Energy Patterns for Web

An Exploratory Study

Rani, Pooja; Zellweger, Jonas; Kousadianos, Veronika; Cruz, Luis; Kehrer, Timo; Bacchelli, Alberto

Citation (APA)
Rani, P., Zellweger, J., Kousadianos, V., Cruz, L., Kehrer, T., & Bacchelli, A. (2024). Energy Patterns for Web: An Exploratory Study. In *Proceedings - 2024 ACM/IEEE 46th International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS 2024* (pp. 12-22). (Proceedings - International Conference on Software Engineering). IEEE. https://doi.org/10.1145/3639475.3640110

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Energy Patterns for Web: An Exploratory Study

Pooja Rani*, Jonas Zellweger*, Veronika Kousadianos†, Luis Cruz§, Timo Kehrer†, Alberto Bacchelli*

{pooja.rani,jonas.zellweger,alberto.bacchelli}@.uzh.ch,{veronika.wu,timo.kehrer}@unibe.ch,L.Cruz@tudelft.nl

* University of Zurich, Zurich, Switzerland
† University of Bern, Bern, Switzerland
§ Delft University of Technology, Delft, The Netherlands

ABSTRACT

As the energy footprint generated by software is increasing at an alarming rate, understanding how to develop energy-efficient applications has become a necessity. Previous work has introduced catalogs of coding practices, also known as energy patterns. These patterns are yet limited to Mobile or third-party libraries. In this study, we focus on the Web domain—a main source of energy consumption. First we investigated whether and how Mobile energy patterns can be ported to this domain and found that 20 patterns could be ported. Then, we interviewed six expert web developers from different companies to challenge the ported patterns. Most developers expressed concerns for antipatterns, specifically with functional antipatterns, and were able to formulate guidelines to locate these patterns in the source code. Finally, to quantify the effect of Web energy patterns on energy consumption, we set up an automated pipeline to evaluate two ported patterns: 'Dynamic Retry Delay' (DRD) and 'Open Only When Necessary' (OOWN). With this, we found no evidence that the DRD pattern consumes less energy than its antipattern, while the opposite is true for OOWN. Data and Material: https://doi.org/10.5281/zenodo.8404487

CCS CONCEPTS

• **Software and its engineering** → **Empirical software validation**.

KEYWORDS

Green Software Engineering, Energy patterns, Web applications, Software sustainability, Coding Practices, Energy consumption

LAY ABSTRACT

The information technology sector significantly affects the climate. With our increasing online activities, from chatting to accessing medical history, software powering these services requires to be energy-efficient. Researchers in software engineering have been

exploring green coding practices, or energy-specific design patterns (aka energy patterns) to make software more eco-friendly. While such energy practices have been explored for other domains including Mobile, Web applications have been somewhat overlooked, despite our daily heavy internet use. We focused on the existing energy patterns from Mobile applications to Web applications. To validate these ported energy patterns, we interviewed six professional web developers from various companies. Then, we tested some patterns to see if these energy patterns indeed save any energy. Our results showed that developers are unaware of the energy practices and some patterns did not make a noticeable difference, while others consume more energy than their counterpart. In a nutshell, our work highlights the knowledge gap between green coding research and industry and emphasize the need to understand the trade-offs in energy practices for sustainable digital future.

1 INTRODUCTION

The ICT sector is estimated to generate up to 5.5% of world carbon emissions and to consume 20% of all electricity [3]. Indeed, from healthcare to communication, every industry prominently runs on software, thus understanding and developing energy-efficient software is urgent.

In this context, the Software Engineering (SE) research community has started investigating *green coding* and energy patterns for source code [17, 24]. Energy-specific design patterns for source code (henceforth, *Energy Patterns*) are best practices developers use to make their source code energy-efficient [24]. While researchers have developed catalogs of energy patterns for Mobile applications [10] and for deep learning libraries [35], some domains are still yet to be covered, prominently the Web domain, which is particularly relevant as its energy consumption is ever increasing [19].

Our goal is to gather and evaluate Web-specific energy patterns. To this aim, we first attempt to port existing Mobile energy patterns [10] to the Web domain. Then, to challenge our ported patterns, we discuss them with six professional Web developers, by means of in-depth structured interviews. In particular, we discuss how understandable these patterns are, how they are perceived, and whether they can be located in source code of Web applications. Consequently, we collect concerns regarding various patterns or their respective antipatterns (not having the pattern) and guidelines for locating them in the source code. Based on the guidelines, we analyze the source code of a company and thus show which of them are easy to locate or not. Finally, we measure the impact of two energy patterns to see whether they indeed save energy.

Our results show that most Mobile energy patterns (16 patterns) can be directly ported to the Web, while a few (four patterns) are only partially applicable to Web applications (*e.g.*, 'Power Save

Mode'), and two of them cannot be ported because they are inapplicable (*e.g.*, 'WiFi over Cellular'). From the interview study, we start by confirming that past results on developers' limited awareness of energy aspects [28, 42] also apply to our Web developers. Once we described the patterns, participants report to understand them and many expressed concerns for energy antipatterns, specifically for functional antipatterns. Developers then were able to provide guidelines to locate the patterns in the source code. Following their guidelines, we could identify eleven antipatterns and nine patterns in their own source code. Some energy patterns, *e.g.*, 'Push over Poll', could be easily found in the source code, while others, *e.g.*, 'Reduce Size', 'Enough Resolution' were harder to identify.

Finally, concerning the energy impact of 'Dynamic Retry Delay' and 'Open Only When Necessary', we found no evidence that the DRD pattern consumes less energy than its antipattern, while the opposite is true for OOWN.

With this paper, we make the following main contributions:

- a porting of existing energy patterns for Mobile to the Web domain;
- a report on the perception of six professional web developers on the ported energy patterns;
- a guideline to find Web energy patterns in source code based on developers' experience and confirmed against these developers' code bases;
- empirical evidence on the effects of antipatterns on energy consumption for 'Dynamic Retry Delay' and 'Open Only When Necessary';
- a replication package (RP) [31] containing detailed patterns with context and discussions, interview results, and an automated pipeline to quantitatively evaluate energy patterns.

2 STUDY DESIGN

In this section we motivate and present our research questions and detail the research method we use to answer them.

2.1 Research questions

While there's been a growing focus on green coding practices in Mobile and embedded systems [24, 35], the Web domain remains under-explored. Cruz *et al.* developed a catalog of 22 Mobile energy patterns [10]. Our aim is to expand current scientific understanding by exploring the portability of these Mobile energy patterns to Web applications. Insights from this study can equip Web developers with tools to integrate energy patterns into their codebase and recognize antipatterns (inefficient practices) to minimize energy use. Hence we ask:

> RQ_1: To what extent can we port energy patterns for mobile applications to web applications?

Developers are the main target of energy patterns, because these patterns—akin to design patterns—should provide a roadmap for developers to code refactoring and reduced energy consumption. So far, previous work that has proposed catalogs of energy patterns [2, 10, 35], did not investigate how professional developers perceive and understand these patterns, and how easily they can be located in source code. We try to fill this gap in our context: We actively engage industrial developers, inviting them to assess and challenge the energy patterns we have ported from the Mobile domain to the Web one. Through this endeavor, we aim to shed light on their perspectives and hurdles regarding energy consumption in software, the ported energy patterns, as well as to measure their commitment to dedicating resources for improving the energy efficiency of Web applications. We ask:

> RQ_2: How do developers perceive the ported web energy patterns and locate them in source code?

In RQ_1 we ported Mobile energy patterns to Web applications. Whether these energy patterns indeed impact the energy consumption of a Website is not explored. Previous work has investigated the energy impact of software specific to its code [7], execution [30, 32], third-party libraries [33], programming languages [29], and Mobile applications [20]. Researchers have also measured the impact of test smells or code smells and their associations with energy consumption. We aim to investigate the actual impact of our ported energy patterns, as validated by the professional developers. Such analysis can help practitioners make informed choices to reduce the energy impact of their Web applications. We focus on two specific patterns (for the reasons mentioned later) on and ask:

> RQ_3: How do *Dynamic Retry Delay* and *Open Only When Necessary* impact energy consumption?

2.2 Methodology

2.2.1 RQ_1: Energy Patterns for The Web. Previous work has identified various coding practices for maintaining source code (*e.g.*, design patterns), or improving the energy consumption (*e.g.*, energy patterns). Cruz and Abreu [10] have identified a set of energy patterns developers apply intending to improve the energy aspect of their applications. They have analyzed 1,783 Mobile applications from both Android and iOS from GitHub. They identified 22 energy patterns appearing in 421 instances [10] (described in the RP [31]). Since they provide a recent and the largest catalog of such patterns, we attempt to port these patterns to the Web. We investigated if and how many of them are applicable on the Web domain. To port these energy patterns to the Web, we restricted our focus to a client-server architecture for the Web since various Web architectures or frameworks can have various implementations of the patterns.

For each pattern, one author studied the original pattern from Cruz and Abreu [10] with its description and examples and attempted to formulate the Web counterpart. To find the Web example, she checked the code examples given by Cruz *et al.* in their RP and searched similar code snippets or the pattern name if exists into the Google search engine. Since the goal is to find similar concepts or examples, she also looked at gray literature like guidelines, documentation, and blogs. Once the patterns were formulated, another author reviewed each definition with its examples and pointed out the differences. We found the differences mainly in definitions of 10 patterns, *e.g.*, 'Dark UI Colors' in Mobile focuses on saving battery for AMOLED screens while for Web, battery is not the primary focus, therefore we phrased the Web pattern accordingly. The detailed discussion on differences is provided in Appendix in the RP. The

third author reviewed the differences and resolved them by mutual discussion. This process led to energy patterns for the Web, with description and examples. Certain patterns exist in the front-end part of the application, while others exist in the back end; we dissected them based on their existence in the front- or back-end part of the application. This helps identify which patterns require access to source code and which can be identified by merely exploring the Website.

2.2.2 RQ_2: Developers' Perceptions on Web Energy Patterns.
To challenge our mapped patterns in practice, we investigated the perception and concerns of professional developers. Following a convenience sampling approach, we interviewed six expert Web developers from four different companies. Three of them work at the same company,[1] thus allowing us to perform a case study. The company provides a web portal where the customers can see their electricity consumption data. The portal is a web application with a client-server architecture developed mainly using the front end technologies: `NodeJs`, `VueJs`, `Element Plus`, `TypeScript`, `Axios`, and the back end technologies: `Python` and `Django` framework. The whole web application consists of 57,000 lines of code (LoC) for the front end and around 1,590,000 LoC for the back end. The three other developers we interviewed belong to three different companies yet focus on similar technologies.

Various methods have been used to investigate developers' perception of energy aspects, such as mining software ecosystems or surveying developers [17, 42]. Fink [15] explained that "*the interview enables to collect information from people to explain their attitude, behavior, and knowledge.*" Therefore we considered it as the best approach to collect our data. Figure 1 shows the design of our interview study, as inspired by Fink [15] and Yamashita and Moonen [42]. We followed various steps to set up the interview: (a) establish the goals and questions, (b) prepare an interview instrument, (c) organize the interview administration, and (d) analyze the responses and collect the guidelines based on them.

Establishing the Goals and Questions. Three authors outlined the goals, mainly focusing on challenging the ported energy patterns with the help of professional developers. We also aimed at collecting guidelines to identify the valid patterns in their source code, and at understanding whether they would invest resources to implement energy patterns or remove antipatterns. Therefore, we defined intermediate goals, *e.g.*, (1) insights on energy pattern awareness, (2) introduction of patterns as shown in Figure 1. Based on the main goals, we chose an *exploratory structured interview* that included both closed and open questions to understand their motivation or choices like why certain energy patterns concerned them.

Preparing Interview Instrument. We began by asking about their awareness of energy patterns and their sources of information. Then, we challenged the ported patterns by introducing them to developers, examples to clarify, and their antipatterns. We gauged their level of concern regarding the presence of antipatterns. We used a five-point Likert scale and designed various open-ended questions to understand their reasons of concerns, where in the source code these energy patterns can be found, how easy it is to identify them, their desired commitment to invest resources, and

---

[1]The identity of the company cannot be disclosed due to a non-disclosure agreement.

their past use of related tools. At the end, we asked demographic questions to understand their background, *e.g.*, role in the organization, years of experience in Python, JavaScript, and TypeScript (languages used in the company) and the programming paradigms as certain patterns can be specific to a language specifically in trusting the guidelines proposed by developers to search energy patterns in source code. The questions are provided in RP [31].

Organizing the Interview. We conducted a personally administered interview using anonymous Google Forms. The questions were asked in person and developers filled their responses directly. For the longer open-ended questions (*e.g.*, where these energy patterns can be found in their source code), the author transcribed developer responses on-site to avoid recording the sessions, which would have raised confidentiality concerns.

Analyzing the Responses and Guidelines. We visually analyzed the graphs of the closed-ended questions, while, for the open-ended ones, we conducted axial and open coding [37]. That is, we extracted themes and excerpts from the developer's responses. Since one question focused on collecting the guidelines, each author independently analyzed half of the responses and reviewed the other half. Both authors handled their disagreement by mutual discussion. Finally, we verified the guidelines by checking the presence or absence of energy patterns in the source code of the company.

2.2.3 RQ_3: Impact of Energy Patterns.
Here, we investigated whether energy patterns indeed impact the energy consumption of a Website. Ghaleb [18] showed the accuracy of software tools in estimating energy consumption. Therefore, to estimate power consumption, we used Intel PowerLog version 3.7.0 which relies on Running Average Power Limit (RAPL) interface. This approach, widely used in green software studies [6, 7, 17], measures real-time power information of macOS based on the energy counters in the Intel Core processor without any hardware instrumentation. However, it does not measure energy use of network traffic, display, or an isolated process. Consequently, certain patterns could not be measured. Various criteria are detailed below to include or exclude patterns.

We included each pattern that:
- I1: is processor-dependent
- I2: is measurable on the client-side browser

We excluded each pattern that:
- E1: is Mobile device-specific (*i.e.*, 'Power Save Mode', 'Power Awareness', 'Race-to-idle', 'WiFi over Cellular', 'Sensor Fusion', 'No Screen Interaction'),
- E2: is network related as the selected energy profiler does not measure network impact (*i.e.*, 'Reduce Size', 'Batch Operations', 'Cache', 'Push over Poll', 'Enough Resolution'),
- E3: is display-specific (*i.e.*, 'No Screen Interaction', 'Dark UI Colors', or 'Avoid Extraneous Graphics and Animations'),
- E4: is user-related as it can not be controlled which parameters to change (*i.e.*, 'User Knows Best', 'Inform Users', 'Manual Sync – On Demand'),
- E5: is power model related as it affects the whole device and thus is not controllable for only browsers (*i.e.*, 'Power Save Mode', 'Power Awareness'),
- E6: can not be simulated systematically or lacks empirical data to support its parameter selection (*e.g.*, what threshold of logs
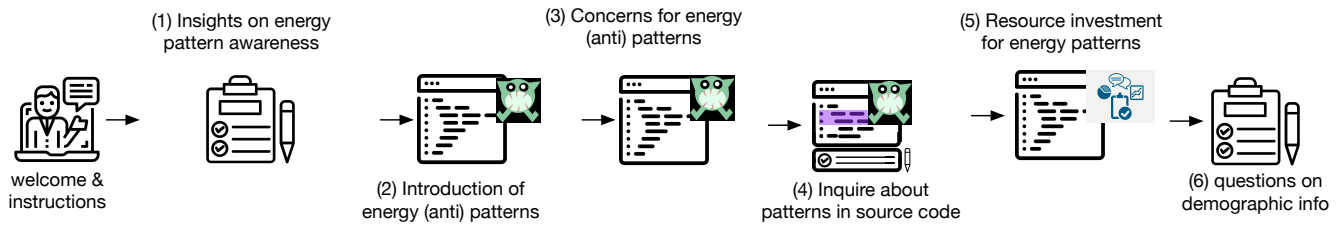
Figure 1: Interview study design to answer RQ2: Industry Insights on Energy Patterns
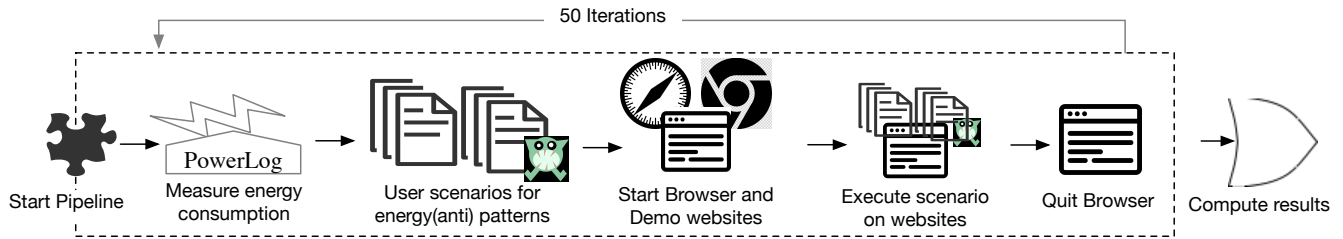

Figure 2: Study design to answer RQ3: Impact of Energy Patterns

should be considered for the Suppress Logs pattern, what amount of notification should be considered for 'Push over Poll', what rate for 'Decrease Rate', and which tasks should be considered abnormal for 'Kill Abnormal Tasks)',

E7: similar patterns but no specific Web code could be extrapolated, *e.g.*, 'Avoid Extraneous Work' and 'Avoid Extraneous Graphics and Animations' are similar to 'Open Only When Necessary', but could not find precise web code for it.

Based on these criteria, which ensured the most reliable measurements with the current knowledge, we were left with two patterns, namely 'Dynamic Retry Delay' (DRD) and 'Open Only When Necessary' (OOWN).

Pipeline Setup. We automated our test process as shown in Figure 2, using Python. The test process initiates PowerLog with Selenium, simulating a user scenario on the demo website. PowerLog measures the energy consumption of the executed commands storing parameters in log files. These user scenarios are coded in Java, and the website interaction is automated using Selenium Web-Drivers (for Chrome and Safari). The browsers ran in incognito mode to ensure no cache or past data usage. The demo Websites built using HTML, JavaScript, and PHP, enabled us to simulate the scenarios for the selected patterns. We analyzed the generated logs to obtain the results.

Implementation of User Scenarios. To measure the impact of each pattern, we simulated two user scenarios, as shown in Figure 2.

DRD: We open the browser, visit the demo website, initiate an HTTP request to an external Web server, which waits for two seconds and then returns an HTTP 503 error code (the server is unavailable to handle the request or the attempt to access the resource failed). Then, the Website retries to access the server. In the pattern scenario, the retry interval increased exponentially after each failed attempt (referred to as *dynamic retry*), while in antipattern scenario, it remained constant (static retry). Dynamic

retry interval is based on the *EcoAndroid* tool [2], *i.e.*, 1 second, 2 seconds, 4 seconds *etc.* and the static retry interval is fixed as 3 seconds. The browser quits after 90 seconds to ensure that the scenarios do not execute infinitely.

OOWN: We focused on content loading (specifically images). In the pattern scenario, the website loads images when the user scrolls or enters the viewpoint (*lazy* loading) and defers loading non-visible content. In contrast, the antipattern scenario loaded all images at once when the user opens the website (*eager* loading). Both scenarios lasted approximately a similar time, *i.e.*, 45–48 seconds, and then the browser quits.

Execution of User Scenarios. We ran the pipeline on a Mac mini desktop computer with a 2.3 GHz Quad-Core Intel® Core™ i7 processor, 16 GB DDR3 RAM with 1600 MHz, 1 TB SATA HDD, running macOS Catalina 10.15.7. We run the scenario on two browsers, *i.e.*, Chrome version 113.0.5672.92 and Safari version 15.6.1. For each scenario, we collected `Cumulative energy` (energy consumption of the processor), `elapsed time` and (`Package Temperature`) in the log files [25].

To mitigate energy consumption variability, each scenario was run for 50 iterations for both pattern and antipattern for each browser, thus 400 times; this took approximately 14 hours. Overall, we took several preventative measures:

P1: Running each scenario for 50 iterations with a 30-second interval (cool-down periods) for temperature consistency.

P2: Deactivating all non essential applications, *e.g.*, including WiFi, Bluetooth.

P3: Limiting the number of accessories, *e.g.*, Mac mini is not integrated with a keyboard, mouse, or display.[3]

P4: Disabling sleep mode (set the sleep time to never).

---

[2]https://plugins.jetbrains.com/plugin/15637-ecoandroid
[3]The pipeline was started with a remote connection (Screen sharing option in macOS), and the remote connection was disconnected as soon as the pipeline started.

P5: Randomizing the scenarios of a pattern to avoid external influences.

P6: Keeping the computer continuously plugged in.

3 RESULTS

3.1 *RQ₁*: Web Energy Patterns

We could port 20 Mobile energy patterns to the Web domain, as shown in Table 1. Of these, 16 patterns fit web applications directly, four patterns mapped partially, and two patterns did not apply (particularly if the website is accessed via desktop or laptop rather than Mobile). Patterns like 'Sensor Fusion' and 'WiFi over Cellular' are Mobile-specific as they rely on features such as accelerometer or cellular networks. As the websites are also accessed on desktop or laptop, we marked these patterns as inapplicable. The porting of patterns required understanding their applicability and trade-offs in specific scenarios, *i.e.*, which patterns should be considered for what applications and when.

Table 1 shows the patterns classified based on whether they are relevant to the client-side (front end) or server-side (back end) of a Web application. This information also helps pinpoint where these patterns can be found. For example client-side patterns (*e.g.*, 'Dark UI Colors', 'Power Save Mode', 'User Knows Best') can be found by looking at the website without inspecting the source code.

```
const audio = document.querySelector("audio");

// Handle page visibility change:
// - If the page is hidden, pause the video
// - If the page is shown, play the video
document.addEventListener("visibilitychange", () => {
  if (document.hidden) {
    audio.pause();
  } else {
    audio.play();
  }
});
```

Listing 1: Mozilla API handles visibility change [9]

Table 1 provides a breakdown of each adapted pattern accompanied by a representative example. As an illustration, the Mozilla's Page Visibility API, showcased in Listing 1, exemplifies the 'Avoid Extraneous Work' pattern. This API informs users if a web page is currently visible, enabling functionalities like pausing audio or video when the page is not in view. Moreover, this API also embodies the 'OOWN' pattern, where resources, like audio, are activated only when necessary—specifically, when the user can see the page. These examples highlight the manifestation of such patterns on the Web, guiding us in spotting them in future web applications.

3.2 *RQ₂*: Developers on Web Energy Patterns

Awareness. Figure 3 shows that most of the interviewed developers are unfamiliar with energy patterns. Only a few possess an understanding of it, primarily gained from university seminars or through their peers. Some interviewees highlighted that discussion forums and social media have contributed to their knowledge; this finding suggest that to enhance awareness and influence within the industry, researchers should consider exploring strategies to effectively engage with online social platforms.
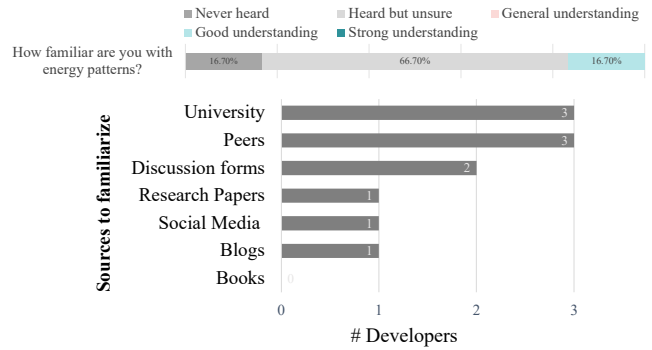
Figure 3: Awareness and sources of Energy Patterns

Ported Energy Patterns. When examining the ported patterns, interviewees found certain patterns easy to understand, such as 'Dark UI Colors', 'Push over Poll', 'Suppress Logs', or 'DRD'. However, for other patterns, many asked clarification questions and examples as they found them hard to imagine in the Web domain. This was mainly due to some patterns being applicable to many contexts (*e.g.*, 'Decrease Rate', 'Reduce Size'), hard to pinpoint to an example (*e.g.*, 'Avoid Extraneous Work'), or unheard of (*e.g.*, 'Manual Sync – On Demand', 'No Screen Interaction'). For example, concerning 'Avoid Extraneous Work', P5 explained: "It's hard to pinpoint, this could be anywhere or everywhere." For 'Avoid Extraneous Graphics and Animations', P3 answered: "Nothing comes to mind," when asked to locate it in the source code. They also asked clarification questions for architecture (*e.g.*, "is it MVC?"), framework, libraries, and programming languages. Although, we restricted our focus on client-server architecture and specific languages (that the interviewees are expert of), such factors seem to play a role in porting them. Furthermore, when discussing patterns, Web developers tend to prioritize application functionality and performance, in contrast to Mobile developers who show greater concern for battery life and cellular network considerations [11, 21].

Concerns. Figure 6 reveals significant interviewees are somewhat or moderately concerned with antipatterns in their code. Most developers are especially wary of patterns that could affect the main functionality (functional requirements) of the website, or can have an impact on overall quality or security of software, *e.g.*, the antipatterns for 'Avoid Extraneous Work', 'Dynamic Retry Delay', 'Kill Abnormal Tasks' *etc.* Many of these patterns recommend avoiding extra work or killing abnormal tasks as they can slow down the performance or even block the main functionality. On the contrary, interviewees are least or slightly concerned for antipatterns like 'Dark UI Colors', 'Inform Users', or 'Power Awareness' as the patterns do not really impact the main functionality and mainly enhance user experience—having these patterns in source code is not a high-priority tasks for developers. Some patterns like 'Race-to-idle', 'Decrease Rate' seem neutral to developers.

A thematic analysis (excerpts shown in the RP [31]) of developer feedback showed that their concerns (somewhat or moderately concerned with energy antipatterns) stems from the energy demand of cloud computing and the link between computational complexity and energy antipatterns. Whether such perceptions correspond

Table 1: Energy patterns with applicability to web, classified to client (C) or Server (S), description, and examples.

Pattern	Applicability	C/S	Description
Avoid Extraneous Graphics and Animations	✔	S	Use battery-intensive graphics or animations with moderation. *e.g.*, A website not loading heavy graphics until users interact with them.
Avoid Extraneous Work	✔	S	Present only relevant data or perform tasks that have a direct impact on the user experience. *e.g.*, Mozilla's API in Listing 1 informs users for the page visibility to let audio/video pause.
Batch Operations	✔	S	Combine multiple operations to perform batch processing. *e.g.*, Web API from Microsoft to group several operations into a single HTTP request [12].
Cache	✔	C	Utilize caching mechanisms to reduce network load. *e.g.*, A code example to cache an API response in the local storage [31].
Dark UI Colors	✔	C	Provide a web application with the dark UI color theme. *e.g.*, Facebook provides an option on the website to switch to a dark theme.
Decrease Rate	✔	S	Increase the time interval between requests to the backend. *e.g.*, Library website refreshes the book availability only a few times a day.
Dynamic Retry Delay	✔	S	Use a systematic retry increasing time interval after each failed attempt to a resource, such as a database, or network. *e.g.*, In the Fibonacci series utilize a retry mechanism API to handle abnormal conditions [31].
Enough Resolution	✔	S	Provide high-accuracy data only when strictly necessary. *e.g.*, *AVIF* and *WebP* image formats reduces file sizes in browsers [8, 14].
Inform Users	partially	C	Inform users of the energy-intensive operations on the website. *e.g.*, Autoplay feature on YouTube consumes a significant amount of energy, but the user is not informed.
Kill Abnormal Tasks	✔	S	Provide means of interrupting energy-hungry operations. *e.g.*, A timeout to interrupt an abnormal operation [31].
Manual Sync – On Demand	✔	S	Perform tasks exclusively when requested by the user. *e.g.*, YouTube, with Autoplay feature off, plays song only when user clicks on it.
No Screen Interaction	partially	C	Whenever possible, allow interaction without using the display. *e.g.*, Users interacting with music player websites via other methods (*e.g.*, audio, keyboard) [31].
Open Only When Necessary	✔	S	Open resources or services only when necessary. *e.g.*, A camera application opens the camera only to capture an image [31].
Power Awareness	partially	C	Enable or disable certain website functionalities based on the power status. *e.g.*, Chrome offers *Energy Saver* mode [41], which limits background activity and visual effects (*e.g.*, animations and videos) when the device's battery reaches 20%.
Power Save Mode	partially	C	Provide an energy-efficient mode for the website. *e.g.*, BooHoo has introduced an energy-saving mode to its website [13].
Push over Poll	✔	S	Use push notifications to receive updates from resources instead of actively querying resources (i.e., polling). *e.g.*, The server pushing the updates rather than the client requesting them [31].
Race-to-idle	✔	S	Release unnecessary resources and services. *e.g.*, Release camera after a video call, unused variables via garbage collection in source code [26].
Reduce Size	✔	S	Transmit only necessary data. *e.g.*, An HTTP request wherein strings bigger than 1KB get compressed [31].
Sensor Fusion	✘	-	Use data from low-power sensors to infer whether new data needs to be collected from high-power sensors.
Suppress Logs	✔	S	Keep the logging rates low. *e.g.*, The logging level shown in the console output as 'warning' [31].
User Knows Best	✔	C	Allow users to customize preferences for energy-critical features. *e.g.*, BooHoo [13] let users choose between certain features (bright screen and full availability of images).
WiFi over Cellular	✘	-	Delay or disable heavy data connections until the device is connected to a WiFi network.

to reality is currently unknown. Some developers expressed indifference about energy consumption on local machine; as P5 (who rated 'somewhat concerned') put it: "It depends if its locally run, or on the cloud. If local, I don't really care. If GPUs are involved or the cloud, I am more concerned because I read a lot about energy consumption of large data centers, and I know that GPUs consume a lot of power."

This feedback indicates the knowledge gap about software energy consumption on the individual level or local machines. Given the high number of pet projects built by developers on GitHub [4], such systems collectively can have a large impact. Moreover, some
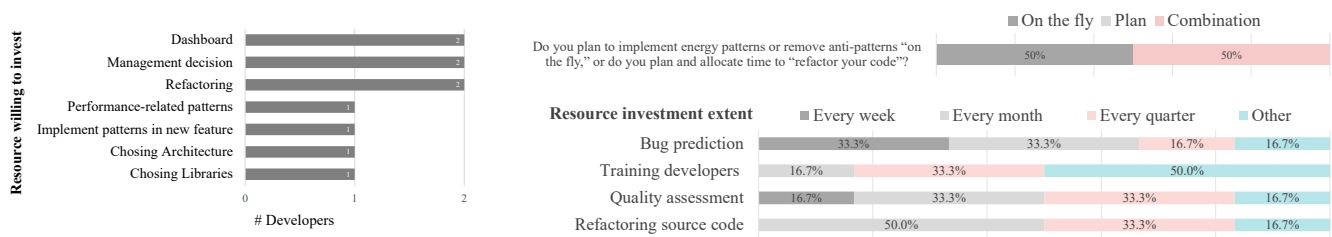
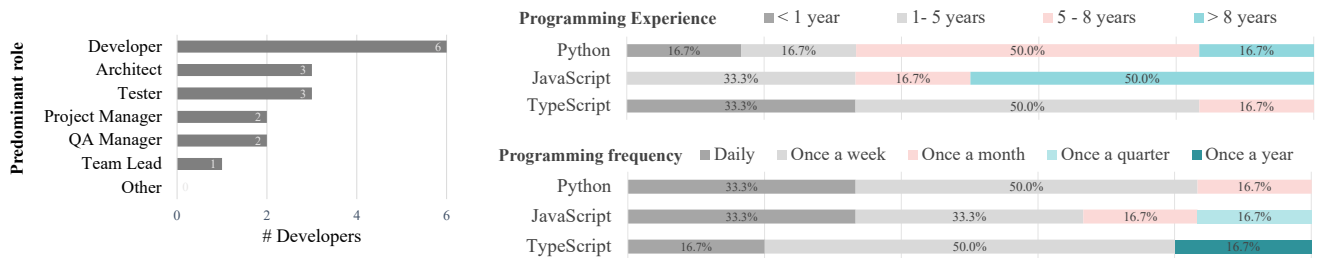**Figure 4: Willingness of developers in investing resource for energy patterns**
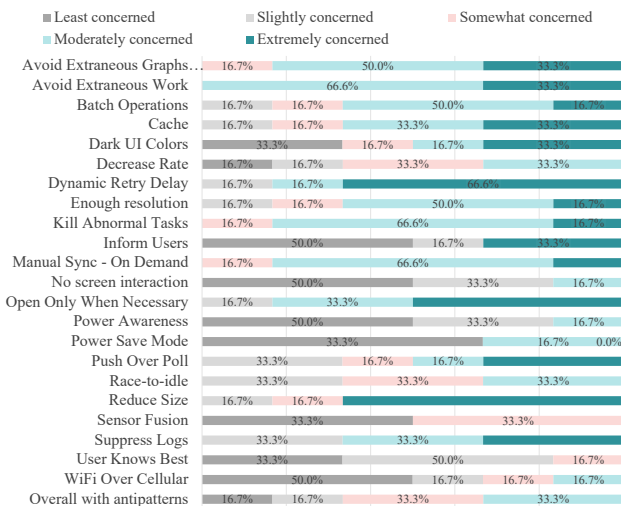

Figure 5: Participants' demographics

Figure 6: Developers concerns for energy antipatterns

developers prioritize application and performance optimization over energy, as stated by P6 (who rated 'somewhat concerned'): "Application optimization and performance are still more important than energy optimization." It calls for more research knowing whether this preference stems from lack of knowledge about energy issues or entrenched focus on application performance throughout the development life cycle.

Locating Patterns in Code. Finding these patterns or antipatterns in source code can be challenging as some are general coding practices and can exist in various components. Developers require a thorough understanding of their web applications. When inquired

about finding these patterns in their code, Table 2 shows instances of developer guidelines on locating them. Following their guidelines, we indicate (in column *Found*) if a pattern or antipattern could be found in the company's source code. For some patterns, they provided a rationale (in *Rationale/Guidelines* column) whether an antipattern exists in their source code or if it is inapplicable, as shown in the table. Not all instances were found, but they could point out potential code components, *e.g.*, modules, classes, functions, or specific annotations. Concerning 'OOWN', one developer (P1) explained: "Our frontend is a single-page application, It loads everything at once and there is a lot of data traffic to be able to render the page. We load certain information (in accessing the dashboard) that is not shown to the user (antipattern), while we don't load dynamic charts (pattern)."

Figure 4 reveals developers' interest in optimizing energy aspects. They expressed willingness to implement energy patterns in their source; but noted it is often a management decision. They wish to have a dashboard reflecting their source code's energy consumption so that they can identify energy hot spots. While half of the interviewees stated to be open to refactor existing code, most prefer to incorporate energy patterns for new source code, *e.g.*, in implementing new features or choosing a new library or architecture based on the energy aspect, rather than refactoring existing code for this specific reason. Indeed, their preference is to address energy concerns spontaneously rather than allocating specific times, as shown in Figure 4. When asked about resource commitment, they showed interest in some activities *e.g.*, quality assessment, refactoring source code, or identifying energy antipatterns (bug prediction) on a weekly or monthly basis, but preferred training developers further apart, like quarterly or yearly.

About the knowledge on energy-related tools or past use of such tools , they mentioned a few tools but not intended to explore the energy aspect, *e.g.*, *Silk* for data transmission, *tick stack monitoring*,

Table 2: Guidelines/Rationales on finding a pattern (P) or antipattern (AP), or not applicable (NA) in the company

Pattern	Found	Rationale/Guidelines to locate the pattern	Code Reference
Avoid Extraneous Graphics and Animations	P, AP	P1, P3: "*we do not use animations, but charts only.*"	Library, component
Avoid Extraneous Work	P	P1: "*we implement custom functions, e.g., compute bills which are needed.*"	Function
Batch Operations	P	P1 : "*we use batch operations in testing, Redis jobs.*"	Functions
		P2: "*look for* `promise.all()`"	Functions
Cache	P	P3: "*most API calls are cached, check* `@memorize` *annotation.*" *	Annotation
Dark UI Colors	AP	P1: "*it is not implemented.*".	Module
Decrease Rate	NA, P	P1: "*we have no sensors so no sync rates (NA).*"	
		P3: "*there is a refresh token in logging (P).*"	Component
Dynamic Retry Delay	AP	P2: "*there are no delays, only maximum retries for connecting to the API.*"	Module
Enough Resolution	NA, AP	P1: "*it is not applicable to us. We use a third-party library that has this feature (NA).*"	Library
		P2: "*API responses have high resolution (AP). We have certain resolutions due to legal reasons.*"	
Inform Users	AP	P1: "*not implemented currently.*"	
Kill Abnormal Tasks	P	P2: "*we use timeouts to cancel a computation process.*"	Annotation
Manual Sync – On Demand	P	P1: "*we fetch data if user requests.*"	Component
No Screen Interaction	NA	P1: "*it does not make sense for our case*"	
Open Only When Necessary	P, AP	P1: "*the frontend is a single page application, we load certain information that is not shown to the user (AP), while we don't load dynamic charts (P).*"	Classes
Power Awareness	NA	P2: "*it is not applicable to our application.*"	
Power Save Mode	AP	P1: "*we do not have options for the user.*"	
Push over Poll	AP	P1: "*when computation happens in the server, the frontend requests the state of computation constantly.*"	Component
Race-to-idle	P	P1: "*we use tear-down functionality in testing (provided by the framework)*"	Module
Reduce Size	AP	P1: "*we sometimes send big payloads from the backend to the frontend to deal with.*"	Class
Suppress Logs	AP	P1: "*we do not do enough logging.*"	File
User Knows Best	AP	P1: "*no options for the user, we show all charts*"	

and Profilers for minimizing database requests. For instance, P6 stated: "I use the 'Network / Performance' tab in Developer Mode in the browser, but it is more for performance or debugging reasons than for energy patterns." Such instances show the limited awareness about existing ones and the dearth of dedicated tools for energy measurement.

Figure 5 shows the demographics of the interviewees in terms of role and experience. More than half of the developers have more than five years of experience in Python and JavaScript while lacking experience in TypeScript. Also, the majority of developers program frequently (daily or weekly). Only one developer reported once in a year programming activity in TypeScript.

3.3 RQ_3: Impact of Energy Patterns

Figure 7 shows the cumulative package energy (CPU) consumption with median values for *DRD* pattern (dynamic interval) and its antipattern (static interval) for Safari and Chrome browsers. The energy difference between both scenarios is insignificant, *i.e.*, the antipattern consumes slightly more energy, but we can see a difference between Safari and Chrome browsers. Looking at the elapsed time (time to execute the scenario), we found Chrome took an average of approximately 95 seconds, while Safari took approximately 109 seconds. Such differences are mostly due to the Safari WebDriver taking longer to start than the ChromeDriver. However, once started, Safari consumed less energy than Chrome. We found similar differences in package temperature, where the average temperature for Safari (35 degrees) is less than Chrome's (38 degrees). Previous work has indicated that there is a negative correlation between the high temperature of the device and battery life [22].
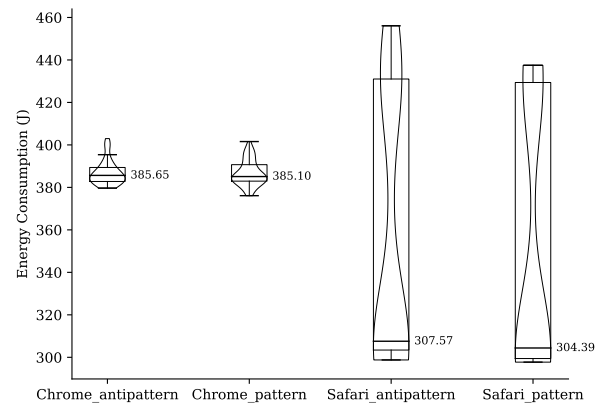
Figure 7: Cumulative energy for DRD

Similarly, Figure 8 shows the cumulative package energy (CPU) consumption for 'OOWN'. We found that the 'OOWN' pattern (lazy loading) consumes more energy than its counter scenario (eager loading). Also, Chrome (both pattern or its antipattern) consumes more energy than Safari. While this may seem counter-intuitive at first glance, since an energy pattern is expected to consume less energy than its antipattern, the reason can be the implementation of the 'OOWN' scenario. To achieve lazy loading of content, scrolling activities must be constantly monitored to detect when new content
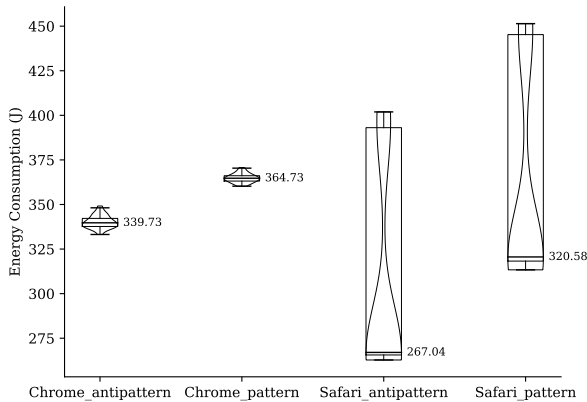
Figure 8: Cumulative energy for OOWN

needs to be loaded, and several calls to the server must be executed to load the content itself. However, the real advantage of 'OOWN' is that certain resources do not have to be loaded at all if a user does not need them. In our simulating scenarios, the same number of images (resources) is loaded for both pattern and its antipattern.

Like 'DRD', we observed the elapsed time for 'OOWN' and found that Chrome took an average of approximately 57 seconds, while Safari took approximately 67 seconds. We found that similar to 'DRD', the average temperature for Safari (43 degrees) is less than Chrome's (45 degrees) for 'OOWN' as well.

Overall, our results show that Safari consumes less energy than Chrome. One of the reasons for this can be that the measurements are done on a machine running macOS, and Safari is optimized for macOS and Apple Hardware. Also, Chrome uses V8 JavaScript engine, while Safari uses JavaScriptCore (aka Niro). The way these engines handles JavaScript can affect energy consumption. Our results also show that the energy efficiency of energy patterns may depend on the specific implementation and use case and the frequency of those scenarios. Therefore, we advise researchers to carefully design scenarios to measure the patterns.

4 IMPLICATIONS AND FUTURE WORK

For Researchers and Developers. In RQ1, we aim to catalog energy patterns for Web apps. We build on the previous work that used GitHub (Mobile repositories) to identify Mobile energy patterns [10]. Others have used platforms like Stack Overflow to identify energy patterns [35]. We plan to expand the catalog by exploring other sources like Stack Overflow and web specific GitHub repositories. Then, verify the catalog with developers to ensure that they are adapted appropriately to the web context.

In RQ3, we measured the energy impact of two patterns. We observed that adapting patterns requires understanding trade-offs in specific scenario. For instance, 'Race-to-idle' pattern suggests releasing the resources as soon as possible, but holding onto them might save energy in some situations, *e.g.*, if the resource request is frequent, then it can consume more energy in the process of

requesting and releasing the resources continuously. We plan to examine these trade-offs in real-world scenarios. Some patterns may save energy at the cost of other software attributes, *e.g.*, performance, code comprehension, or maintainability. Previous work has investigated the effects of code smells, design patterns, or antipatterns on maintainability [1]. They also investigated the effect of energy-efficient changes on maintainability and found that such changes hinder the maintainability of Android applications [11]. Whether such findings are specific to maintainability or Mobile apps, is yet unknown. Knowing such trade-offs can help developers in choosing when to include or avoid patterns in their source code to build green web applications.

For Tool Designers. RQ2 findings show developers' interest in exploring the software energy usage, *e.g.*, via Dashboard or identify antipatterns in their source code. However, many are unaware of existing tools or find them unsuitable for energy evaluation. Many developers mentioned tools like *Silk*, or *tick stack monitoring*, but use them for debugging or device performance rather than energy analysis. Future work should explore if this gap is due to unawareness or the need for better tools.

For Organizations. The results of RQ2 show that developers learn about energy efficiency from peers and University seminars. Developers who are somewhat concerned for antipatterns are motivated due to company mission or team leaders for achieving sustainability. Future work can investigate the awareness and role of the team leaders on source code sustainability. Companies can leverage this by offering training programs and urging team leaders to adopt green coding practices and make other developers in the team aware of them. Also, integrating software energy efficiency in the University curriculum can educate students or future developers learn the importance of green software engineering and energy-related trade-offs in development.

5 THREATS TO VALIDITY

Catalog of Patterns. There is a possibility of more patterns that can be suitable to the Web domain. We have searched the literature on the existing Mobile patterns and selected the largest catalog by Cruz and Abreu [10] to adopt. Also, some patterns are specific to domains and code implementation, while others are more general, as they (*e.g.*, 'OOWN') are relevant across various code components *e.g.*, from front-end to database to network components; thus require adaptation to a broader context. We have, for now, kept the patterns generalized and indicated one example for each pattern.

Interview Study. For the interview study, we interviewed three developers from one company and three from different ones. As a result our convenience sample is not representative of web developers. Our choice was dictated by the need to conduct in-depth in-person interviews (60–90 minutes) and to access to their source code to verify their guidelines. Given that reducing carbon emissions and our energy footprint is a pressing societal need, we cannot rule out that some of the interviewees could have given specific answers due to social desirability bias [16]. Overall, further studies are needed to both challenge our findings with a more general population (questions to consider include: How familiar are developers with energy patterns? Do they share similar concerns about

antipatterns? Can one easily locate Web energy patterns in source code?) and lower the risk of social desirability bias, by means of guaranteed anonymity. Also our analysis of their feedback can pose a threat. To minimize this issue, two authors independently analyzed the responses and a third author reviewed the disagreements. We handled the disagreements by mutual discussion.

Tools. As we have conducted the experiment on MacOS, it can introduce a bias on our power consumption estimation. We replicated the experiment additionally on Chrome Browser to minimize this risk. We suggest future work to replicate our work on other operating systems and browsers. For 'DRD' scenarios, we drew inspiration from the EcoAndroid tool and used Intel Power Gadget for energy consumption measurement. Any error in these tools can influence our results. We tried to minimize this risk by ensuring to be consistent with how these tools have been evaluated and used in previous work. Although Intel Power Gadget is heavily used in previous work, various factors, such as electricity outage, or room temperature can affect energy consumption. We maintained our conditions consistent, *e.g.*, by keeping the desktop computer plugged in and at room temperature for the entire duration of the experiment. Since the temperature can play a major role in such experiments, we included warm-up and cool-down periods.

User Scenarios. Having a realistic yet precise simulating scenario is a vital part of energy measurement. We designed two scenarios per pattern, the scenario of pattern and antipattern. To minimize external influences, we removed additional details, *e.g.*, in 'DRD', API requests contain only a header. Similarly, in 'OOWN' we have selected images as the content to be loaded dynamically. As 'OOWN' can be applied to other components, *e.g.*, database, camera, and thus can have a different energy impact in those scenarios. On the Web, there is a constant exchange of network information. PowerLog does not measure network traffic; therefore, we have focused on measuring the more processor-dependent patterns.

6 RELATED WORK

Energy Patterns. While previous research in SE has mostly focused on traditional software design issues, such as code smells, or design smells [5, 38], attention has shifted recently towards energy-related concerns in the source code, leading to identifying energy patterns in various domains. For example, Albonico *et al.* analyzed energy practices in robotics software [2], finding ten recurring antipatterns and proposing 14 recommendations to address them. Similarly, Shanbhag *et al.* identified eight energy patterns for deep learning applications [35]. Although many of their patterns are focused on large language models, patterns such as avoiding unnecessary data referencing is a more generalized recommended practice and similar to ours as 'Avoid Extraneous Work', 'Open Only When Necessary', 'Avoid Extraneous Graphics and Animations'. Cruz *et al.* identified 22 energy patterns for Mobile-based applications. We mapped their energy patterns to the Web domain, checked them with web developers, and asked developers for guidelines to identify them in source code.

Developers' insights on Energy Patterns. Despite growing interest in the energy aspect, assessing software energy consumption is rare in the industry. Previous surveys have shown that developers rarely consider energy efficiency in developing software [28, 35]. Manotas et al. [24] extended the survey to a broad domain of developers from various organizations, such as Microsoft, Google, and IBM. They found that developers care about energy, but they lack the required information and infrastructure to develop software. We confirmed the past research in this regards. We collected their understanding and concerns about energy patterns. We inquired for the guidelines to locate these patterns in code. With the guidelines, we could find eleven patterns and nine antipatterns in their source code.

Measurement of Software Energy Consumption. Researchers have analyzed various software or parts of the software to identify energy-hungry practices. They measured energy consumption of software in various context, such as code execution [7, 30, 32], design patterns [23], third party libraries [33], dataframe processing libraries [34], programming languages [29], Java Applications [40], embedded software [39], Mobile applications [20]. As a result, they identified various coding practices or antipatterns that can cause energy inefficiencies.

Given the increasing usage of Web and ecological cost, its energy efficiency is crucial. Some organizations have recommended energy-saving practices for Web applications, like optimizing image formats (AVIF, WebP) to reduce file transfer over browsers. Greenspector [27] measured the energy impact of these image formats and resolutions for various browsers. Singh *et al.* explored the energy cost of Java APIs on servers and found that developers can choose energy-efficient APIs to reduce the energy cost [36]. Inspired by such studies, we measured the impact of two patterns and their antipatterns in two browsers and found that patterns do not always guarantee less energy consumption and browsers and many other factors play a role.

7 CONCLUSION

Given the increasing usage of web applications, we aimed to define Web energy patterns to reduce the energy footprint of these applications. We investigate the porting of existing Mobile energy patterns to the Web domain and found that 20 patterns could be ported. We interviewed developers from different companies to challenge these ported energy patterns and to investigate their awareness and concerns for energy topics. While our interviewees were not familiar with energy issues, most expressed concerns for antipatterns, especially with functional antipatterns. Together with their help, we collected guidelines to identify the ported energy patterns in source code. Based on the guidelines, we could find nine patterns and eleven antipatterns in the company's source code. Finally, we provided evidence that, although these patterns aim to save energy, they do not always succeed. We measured the impact of two patterns, 'OOWN' and 'DRD' and found that the 'OOWN' pattern consumes more energy compared to its antipattern, while we found no evidence for the 'DRD' pattern compared to its antipattern.

ACKNOWLEDGMENTS

REFERENCES

[1] Marwen Abbes, Foutse Khomh, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2011. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In *2011 15Th european conference on software maintenance and reengineering*. IEEE, 181–190.

[2] Michel Albonico, Ivano Malavolta, Gustavo Pinto, Emitza Guzman, Katerina Chinnappan, and Patricia Lago. 2021. Mining energy-related practices in robotics software. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 483–494.

[3] Anders SG Andrae and Tomas Edler. 2015. On global electricity usage of communication technology: trends to 2030. *Challenges* 6, 1 (2015), 117–157.

[4] Earl T Barr, Christian Bird, Peter C Rigby, Abram Hindle, Daniel M German, and Premkumar Devanbu. 2012. Cohesive and isolated development with branches. In *Fundamental Approaches to Software Engineering: 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24-April 1, 2012. Proceedings 15*. Springer, 316–331.

[5] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *2012 28th IEEE international conference on software maintenance (ICSM)*. IEEE, 56–65.

[6] Bobby R Bruce, Justyna Petke, and Mark Harman. 2015. Reducing energy consumption using genetic improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 1327–1334.

[7] Tiago Carção. 2014. Measuring and visualizing energy consumption within software code. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 181–182.

[8] AVIF contributors. 2023. *AVIF Image format*. Retrieved on Oct 2023 from https://avif.io/

[9] MDN contributors. 2023. *Mozilla Page Visibility API*. Retrieved on Oct 2023 from https://developer.mozilla.org/en-US/docs/Web/API/Page_Visibility_API

[10] Luis Cruz and Rui Abreu. 2019. Catalog of energy patterns for mobile applications. *Empirical Software Engineering* 24 (2019), 2209–2235.

[11] Luis Cruz, Rui Abreu, John Grundy, Li Li, and Xin Xia. 2019. Do Energy-Oriented Changes Hinder Maintainability?. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 29–40. https://doi.org/10.1109/ICSME.2019.00013

[12] Jim Daly, Peter Hecke, Divya Kamath, and Amol Deore. 2023. *Microsoft web API for Batch Processing*. Retrieved on Oct 2023 from https://learn.microsoft.com/en-us/power-apps/developer/data-platform/webapi/execute-batch-operations-using-web-api

[13] BooHoo Developers. 2022. *BooHoo*. Retrieved on Oct 2023 from https://www.boohoo.com/page/sustainability-guide.html

[14] Google Developers. 2023. *WebP Image format*. Retrieved on Oct 2023 from https://developers.google.com/speed/webp

[15] Arlene Fink. 2003. *The survey handbook*. sage.

[16] Adrian Furnham. 1986. Response bias, social desirability and dissimulation. *Personality and individual differences* 7, 3 (1986), 385–400.

[17] Stefanos Georgiou, Stamatia Rizou, and Diomidis Spinellis. 2019. Software development lifecycle for energy efficiency: techniques and tools. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–33.

[18] Taher Ahmed Ghaleb. 2019. Software energy measurement at different levels of granularity. In *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE, 1–6.

[19] Jonathan G Koomey et al. 2007. Estimating total power consumption by servers in the US and the world.

[20] Young-Woo Kwon and Eli Tilevich. 2013. Reducing the energy consumption of mobile applications behind the scenes. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 170–179.

[21] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2013. Where has my battery gone? Finding sensor related energy black holes in smartphone applications. In *2013 IEEE international conference on pervasive Computing and Communications (PerCom)*. IEEE, 2–10.

[22] Shuai Ma, Modi Jiang, Peng Tao, Chengyi Song, Jianbo Wu, Jun Wang, Tao Deng, and Wen Shang. 2018. Temperature effect and thermal impact in lithium-ion batteries: A review. *Progress in Natural Science: Materials International* 28, 6 (2018), 653–666.

[23] Sepideh Maleki, Cuijiao Fu, Arun Banotra, and Ziliang Zong. 2017. Understanding the impact of object oriented programming and design patterns on energy efficiency. In *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. IEEE, 1–6.

[24] 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*. 237–248.

[25] Timothy McKay and Patrick Konsor. 2019. *Intel Power Gadget*. Retrieved on Oct 2023 from https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html

[26] Jan Monschke. 2019. *Garbage collection in source code*. Retrieved on Oct 2023 from https://developers.soundcloud.com/blog/garbage-collection-in-redux-applications

[27] Philippot Olivier. 2022. *Greenspector*. Retrieved on Oct 2023 from https://greenspector.com/en/which-image-format-to-choose-to-reduce-its-energy-consumption-and-its-environmental-impact/

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

[29] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy efficiency across programming languages: how do energy, time, and memory relate?. In *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*. 256–267.

[30] Gustavo Pinto, Fernando Castor, and Yu David Liu. 2014. Understanding energy behaviors of thread management constructs. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. 345–360.

[31] Pooja Rani, Jonas Zellweger, Veronika Kousadianos, Luis Cruz, Timo Kehrer, and Alberto Bacchelli. 2024. *Replication Package for 'Energy Patterns for Web: An Exploratory Study'*. Retrieved on Jan 2024 from https://doi.org/10.5281/zenodo.8404487

[32] Haris Ribic and Yu David Liu. 2014. Energy-efficient work-stealing language runtimes. *ACM SIGARCH Computer Architecture News* 42, 1 (2014), 513–528.

[33] Andreas Schuler and Gabriele Anderst-Kotsis. 2020. Characterizing energy consumption of third-party api libraries using api utilization profiles. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[34] Shriram Shanbhag and Sridhar Chimalakonda. 2022. On the Energy Consumption of Different Dataframe Processing Libraries–An Exploratory Study. *arXiv preprint arXiv:2209.05258* (2022).

[35] Shriram Shanbhag, Sridhar Chimalakonda, Vibhu Saujanya Sharma, and Vikrant Kaulgud. 2022. Towards a Catalog of Energy Patterns in Deep Learning Development. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering* (Gothenburg, Sweden) *(EASE '22)*. Association for Computing Machinery, New York, NY, USA, 150–159. https://doi.org/10.1145/3530019.3530035

[36] Jasmeet Singh, Kshirasagar Naik, and Veluppillai Mahinthan. 2015. Impact of developer choices on energy consumption of software on servers. *Procedia Computer Science* 62 (2015), 385–394.

[37] Anselm Strauss and Juliet Corbin. 1998. Basics of qualitative research techniques. (1998).

[38] Davide Taibi, Andrea Janes, and Valentina Lenarduzzi. 2017. How developers perceive smells in source code: A replicated study. *Information and Software Technology* 92 (2017), 223–235.

[39] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. 1994. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2, 4 (1994), 437–445.

[40] Narayanan Vijaykrishnan, M Kandemir, Soontae Kim, S Tomar, Anand Sivasubramaniam, and Mary Jane Irwin. 2001. Energy behavior of Java applications from the memory perspective. In *Java (TM) Virtual Machine Research and Technology Symposium (JVM 01)*.

[41] Philip Walton. 2022. *Energy Saver Mode in Chrome Browser*. Retrieved Aug 2023 from https://developer.chrome.com/blog/memory-and-energy-saver-mode/

[42] Aiko Yamashita and Leon Moonen. 2013. Do developers care about code smells? An exploratory survey. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 242–251. https://doi.org/10.1109/WCRE.2013.6671299