEEE Software* 31, 3 (2014), 76–78.

[6] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. 2009. Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Software Engineering. (2009), 11.

[7] Maxime Colmant, Romain Rouvoy, Mascha Kurpicz, Anita Sobe, Pascal Felber, and Lionel Seinturier. 2018. The next 700 CPU power models. *Journal of Systems and Software* 144 (2018).

[8] Nelly Condori Fernandez and Patricia Lago. 2018. The Influence of Green Strategies Design onto Quality Requirements Prioritization. In *Requirements Engineering (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics))*. Springer/Verlag, 189–205. https://doi.org/10.1007/978-3-319-77243-1_12

[9] John W. Creswell. 2003. *Research Design: Qualitative, Quantitative, and Mixed Method Approaches* (2nd ed ed.). Sage Publications, Thousand Oaks, Calif.

[10] Denae Ford, Titus Barik, Leslie Rand-Pickett, and Chris Parnin. 2017. The Tech-Talk Balance: What Technical Interviewers Expect from Technical Candidates. In *2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, Buenos Aires, Argentina, 43–48. https://doi.org/10.1109/CHASE.2017.8

[11] Thomas Fritz and Gail C. Murphy. 2011. Determining Relevancy: How Software Developers Determine Relevant Information in Feeds. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11*. ACM Press, Vancouver, BC, Canada, 1827. https://doi.org/10.1145/1978942.1979206

[12] Ashish Gupta, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, Thirumalesh Bhat, and Syed Emran. 2014. Mining Energy Traces to Aid in Software Development: An Empirical Case Study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM âĂŹ14)*. Association for Computing Machinery, New York, NY, USA, Article 40, 8 pages. https://doi.org/10.1145/2652524.2652578

[13] Sarra Habchi, Xavier Blanc, and Romain Rouvoy. 2018. On Adopting Linters to Deal with Performance Concerns in Android Apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018*. ACM Press, Montpellier, France, 6–16. https://doi.org/10.1145/3238147.3238197

[14] Erik Jagroep, Giuseppe Procaccianti, Jan Martijn van der Werf, Sjaak Brinkkemper, Leen Blom, and Rob van Vliet. 2017. Energy Efficiency on the Product Roadmap: An Empirical Study across Releases of a Software Product: Energy Efficiency on the Product Roadmap. *Journal of Software: Evolution and Process* 29, 2 (Feb. 2017), e1852. https://doi.org/10.1002/smr.1852

[15] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 672–681. https://doi.org/10.1109/ICSE.2013.6606613

[16] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 2, Article 9 (March 2018).

[17] Foutse Khomh and S. Amirhossein Abtahizadeh. 2018. Understanding the impact of cloud patterns on performance and energy consumption. *J. Syst. Softw.* 141 (2018), 151–170. https://doi.org/10.1016/j.jss.2018.03.063

[18] Mohit Kumar, Youhuizi Li, and Weisong Shi. 2017. Energy Consumption in Java: An Early Experience. In *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. IEEE, Orlando, FL, 1–8. https://doi.org/10.1109/IGCC.2017.8323579

[19] Thomas D. LaToza, Evelina Shabani, and Andre van der Hoek. 2013. A Study of Architectural Decision Practices. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, San Francisco, CA, USA, 77–80. https://doi.org/10.1109/CHASE.2013.6614735

[20] Ding Li and William G. J. Halfond. 2014. An Investigation into Energy-Saving Programming Practices for Android Smartphone App Development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software - GREENS 2014*. ACM Press, Hyderabad, India, 46–53. https://doi.org/10.1145/2593743.2593750

[21] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining Energy-Greedy API Usage Patterns in Android Apps: An Empirical Study. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. ACM Press, Hyderabad, India, 2–11. https://doi.org/10.1145/2597073.2597085

[22] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. 2016. An Empirical Study of Practitioners' Perspectives on Green Software Engineering. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. ACM Press, Austin, Texas, 237–248. https://doi.org/10.1145/2884781.2884810

[23] Irene Manotas, Lori Pollock, and James Clause. 2014. SEEDS: A Software Engineer's Energy-Optimization Decision Support Framework. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 503–514. https://doi.org/10.1145/2568225.2568297

[24] Irene Manotas, Cagri Sahin, James Clause, Lori Pollock, and Kristina Winbladh. 2013. Investigating the Impacts of Web Servers on Web Application Energy Usage. In *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*. IEEE, San Francisco, CA, USA, 16–23. https://doi.org/10.1109/GREENS.2013.6606417

[25] Rodrigo Morales, Rubén Saborido, Foutse Khomh, Francisco Chicano, and Giuliano Antoniol. 2018. EARMO: An Energy-Aware Refactoring Approach for Mobile Apps. *IEEE Trans. Software Eng.* 44, 12 (2018), 1176–1206. https://doi.org/10.1109/TSE.2017.2757486

[26] M. Morisio, P. Lago, N. Meyer, H. A. MÃijller, and G. Scanniello. 2015. 4th International Workshop on Green and Sustainable Software (GREENS 2015). In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 981–982.

[27] D. G. Oliver, J. M. Serovich, and T. L. Mason. 2005. Constraints and Opportunities with Interview Transcription: Towards Reflection in Qualitative Research. *Social Forces* 84, 2 (Dec. 2005), 1273–1289. https://doi.org/10.1353/sof.2006.0023

[28] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. 2016. What Do Programmers Know about Software Energy Consumption? *IEEE Software* 33, 3 (May 2016), 83–89. https://doi.org/10.1109/MS.2015.83

[29] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. 2011. Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile Devices. , Article Article 5 (2011), 6 pages. https://doi.org/10.1145/2070562.2070567

[30] Patton, Michael Quinn. 1990. *Qualitative Evaluation and Research Methods, 2nd Ed.* (2 ed.). Number 0-8039-3779-2. Sage Publications, Inc.

[31] Gustavo Pinto, Fernando Castor, and Yu David Liu. 2014. Mining Questions about Software Energy Consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. ACM Press, Hyderabad, India, 22–31. https://doi.org/10.1145/2597073.2597110

[32] G. Procaccianti, P. Lago, A. VetrÃš, D. M. FernÃąndez, and R. Wieringa. 2015. The Green Lab: Experimentation in Software Energy Efficiency. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 941–942.

[33] Cagri Sahin, Lori Pollock, and James Clause. 2014. How Do Code Refactorings Affect Energy Usage?. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*. ACM Press, Torino, Italy, 1–10. https://doi.org/10.1145/2652524.2652538

[34] Cagri Sahin, Philip Tornquist, Ryan Mckenna, Zachary Pearson, and James Clause. 2014. How Does Code Obfuscation Impact Energy Usage?. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, Victoria, BC, Canada, 131–140. https://doi.org/10.1109/ICSME.2014.35

[35] Todd Sedano, Paul Ralph, and Cecile Peraire. 2017. Software Development Waste. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, Buenos Aires, 130–140. https://doi.org/10.1109/ICSE.2017.20

[36] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2019. How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool. *IEEE Transactions on Software Engineering* 45, 9 (Sept. 2019), 877–897. https://doi.org/10.1109/TSE.2018.2810116

[37] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *Proceedings of the 38th International Conference on Software Engineering (ICSE âĂŹ16)*. Association for Computing Machinery, New York, NY, USA, 120âĂŞ131. https://doi.org/10.1145/2884781.2884833

[38] Kristin Fjola Tomasdottir, Mauricio Aniche, and Arie van Deursen. 2017. Why and How JavaScript Developers Use Linters. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Urbana, IL, 578–589. https://doi.org/10.1109/ASE.2017.8115668

[39] D. Torre, G. Procaccianti, D. Fucci, S. Lutovac, and G. Scanniello. 2017. On the Presence of Green and Sustainable Software Engineering in Higher Education Curricula. In *2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)*. 54–60.

[40] Konstantina Vasileiou, Julie Barnett, Susan Thorpe, and Terry Young. 2018. Characterising and Justifying Sample Size Sufficiency in Interview-Based Studies: Systematic Analysis of Qualitative Health Research over a 15-Year Period. *BMC Medical Research Methodology* 18, 1 (Dec. 2018). https://doi.org/10.1186/s12874-018-0594-7

[41] Helmut R. Wagner. 1968. The Discovery of Grounded Theory: Strategies for Qualitative Research. By Barney G. Glaser and Anselm L. Strauss. Chicago: Aldine Publishing Company, 1967. 271 pp. . *Social Forces* 46, 4 (06 1968), 555–555. https://doi.org/10.1093/sf/46.4.555 arXiv:https://academic.oup.com/sf/article-pdf/46/4/555/6891522/46-4-555.pdf