Master thesis

Human-Computer Interaction

Utrecht University

Sharing science,
shaping tomorrow

Empowering embedded systems developers to reduce the energy consumption of their software through visualization: uncovering their information needs

---

Author:
Mark Rietvelt
Utrecht University
mail@markrietvelt.nl

Daily supervisor:
dr. Bernard van Gastel
Radboud University
bernard.vangastel@ru.nl

Internal supervisor:
dr. Hieke Keuning
Utrecht University
h.w.keuning@uu.nl

Second examiner:
dr. Evanthia Dimara
Utrecht University
e.dimara@uu.nl

July 9, 2024

Abstract

Embedded systems software can be optimized for energy consumption to meet strict energy requirements and reduce greenhouse gas emissions. To support this process, we have investigated which information embedded systems software developers need when optimizing for energy consumption. For this purpose, we have used an observational method, a semi-structured interview, and a questionnaire. The existing practices that we identified can help embedded systems software developers in optimizing for energy consumption and uncover ways in which hardware vendors can support this process. Furthermore, our findings about how developers want energy measures to be collected and visualized can aid researchers and tool makers in creating tool support.

1 Introduction

In the past decade, the energy consumption of software has become a concern [1]. Whereas before, improvements in energy consumption were only expected to come from hardware, recently, the responsibility of software itself has been recognized.

While the use of software is growing, there are important reasons to restrain its energy usage. First of all, some software runs on devices that have a battery with limited capacity, such as mobile devices or some embedded systems [2]. Secondly, the production of electricity that is required to run software produces greenhouse gas emissions, which should be contained to honor covenants such as the Paris Agreement to avoid catastrophic effects of global warming [3]. In scientific literature, different approaches have been chosen to limit the energy consumption of software. Approaches target different phases of the development life cycle [2], [4] and vary in the extent to which they take a hardware or software perspective [1].

Many tool prototypes proposed by researchers try to automatically optimize the energy efficiency of software by changing for example the source code itself or its configurations. In their systematic literature review, Balanza-Martinez et al. [5] refer to these approaches as *optimization*. Another popular research avenue instead aims to keep developers in the loop by allowing them to monitor the energy consumption of their code, which is aptly called *monitoring* [5]. Even though past studies have shown gaps in the knowledge of developers of software energy consumption [6], they also discovered that software developers are eager to learn about it [7], [8].

Unfortunately, tools created for energy consumption *monitoring* have been lacking [2]. They have been criticized for being too course-grained because they do not relate energy consumption to specific enough program abstractions [8], and requiring in-depth knowledge [2], [6]. Furthermore, software developers have mentioned usability as an important requirement for these tools and have proposed the use of visualizations to achieve this [9]. Some techniques address this constructive criticism by monitoring energy consumption on the source code level and using visualizations to show their results [10]–[13]. Each of these visualizations, however, makes different assumptions about which information should be shown to developers. Some of these choices have been motivated by the authors. For example, Klinik et al. [13] estimate and show the power draw of external devices rather than of the central processing unit. They motivate this decision by stating that the energy consumption of the central processing unit is sometimes negligible in control software for embedded systems. However, it is yet unexplored whether choices like these really serve the information needs of developers when they optimize the energy efficiency of their code.

In this thesis, we have explored which information should be shown to developers of embedded systems software to aid them in their optimization process. Thus our research question is: "What do embedded systems software developers need to see to optimize the energy consumption of their software?". We focus on the domain of embedded systems specifically, because we hypothesize that information needs of developers can differ between application domains. For example, as Klinik et al. argued, embedded systems developers might require energy consumption information about external devices rather than the central processing unit. Furthermore, the domain of embedded systems often deals with power constraints [14] and guidelines suggest that sophisticated practices for optimizing energy consumption already exist and are worthwhile to study [15]. Drawing from design study methodology [16], we explore our research question by learning about the current practices, challenges, needs, and reflections of embedded systems performance experts. To this end, we use three different methods: an observational study, a semi-structured interview, and a questionnaire.

In section 2 we provide more background information. Next, in section 3, we explain the

methodology that we have used to address our research question. After that, we describe our results in section 4. In section 5, we discuss the limitations of our study, relate our findings to earlier work, and provide recommendations for future work. Lastly, in section 6 we draw our final conclusion.

2 Background

In this section we first discuss software energy consumption, the perspectives of developers on it, and its relation to embedded systems. Next, we give an overview of monitoring techniques that relate energy consumption back to source code and indicate their differences. Last, we discuss software performance visualizations, place energy consumption visualizations in this context, and discuss what information they visualize.

2.1 Software energy consumption

Software energy consumption is a tricky term. Though software certainly is responsible, it is in reality the hardware that it controls that consumes energy. This is clear when the software controls an actuator like an LED light or a temperature sensor for example. However, also when performing simple arithmetic or saving or retrieving values from memory, it is hardware (e.g. the CPU or memory card) that consumes energy. Furthermore, as Georgiou et al. [4] aptly note in their literature review, hardware does in fact not consume energy but rather converts electrical energy to heat.

Software energy consumption is closely related to software performance metrics such as execution time. This is not surprising, given that energy consumption equals power draw multiplied by time. Developers have been relying on this relationship to reduce energy consumption. It should be noted however, that research has shown that reducing metrics such as execution time does not always result in reduced energy consumption [2]. For example, optimization for other metrics might result in reduced execution time, but increased power draw.

The energy efficiency of software has only recently been recognized as a concern for software developers [2]. Hence, it is not yet well-defined as a software quality. In this thesis, we will classify software consumption as a performance metric. Thus, if we mention performance metrics, we refer not only to execution time and other established metrics but to energy consumption as well.

2.2 Software developer perspective

There has been earlier work aiming to understand the perspective of software developers on software energy consumption. Pinto et al. [6] investigated the questions that developers ask about software energy efficiency and the answers that they receive. Furthermore, Pang et al. [7] investigated mobile and desktop developers' awareness of and knowledge about software energy consumption. Pinto and Castor [2] had a similar objective, but focused only on mobile software developers. Furthermore, Manotas et al. [8] performed a larger study on a broader audience (including embedded systems developers) investigating existing practices and perspectives of practitioners and taking into account the full development life cycle. Lastly, Ournani et al. [9] performed a smaller study focused specifically on developer needs. Ournani et al. not only investigated the awareness and knowledge of software developers but also identified the hurdles and constraints that developers encounter and the requirements they have for tooling.

Collectively, these studies diagnosed limited knowledge of developers about software energy consumption and ways to reduce it. They found that developers have misconceptions about the topic and that they find it challenging. Fortunately, the work of Pang et al. [7] and Manotas et al. [8] identified that developers do want to learn about it. All the studies mention monitoring tools as a limiting factor. Pinto et al. [6] identified a lack of tools, Pang et al. [7] diagnosed difficulties in measuring and optimizing energy consumption even when tools were available and Pinto and Castor [2] mention that most monitoring tools are too course-grained for developers. Manotas et al. [8] corroborate these findings and mention that monitoring tools should be easy to use. Ournani et al. [9] echo this call for tool usability, identifying integration with existing tool chains as an important requirement and proposing graphical representations to simplify interaction.

In this thesis, we build upon these findings. We research which information monitoring tool visualizations should show, such that they can better suit the needs of software developers. Specif-

ically, we focus on the application domain of embedded systems which sometimes deal with heavy power constraints [14]. In doing so we follow the example of these previous studies by involving embedded systems software developers to learn about their existing practices, challenges, and needs.

2.3 Embedded systems

An embedded system is a computing system which is designed for specific control functions and is embedded as part of the complete device which may include hardware and mechanical parts [14]. Using sensors and actuators it interfaces with the physical world. As stated previously, embedded systems sometimes have to deal with heavy power constraints. For example, they might be powered using a small battery and are required to function on a single battery charge for multiple years. Fortunately, vendors have been creating microprocessors that have a low power consumption and have many options to tweak it [17]. For example, microcontrollers have low power modes in which their energy consumption is reduced. There are also guidelines for the development of low-power applications which are communicated in formal education, workshops, blog posts, and books. For example, it is good practice to turn off peripherals if they are not used [15], [17]. Furthermore, one could change the clock frequency, select a low-power microcontroller, and use an event-driven architecture to save energy. Researchers have been looking for and continue to develop new ways to optimize embedded systems for energy efficiency [14], [18]. However, except for the study of Manotas et al. [8] (whose sample included embedded systems practitioners), to date we are not aware of any peer-reviewed work that has investigated how embedded systems software is optimized for energy consumption in practice.

2.4 Source code monitoring techniques

The most direct way to monitor the energy consumption of software is to measure its energy use during execution. Measurements can be over time (energy consumption) but also for a specific moment in time (power draw). In this thesis, we will refer to both measures as "energy measures". Historically, measurement required physical power meters [19], but recently hardware manufacturers have started to supply APIs for energy measurement [20] (e.g. RAPL from Intel for PCs and servers and Trepn from Qualcomm for mobile devices). Using these instruments, researchers have attempted to relate energy consumption to source code [10]–[12], [21]. By taking inspiration from Spectrum-based Fault Localization (SFL) techniques [22] they have used information about which program abstractions are executed by which test cases (also called program spectra or code coverage) to correlate energy consumption measurements to these program abstractions.

Instead of (only) using measurements, other monitoring techniques rely on energy models to make estimations [23]. These models capture the energy cost of software constructs given specific hardware [23] and are often calibrated using actual measurements [2]

Some of these estimation-based monitoring techniques use static analysis. They analyze the code without executing it to provide energy consumption estimations. Others use dynamic analysis and execute the code to gather run-time measurements. The execution statistics that they gather are then provided to the energy model to obtain estimations. There are also monitoring techniques that combine both static and dynamic analysis [2].

All these techniques assign a value of energy consumption or power draw to program abstractions. They differ, however, in which abstractions they target, what selection of executions they include, and whether they aggregate over these executions. In section 2.5.2 we will give examples of different techniques and the aspects in which they differ.

2.5 Visualizations

Visualizations have been used successfully and extensively to aid software developers in their activities. The field of software visualization (SoftVis) has even been recognized as a large subfield of information visualization (InfoVis). One of the software development activities for which visualizations have been developed is performance optimization. In this section, we will first discuss various categorizations of performance visualizations. Next, we will discuss four visualizations that have been created specifically to aid in the optimization of energy consumption. We will discuss what information these energy visualizations show and we will classify them using the earlier-mentioned performance visualization categorizations.

2.5.1 Performance

In their state-of-the-art report, Isaacs et al. [24] give an overview of performance visualizations published between 2004 and 2014. Though none of the visualizations that they discuss visualize energy consumption, they define performance optimization as relating to both improving execution time and energy consumption.

In their report, the authors identify three subgoals of performance optimization for which visualizations have been created: (1) Global comprehension, (2) Problem detection, and (3) Diagnosis and attribution. According to their classification, global comprehension involves understanding the regular performance behavior of the software. Furthermore, the authors describe problem detection as the identification of erroneous performance behavior. Lastly, with diagnosis and attribution, they refer to the linking of the erroneous performance behavior to its causes. They note that diagnosis and comprehension might already be captured within problem detection and that visualizations can aid in multiple of these subgoals.

Isaacs et al. [24] also categorize visualization techniques based on which context(s) they represent: Hardware, Software, Tasks, or Application. Relevant to the energy consumption visualizations, which we will introduce shortly, for Software visualization techniques the authors distinguish between: (A) Serial trace, (B) Call graph, and (C) Code and Code Structure visualizations.

2.5.2 Energy

As far as we know, to date, there are only four visualizations related to energy measures that have been published in peer-reviewed work. Images of these visualizations can be found in Figure 1. Here we describe these four visualizations, identify which information they show, and relate them to the classification of Isaacs et al. [24].

Figure 1a shows a visualization by Li et al. [10]. The authors measure energy consumption averaged over a select set of executions. They run test cases many times to reliably measure their energy consumption. They relate this energy consumption back to source code lines. They show the relative difference in energy consumption by highlighting lines of code. They give code lines a color on a gradient from blue to red. They highlight the line of code that consumed the least amount of energy in blue, the line of code that consumed the most amount of energy in red, and the others in a shade of purple.

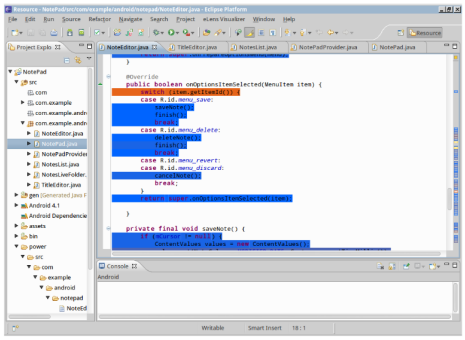
Figure 1b shows a visualization by Verdecchia et al. [11]. They also measure energy consumption averaged over a select set of test case executions. They, however, relate this energy consumption back to source code functions, branches as well as lines. They also show the relative difference in energy consumption but use a heatmap for this purpose. On this heatmap, they display the names of the program abstractions in alphabetical order. They color their cells on a gradient from white to red. They color the function, branch, or line that consumes the least amount of energy white, the one that consumes the most amount of energy red, and the others a shade in between.

Figure 1c shows a visualization by Couto et al. [12]. They also measure energy consumption averaged over a select set of test case executions. They relate this energy consumption back to methods and propagate this to classes, packages, and projects. They show their involvement in large energy usage in a sunburst diagram. Program abstractions with low involvement are colored in green, those with high involvement in red, and those in between in yellow. Unlike Verdecchia et al., they show the results for all considered program abstractions in one visualization, using their hierarchical structure.

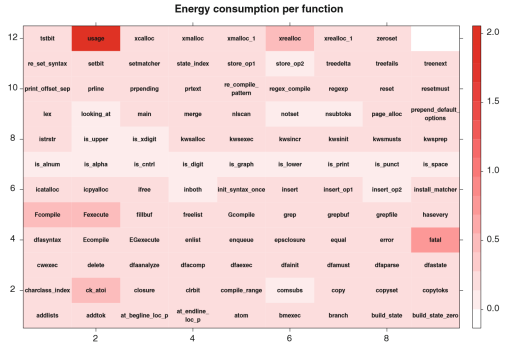
Figure 1d shows a visualization by Klinik et al. [13]. They deviate from the previously mentioned visualizations by estimating power draw for all possible executions over the code. They relate this power draw back to code statements by showing the cumulative power draw as lines next to the source code. Changes in cumulative power draw are represented by horizontal displacement of the lines. All possible executions with different power draw are represented by unique lines of different colors.

As their introductions reveal, these visualizations make different assumptions about which information should be shown to developers. An exhaustive list of these assumptions and which visualizations make them can be found in Appendix A.

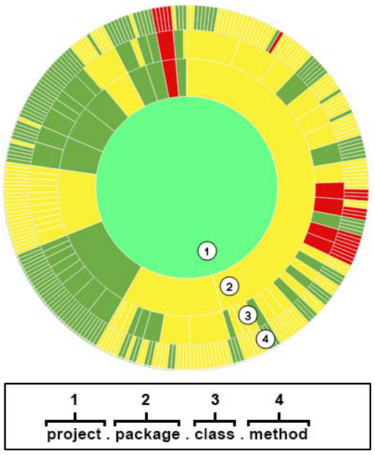
Furthermore, these four energy visualizations follow the classifications of Isaacs et al [24]. Li et al. [10] mention that their work has the potential to "help developers understand the energy related behavior of their applications.". Furthermore, Couto et al. [12], Verdecchia et al. [11] and



(a) Coloring of source code lines by Li et al. [10] showing their relative energy consumption. Reproduced without permission.

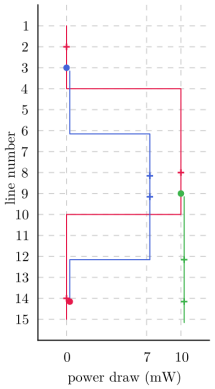


(b) Heatmap by Verdecchia et al. [11] showing the relative energy consumption of methods. Reproduced without permission.



(c) Sunburst diagram by Couto et al. [12] showing the involvement of methods, classes, packages, and a project in high energy consumption. Reproduced without permission.

```
1  int main() {
2    x = SENS.readTemp();
3    if( x ≤ 10 ) {
4      LED1.switchOn();
5    } else {
6      LED2.switchOn();
7    }
8    sleep(100);
9    if( x < 10 ) {
10     LED1.switchOff();
11   } else {
12     LED2.switchOff();
13   }
14   return 0;
15 }
```



(d) Skylines by Klinik et al. [13] showing the (changes in) cumulative power draw for all possible executions of the source code. Reproduced without permission.

Figure 1: All software energy consumption visualizations published to date in peer-reviewed literature, as far as we are aware.

Klinik et al. [13] all mention that they envision their work to be used specifically to identify energy consumption hotspots. Thus, all these visualizations attempt to serve the global comprehension subgoal of Isaacs et al.

Furthermore, all of these visualizations can be identified as code and code structure software visualizations. The sunburst diagram from Couto et al. [12] is a code structure visualization. The source code line coloring of Li et al. [10] and the heatmap of Verdecchia et al. [11] are code visualization because they visualize the code together with the energy consumption information by coloring the background of the relevant program abstractions. Lastly, the skylines of Klinik et al. [13] is a code visualization that shows power draw in close conjunction with the code.

The fact that these energy consumption visualizations do not cover all performance subgoals and contexts that have been identified by Isaacs et al. [24] might indicate opportunities for future visualizations. By investigating the existing practices of experts, we can learn whether experts indeed pursue these different subgoals during their optimization process. If so, we can learn for each of these subgoals which information sources they already consult, which challenges they face, and which (information) needs they have. Energy visualizations can then target these specific subgoals using appropriate contexts to suit experts' needs.

3 Methodology

We have observed and questioned developers who have optimized embedded systems software for energy consumption, to learn about their current practices, their challenges, and their needs, and to what extent existing energy monitoring techniques and visualizations suit their needs. In this section, we will first introduce the design study methodology on which our methodology is based. Then, we will describe how we recruited participants. Next, we will explain the three methods we employed: reenactment, semi-structured interview, and questionnaire. After that, we will describe the data collection and analysis for these methods. Lastly, we will discuss how we preserved ethical integrity and respected the privacy of participants.

3.1 Design study methodology

In this thesis, we explore the question *"Which information do embedded systems software developers need to optimize the energy efficiency of their software?"*. By doing so, we took a first step towards the creation of problem-driven energy consumption visualizations. Specifically, we engaged in the discover stage of the nine-stage design study methodology framework prescribed by Sedlmair et al. [16]. This framework is specifically made for visualization design studies which Sedlmair et al. define as "a project in which visualization researchers analyze a specific real-world problem faced by domain experts, design a visualization system that supports solving this problem, validate the design, and reflect about lessons learned in order to refine visualization design guidelines." In our case, the domain experts are embedded systems developers, who face the real-world problem of optimizing the energy efficiency of their software. The discover stage of the nine-stage framework of Sedlmair et al. focuses on the analysis of the real-world problem. In this stage, a researcher investigates the existing processes, challenges, and needs of domain experts to characterize the problem for which they intend to create a visualization. Therefore, we aimed to answer the following subquestions (SQ):

SQ1 Which subgoals do embedded systems software developers pursue when optimizing the energy consumption of their software?

SQ2 Which information do embedded systems software developers already use to optimize the energy consumption of their software?

SQ3 Which challenges do embedded systems software developers face when optimizing the energy consumption of their software?

SQ4 Which needs do embedded systems software developers have when optimizing the energy consumption of their software?

The discover stage requires close involvement with domain experts. Before we elaborate on how we involved the domain experts, we will describe who they are and how we approached them.

3.2 Participants

We have recruited people who have at least once optimized embedded systems software for energy consumption. We required this experience of participants so we could learn about their existing practices. We did not impose a stricter requirement as novice experiences and views would also be valuable to our research and a stricter requirement would have prevented us from reaching a large enough sample.

In total, we recruited 9 participants as their participation was quite time intensive for them as well as for us to analyze. We hoped to be able to reach full saturation within 5 to 10 participants. Indeed fewer and fewer novel points were shared by participants, but we did not reach full saturation. However, after 9 participants we did not deem the novel information important enough anymore to warrant future time from us and participants. Though the sample size of 9 participants did not allow us to statistically assess the quantitative data gathered, it did provide us with sufficient examples to identify data trends.

For recruitment we used a combination of self-selection, convenience, and purposive sampling. We used self-selection sampling by creating a LinkedIn post that explained the study and mentioned the recruitment criterion that allowed participants to contact us if they wanted to participate. We shared this post with personal contacts (convenience sampling), but only those whom we expected

to have contacts that would meet our recruitment criterion (purposive sampling). From several of our contacts, we received a list of companies that they deemed likely to have employees that meet our recruitment criterion (purposive sampling). We then called and/or emailed these companies to recruit participants. All participants developed embedded systems software professionally, each working at a different company in The Netherlands.

3.3 Involvement

We asked participants to partake in a single session of one to one and a half hours. These sessions were held in person at the company of the participant unless they preferred to conduct them using video calling instead. Five out of the nine participants opted for a video call and the other four were visited in person. The research was conducted in the mother tongue of the participants, which was Dutch for all participants. The research was executed by a single researcher to whom we will from now on refer to as "the researcher". During the sessions, the researcher followed the protocol which can be found in the replication package[1]. This repository also contains all other materials used during these sessions. We will now introduce the three different methods that we employed during these sessions: a reenactment, a semi-structured interview, and a questionnaire. See Table 1 for the duration of each method.

	Reenactment	Interview	Questionnaire	Session
Min	22	8	18	62
Average	32	11	31	73
Max	38	15	51	93

Table 1: Shortest, average, and longest duration in minutes of the reenactment, semi-structured interview, and questionnaire (as well as the complete session).

3.3.1 Reenactment

To learn about the current practices of the participants, the researcher asked participants to lead him through a recent process in which they optimized embedded systems software for energy consumption. We refer to this method as reenactment.

The researcher steered participants to focus on the part of the process where they optimized software for energy consumption whilst writing or refining it. However, the researcher also left them room to share relevant parts of their process prior to this point. Furthermore, if they had optimized the energy consumption of multiple products the researcher asked them to focus on the one that they deemed most relevant to share.

The optimization processes that participants explained took between a couple of days up to a couple of years, often running in parallel with feature development. We indeed anticipated participants to be involved in the optimization process infrequently and for extended and variable length. Therefore, we opted for a retrospective inspection. Even though observing the full process in real-time would probably have resulted in a more detailed understanding, the large time investment and logistics involved in attending the process from the start made it unfeasible for this thesis.

We invited participants to show source code, tools, and information that they used in the optimization process while they talked the researcher through it. This bears great similarity to the retrospective think-aloud methodology which sees great use in Human-Computer Interaction studies [25], with the exception that there is no recording of the optimization process but instead the reenactment of the participant of it. Whenever it was unclear why the participant performed a certain action, the researcher interrupted and asked for clarification. This aspect is borrowed from contextual inquiry [25], which has proven its usefulness for learning about the existing practices of domain experts in past design studies [16]. We have coined this method reenactment as it asks the participant to relive and retell their experience in a realistic manner. Though it can be classified as a Cognitive Task Analysis [26] or knowledge elicitation technique, we are not aware of the previous use of this method.

In practice, during the reenactment, participants showed less source code, tools, and other information and discussed their optimization process more abstractly than we expected. This is

[1] `https://github.com/MarkRietvelt/energy-optimization`

probably due to the fact that the optimization processes of the participants were longer than we anticipated. We had anticipated them to be a couple of hours to a couple of days rather than a couple of days up to a couple of years as they turned out to be. Likely, this made participants look at their optimization from a distance and let go of some details.

Observing the reenactment of the optimization process and asking for clarification during it allowed us to understand the current practices in optimizing the performance of embedded systems software. Specifically, it helped to learn whether participants perform a clear set of subgoals (SQ1) and which information participants consult (SQ2).

3.3.2 Semi-structured interview

The reenactment was directly followed by a semi-structured interview. The exact questions asked during the interview can be found in the protocol in the replication package. In the interview, the researcher first asked whether the participant had optimized embedded systems software for energy consumption more often. If they had, the researcher would ask whether these optimization processes differed from the one they reenacted and if so how. Furthermore, the researcher would then phrase the remaining three questions to include these other optimization processes to get a more comprehensive understanding.

In these remaining questions, the researcher respectively asked about their subgoals (SQ1), challenges (SQ3), and information needs (main research question) whilst optimizing embedded systems software for energy consumption. Although we could have learned about possible subgoals during the reenactment, we chose to ask about them explicitly in the semi-structured interview as well for two reasons. First, participants might be able to identify subgoals in their optimization process only after fully reenacting it. Second, first asking participants about different optimization processes that they did allowed them to identify subgoals during these other processes and those shared between them.

To ensure a high recall rate of subgoals, challenges and information needs the researcher often prompted participants after they mentioned one subgoal, challenge or information need whether they could recall more. For challenges we specifically prepared a prompt asking about any long or difficult parts of the optimization process. For all questions the researcher sometimes listed the subgoals, challenges or information needs that the participant already expressed during the interview as a prompt and to clarify the question. For subgoals, the researcher sometimes used the synonym “interim goals” to clarify the intended meaning.

We only asked about challenges in the semi-structured interview, which followed the reenactment, to avoid the pitfall of focusing only on problematic parts of the workflow (mentioned as pitfall 15 in [16]). Similarly, we only asked about information needs after the reenactment, to avoid the pitfall of focussing on possible solutions too soon (mentioned as pitfall 17 in [16]).

3.3.3 Questionnaire

Even without a thorough understanding of the existing practices, challenges, and needs of embedded systems developers when optimizing for energy consumption, earlier work has created techniques to help them in this process. These techniques make assumptions about what information these developers should see. In an online questionnaire, we aimed to learn whether participants agree with these assumptions and in which way they believe that energy measures could help them to optimize embedded systems software. Participants were asked to choose between different options, indicate their agreement to statements on a 4-point Likert scale, and asked to elaborate upon their answers in open questions. An even-point Likert scale was chosen to force participants to select a non-neutral response. Multiple choice questions were accompanied by open questions to allow participants to add nuance and explain their decision-making process. An exported version of the online questionnaire can be found in the replication package.

Kind of energy measures Several questions asked participants about what kinds of energy measures should be collected. Participants were asked whether power draw or energy consumption measures should collected, if measures should be collected for a single, several, or multiple execution(s), and if they should be collected for the CPU or external hardware components. These questions originated from assumptions made in the four energy visualizations described in section 2.5.2. The assumptions that these visualizations make can be found in Appendix A. Based on the original ideas of one of the authors of this thesis, participants were also asked whether the software

they wrote was used on multiple different hardware combinations and executed under different usage scenarios and if so whether energy measures should be collected for all of them. Lastly, participants were given the opportunity to list ideas about what other characteristics of energy measures would be useful or counterproductive.

Relation to source code A couple of other questions asked participants about how energy measures should be related to the source code. The first of these questions asked participants to rank program abstractions based on how useful it would be if energy measures are related to them. The next questions asked whether energy measures should be related to multiple program abstractions, and assuming so which program abstractions that should be and how they should be visualized. Again, these questions originated from assumptions made in the four energy visualizations described in section 2.5.2 and these assumptions can be found in Appendix A.

Subgoals and Challenges Participants were also asked if understanding the energy-related behavior of embedded systems and finding energy consumption hotspots are useful steps in the optimization process. These questions were inspired by the goals of the energy visualizations discussed in section 2.5.2 which serve the first subgoal proposed by Isaacs et al. [24]: Global comprehension. Furthermore, participants were also invited to provide additional suggestions for useful subgoals. Lastly, the questionnaire also asked about challenges that energy measures could help overcome.

Research questions Questions about which kind of energy measures should be collected and how they should be related to the source code help towards answering the main research question. Questions about the subgoals and challenges help answer the first and third subquestions (SQ1 and SQ3) respectively.

Procedure Participants were asked to fill in the questionnaire after a 5-minute break following the interview. Participants were instructed to follow their intuition if they were unsure about their answers and explain their answers in the open questions. They were invited to keep this explanation short and to refer to the reenactment and the interview. Furthermore, they were encouraged to ask questions whenever they had any or when something was unclear. The researcher was available to answer these questions and resolve any unclarities directly. Due to circumstances, one participant filled in the survey slightly later and therefore did not have the opportunity to ask questions to the researcher.

The questionnaire was very valuable in testing existing solution ideas but was intentionally preceded by the reenactment and semi-structured interviews to prevent initial conversations from being steered by these ideas (see pitfall 17 in [16]).

3.4 Piloting

All methods were refined in a pilot phase with three pilot participants (Pi1-3). The reenactment and the semi-structured interview were piloted with Pi1 and Pi2. The questionnaire was piloted with Pi2 and Pi3

Pi1 reenacted an optimization process from about 5 years ago and recalled it surprisingly well. Based on this, the criterion that the optimization process had to be performed in the last 2 years was relaxed to needing to be recalled with sufficient detail. Pi2 did not have one concrete optimization process to reenact, which made the reenactment unfocused. This emphasized the criterion of having a concrete optimization process for the reenactment. Based on the pilot with Pi1, the interview questions were also written out in full to ensure consistency.

During the pilot, both Pi2 and Pi3 were asked to verbally explain their reasoning process. This uncovered unintended interpretations of several questions, based on which these questions were rephrased. These verbal explanations also provided useful contributions toward answering the research questions. Therefore, open-ended elaboration questions were made mandatory instead of optional.

3.5 Analysis

Now we will describe which data we collected during the methods, how we processed it, and how we have analyzed the processed data.

3.5.1 Reenactment & Semi-structured interview

An audio recording was made of the reenactment as well as the semi-structured interview which was transcribed automatically but corrected manually using Microsoft Word 365[2]. Furthermore, the optimization process of each participant was summarized. Participants were asked if they would like to receive this summary. Four participants expressed their liking and were sent the summary. These four participants were also given the opportunity to indicate and correct any erroneous information which led to a couple of minor adjustments.

The transcripts were coded manually using NVivo[3]. Prior to the data collection, a set of a priori codes was derived from the study objectives. These codes and their descriptions can be found in Table 2. These codes were intended as top-level codes in the code hierarchy. Future coding was intended to be emergent in the sense that interesting concepts or ideas in the data that were related to these top-level codes could be coded as subcodes. For example, if a participant would express that they designed the hardware in an energy-efficient way this could be coded as "Hardware Design" as a subcode of "Subgoal".

Code	Description
Task	An action performed by a participant to optimize embedded systems software for energy consumption
Subgoal	A partial goal towards the optimization of embedded systems software for energy consumption
Tool	A physical or software tool used in optimizing embedded systems software for energy consumption
Information	Information relevant to optimizing embedded systems software for energy consumption
Visualization	A visual representation of some Information
Challenge	A difficulty a participant faces while optimizing embedded systems software for energy consumption
Need	A need a participant has while optimizing embedded systems software for energy consumption

Table 2: A priori defined codes that match the objectives of the study.

A first part of the coding was assisted by an HCI researcher who was unrelated to this research and had experience in qualitative methods. This dual coding part consisted of three rounds. In each round, both researchers separately coded two 5-minute excerpts from the transcripts of a participant, one 5-minute excerpt from the reenactment and one 5-minute excerpt from the interview. To cover a large variety of topics, the 5-minute excerpts were taken from the beginning of the reenactment and interview in round 1, from the middle in round 2, and from the end in round 3.

For the first round, both researchers were supplied with a codebook containing the a priori defined codes. After each round, the researchers went chronologically through the excerpts, discussing each coded part until reaching consensus. The resulting updated codes and codebook provided the start of the next round. This iterative process for example resulted in the merging of the codes for "Task" and "Subgoal" as it turned out that a specific subgoal could be formulated for every task. The remainder of the coding was performed solely by the researcher involved with the research, starting with the codebook produced after reaching consensus after the last dual coding round.

[2] `https://www.office.com`
[3] `https://lumivero.com/products/nvivo/`

3.6 Ethics and privacy

Next, we will discuss how we preserved ethical integrity and respected the privacy of participants all the way from the recruitment until the finalization of the study.

After participants showed interest in participating in the study, they received an email containing an explanation of the study. The email explicitly explained the use of an audio recording and invited participants to ask any questions they still have by email, phone call or before the start of the study. Attached to this email, participants received an information sheet and consent form which can be found in the replication package. Participants were invited to read these documents and were given the option to send a filled-in consent form back to confirm their participation. If they did not do so before the start of the study, they were asked to read the documents, ask any questions they still had, and fill in the consent form at that moment. If the participant was visited in person, paper versions of these documents were provided and they filled in one additional consent form for their own record.

The information sheet explains the study and states that participation is voluntary and that the participant may withdraw at any time without providing a reason for withdrawal. Furthermore, it explains that participants can ask for the deletion of their data until it is anonymized. Additionally, it explains that the participants will be asked whether they consent to their anonymized data being shared with other researchers for dissemination and future research purposes. Lastly, it informs participants that they can ask any questions they still have via email or at the start of the study. The consent form asks the participants for explicit consent for participation in this research and the use of their data.

The reenactment and the questionnaire were preceded by an explanation which can be found in the replication package. The explanation preceding the reenactment reiterated that participation is voluntary and that the participant could withdraw from the study at any time without reason. Furthermore, after both explanations, participants were asked whether anything was still unclear or whether they had any more questions.

Furthermore, participants were assigned an identifier, which they entered in their questionnaire response. This allowed us to link their questionnaire response to their reenactment and interview without them having to submit personally identifiable information.

This thesis was screened beforehand by the Ethics and Privacy Quickscan and found to be low risk. A submitted version of this document can be found in the replication package.

4 Results

In this section, we will discuss the results from the reenactment, interview, and questionnaire. We will first discuss the sample of participants and then discuss the results pertaining to each sub research question as well as the main research question. Most results are from the reenactments or interviews and we explicitly mention when results are from the questionnaire. We always mention which participant raised a certain point and sometimes use their quotes[4] if these are illustrative or to highlight subtleties in their responses. If a participant is not mentioned it does not imply that they did not experience or agree with the matter. It simply means that the topic was not raised in the reenactment, interview, or open questionnaire responses[5].

4.1 Participants

Overall, participants reported that they sometimes optimize the *energy consumption* of embedded systems software. On a 4-point scale from never to often, P3 and P6 answered 'rarely', P7 answered 'often' and all other participants responded 'sometimes'. Overall the participants optimized the *performance* of embedded systems software a bit more, with five participants (P2,5,6,7,9) responding 'sometimes' and the other four participants answering 'often'. Only P7 showed the opposite trend, which suggests that they do not consider energy consumption as a type of performance.

Participants had varying years of experience in developing software for embedded systems, ranging from three to 26 years[6]. Their job titles and roles mentioned that four participants were

[4] Quotes were originally in Dutch but have been translated to English

[5] For the multiple choice questions we have responses from all participants

[6] For the three participants that reported part-time, school, or non-professional years of experience, these years were fully included.

engineers, four were architects (with one being both) and the titles of eight of them explicitly included the word software or firmware. One participant was an intern, the others either were a lead, senior, or principal or had around 20 years of experience.

Participants also optimized embedded systems software for various applications in various sectors. However, the sample might not be fully representative as all embedded systems of which the participants described their optimization process were relatively complex (e.g. use an OS or have communication functionality). This might be caused by the fact that all participants worked at a company in The Netherlands.

	Energy	Performance	Experience	Job
P1	Sometimes	Often	25	Embedded firmware architect
P2	Sometimes	Sometimes	3	Lead engineer
P3	Rarely	Often	3	Intern / Embedded Software Engineer
P4	Sometimes	Often	26	Senior Software Architect
P5	Sometimes	Sometimes	8	Senior Embedded Software Engineer
P6	Rarely	Sometimes	18	Principal engineer, System and software architect
P7	Often	Sometimes	10	Teamlead embedded firmware
P8	Sometimes	Often	19	Software Architect
P9	Sometimes	Sometimes	11	Embedded tech lead, mostly developing firmware

	Sector	Application
P1	Industry, Automotive, Airfield	Industry, Automotive, Airfield, RFID
P2	Medical, HVAC systems	Medical, Room thermostat
P3	Agriculture and karting	Low-power sensors, Rev limiter
P4	Consumer Electronics	Electrified gear change for bicycles
P5	Building Management/Real Estate	IoT Sensors
P6	Battery powered consumer devices	Door lock, Air quality measuring
P7	Logistics, Heating	Asset tracking
P8	Sensing, industrial and consumer	Mostly measuring equipment
P9	IoT	Battery operated radio devices

Table 3: Table with the experience of participants with developing and optimizing embedded systems software.

To summarize

Our sample consists of a representative group of participants with sufficient to much experience in software development for complex embedded systems for various applications in various sectors. The majority of our participants sometimes optimize for energy consumption.

4.2 Subgoals

We will now discuss the results that pertain to the sub research question: *Which subgoals do embedded systems software developers pursue when optimizing the energy consumption of their software?*

4.2.1 Workflow

About their optimization process, P8 mentioned that "in general, this is a trajectory where we are very systematic. Because, as I mentioned, you don't design low power consumption in afterward, so it really has to be an integral part of the product.". From the reenactments and interviews, we could indeed identify a systematic trajectory for embedded systems software developers who are optimizing for energy consumption using an energy requirement. We have visualized this workflow in Figure 2 and will explain it step by step.

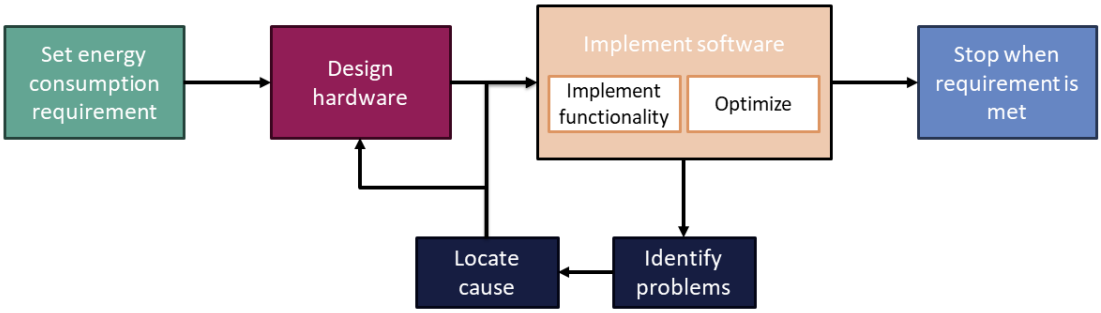


Figure 2: Workflow of embedded systems software developers who are optimizing for energy consumption using an energy requirement.

Energy consumption requirement Except for P1, all participants started their optimization process with a requirement on the energy consumption of their embedded system. Except for the system of P3, all these systems were battery-powered and required to function for a certain amount of time on a single battery charge. The system developed by P3 was instead constrained by a limited power supply. Similarly, P8 mentioned an example of a solar-powered device in which the requirement entailed maximum power consumption. Also, one of the two embedded systems discussed by P6 could recharge its battery using a solar panel to a limited extent. P1 did not have a requirement for energy consumption specifically. However, it did have standards for the amount of disturbance the product can create and the speed with which it reacts, which in turn limits its energy consumption.

It does not appear to be a coincidence that only participants developing an embedded system without plenty or continuous power (e.g. battery-powered systems) have an energy consumption requirement. Statements of P3, P6, P7, P8, and a colleague of P5 all suggest that, currently, only these systems have an energy consumption requirement.

Hardware design With the exception of P1, all participants mentioned the design of the hardware as a first and important step towards meeting their energy consumption requirement. They all indicated that components were selected to ensure low energy consumption and P8 noted in the questionnaire that "usually a lot of preliminary research is done for that". P2 and P6 specifically mentioned that it was checked which sleep modes their components could go into. P4 and P5 remarked that hardware components were chosen that supported the existing low-energy communication protocols BLE and LoRa respectively. P3, P7, and P9 even went beyond sleep modes and designed their hardware in such a way that they could turn off different components by turning off their power supply using so-called power domains. Lastly, P1 and P8 mentioned that specialized hardware could be faster and more energy efficient than software that supports the same functionality. P8 even once worked on a product for which separate chips for communication and calculations were chosen which lowered the energy consumption but increased software complexity. P6 also considered adding a specialized cryptography chip to their PCB to lower energy consumption but decided against it to limit complexity and cost. Similarly, P4 questioned the decision to include an expensive motion sensor. They argued that the energy consumption of the motion sensor diminished the energy it saved by allowing a microcontroller to sleep.

P2, P3, and P5 explained that a use case was calculated to ensure that the hardware they selected could meet their energy consumption requirement. P2 and P6 mentioned using their use duration requirement to calculate how much power their embedded system could consume on average. P6 then created a power budget in which they prescribed for each peripheral how long it may be active.

Software design Except for P1, all participants used an event-driven architecture in which they put components to sleep or turned them off when they were not needed and used interrupts to activate them again. P6 mentioned specifically choosing an OS to support this sleep functionality. P2 used the same OS and mentioned researching which power optimizations it supports. P2 also mentioned that if no such OS were used, they would have to think and decide for themselves when the system would go to sleep.

Implementation P1, P3, and P8 mentioned that they first implemented their software while paying little to no attention to its energy consumption to first get a working implementation. As P2 and P4 were adapting the software of an existing embedded system, they also started their optimization process with a working implementation. To determine whether they needed to optimize this working implementation for energy consumption, P2 measured its initial energy consumption. Once they have completed their implementation, P3 intends to do the same thing. Next, P3 and P8 will apply/applied the optimizations that they conceived one by one. In contrast, P2 isolated different parts of the functionality to measure their energy consumption and optimize it.

Other participants did not postpone energy consumption optimization until after creating a working implementation. P5 explained that they thought in advance about how to smartly implement the desired functionality to limit its energy consumption. P8 also mentioned that their low-power communication protocol has been designed early on. Furthermore, P6 and P9 mentioned that they optimized for energy consumption whilst implementing functionality.

P2, P3, P4, P5, P7, and P8 all mentioned that they measured the energy consumption after applying an optimization. Table 4 shows the optimizations that participants mentioned.

Optimization		Participant(s)
Event-driven architecture	Use sleep modes	P2-9
	Use low power modes	P3,4,7-9
	Turn off component	P3,4,7,9
	Turn off energy supply	P3,7,9
	OS with automatic sleep functionality	P2,6
Change configurations	Lower clock speed	P3
	Configure peripherals	P3,5
	Configure registers	P5
	Other	P7
Sensing	Optimize measurement interval	P5,8
	Lower sampling rate	P8
Data analysis	Analyze data on device	P5
	Compress data early	P8
	Create efficient algorithms	P8
Communication	Tweak communication parameters	P2,4,6,9
	Write custom communication protocol	P8,9
	Buffer data before sending	P5,8
Software engineering optimization	Minimize duplication, reuse	P8
	Swap loops or if statements	P8
	Write out loops	P8

Table 4: Table with optimization which participants used or planned to use in the optimization process that they described or in another optimization process that they mentioned. Some optimizations are left out because they are very specific to the application at hand.

Identify problems P6 expressed that "during software development, you find out that certain things may or may not function or consume more power than they should.". Indeed, many participants raised problems that they identified during their optimization process.

For two participants, the identification of problematically high energy consumption was the reason for their optimization process. P2 found the initial energy consumption of their embedded system to surpass their energy consumption requirement and the company of P4 was asked to change the software as the battery life of the embedded systems of their client was too low and varied greatly. Other problems were identified during development. P3, P5, P6, P7, and P9 all mentioned that they at some point discovered that the actual energy consumption surpassed their expectations. Furthermore, while tweaking parameters, P1 discovered that the hardware was not fast enough. Similar to why P4 was asked to change the software, P7 identified a low battery life only after their product was already out in the field. To catch these issues early, P6 and P7 validate the energy consumption of their system before each software release. Similarly, P1 and P5 mentioned validating the energy consumption after a new hardware iteration.

Locate cause When they discovered problems, many participants tried to locate their cause. P2, P3, P5, P6, and P7 mentioned trying to look for causes of high energy consumption. Furthermore, P7 and P9 also described locating causes of unexpected and undesired behavior. Contrarily, P4 did not attempt to locate the cause of the low battery life but instead changed the software such that the system could only wake up from local electrical signals which eliminated the cause and successfully solved the issue. P5 located causes together with their hardware supplier. In this process, P5 identified where in the code too much energy is consumed. Their hardware supplier then located the cause of the issue in the hardware.

P5, P6, and P7 explained that in this locating process, they isolated components to measure their energy consumption. In the questionnaire, P5 also mentioned this as a subgoal. To do so, P5 and P6 turned off parts in the software. If that did not work, P6 resorted to "soldering a jumper between the power supply and the chip". Furthermore, P7 measured voltage at different places. In contrast, P9 expressed that "they never did a real component by component analysis". They reasoned that hardware components can already be recognized sufficiently well from a power consumption measurement of the whole system over time due to the difference in magnitude and timing of their power consumption. They for example rely on the knowledge that communication consumes most power. A piece of knowledge which P2, P5, P6, P7, and P8 mentioned as well. They acknowledged that this only allows peripherals other than the radio to be recognized when the radio and microcontroller are in sleep mode, but stated that their energy consumption would be relatively insignificant if these components were awake.

P6 mentioned that sometimes an identified problem was only investigated and resolved later. By postponing this, more urgent problems, which for example require a hardware change that involve long lead times, could be located first.

Solve problems Many problems that participants encountered were related to the hardware. See Table 5 for the causes of and solutions to the problems that participants mentioned.

P1, P4, and P5 mentioned that changes in the hardware design were made during the development. P5, P6, P7, and P8 mentioned changes in the wiring of the hardware components. Furthermore, P6 mentioned the replacement of a component that had no sleep functionality due to a mistake in the bill of materials. No mentions were made of components being changed within the development of a product to solely benefit energy consumption. When asked about this, P8 answered that this did happen between product iterations but that "usually hardware choices are not adjusted anymore, just because the lead time is too high" and that changes were therefore too expensive.

Some problems were instead solved in the software. P6 chose to fix an issue through software to save on costs, mentioning that "a solution in software is often cheaper, but often not the most optimal solution". Furthermore, P7 fixed an issue in software as they already had products out in the field. P1 however expressed that "It is always a dialogue between the hardware and the firmware. The firmware alone cannot solve the problem".

Problem	Solution	Participant(s)
Hardware not fast enough	Change hardware design	P1
Sensor too sensitive	Tweak sensor parameters	P4
Wrong resistor value	Change resistor value	P5
Switch working badly	Solution not mentioned	P5
Pin set differently in software than hardware	Change assignment in software	P5,7
Pin set differently in software than hardware	Change hardware wiring	P6
Initialization too energy intensive	Keep component awake	P6
Wrong hardware component ordered	Replace hardware component	P6
Timers prevent OS from going in sleep mode	Change timer type	P6
Undocumented hardware bug	Workaround in software	P6,9
Random behavior due to slow drop in voltage	Keep buck-boost converter on	P7
Random behavior when turned off	Put flash chip in sleep mode	P7
Random behavior when turned off	Put flash chip in low power mode	P9

Table 5: Table with causes of and solutions to problems that participants encountered during the optimization process that they described or another optimization process that they mentioned.

End P2, P4, P8, and P9 mentioned that their optimization processes came to an end when it was determined that no further energy consumption saving was needed. P8 and P9 explicitly mentioned that it was the company that changed their priorities which shifted towards creating new features or products. P2, P5, P7, and P8 all mentioned using a long-term test to evaluate the energy consumption of the embedded system. For this long-term test, P2 measured the energy consumption over a day and extrapolated that measurement to estimate the battery life. P8 mentioned using power measurements and measuring how long the system could function on one battery charge (which was feasible as it was around 6 hours). P5 and P7 mentioned collecting battery level measurements reported by the system through server logs.

To determine that no further energy consumption saving was needed, P4 created a use case. They determined relevant scenarios of the system, measured their energy consumption, and calculated how many times or for how long they could be executed on one battery charge. Then, using estimations of their typical frequencies/durations they estimated the battery lifetime which was found to be sufficient. In the questionnaire, they mentioned determining relevant scenarios as a specific subgoal.

4.2.2 Global comprehension

The questionnaire also specifically asked about subgoals related to the Global comprehension subgoal of Isaacs et al. [24].

Understanding energy-related behavior P8 and P9 somewhat agreed, and all other participants strongly agreed, that understanding the energy-related behavior of embedded systems software is a useful step in the process of optimizing its energy consumption. It is good to note however that the depth to which participants go to understand the energy-related behavior of their system probably differs. As will be mentioned in section 4.6.1, P1 expressed desiring better insight into CPU power usage even though it does not use the majority of the energy consumption in their project. Later in the questionnaire P1 also states that "insights are always good". This contrasts with a comment of P9 that "the effort you put into the measurements need to be able to be justified against the winnings".

Finding energy consumption hotspots All participants strongly agreed (except for P8 who somewhat disagreed) that finding energy consumption hotspots is a useful step. P2 and P9 mentioned that optimizing "the most power hungry parts" i.e. "hotspots" gains the most energy savings. P8, however, disagrees and mentions that "in many cases the hotspots are driven by specific hardware needs and there's little you can do about those". Interestingly, however, P8 noted earlier that it would be counterproductive to focus "on irrelevant parts (ie a part that only uses 1% of the power to begin with)". Furthermore, P3 mentions "finding hotspots by measuring energy consumption of peripherals inside the MCU" as another subgoal, but P4 warns that "knowing what parts of a system consume energy is not trivial". Furthermore, P1 notes that embedded firmware engineers require in-depth hardware knowledge to optimize for energy consumption and P7 says that "usually it is very logical to start with the low hanging fruit and work your way up from there".

To summarize

We can conclude that, in practice, embedded systems developers optimize software to meet an energy consumption requirement and that currently this requirement is only set for systems that have a limited or fluctuating power supply. When such a requirement is set, it is taken into account extensively during the whole development process. The hardware design is a first and important step towards meeting this energy consumption requirement. Furthermore, developers often design the software using an event-driven architecture in which components are put to sleep or turned off and interrupts are used to wake the system. Next, they either first implement the system's functionality and apply optimizations later or they already make the first implementation energy efficient. In any event, they measure energy consumption after applying optimizations to see if these meet their desired effect. When aiming for ultralow power, problems are bound to occur. These problems can be spotted during development or even after release (though this can be prevented by validation before each release). When problems are discovered, their causes are often located by isolating components. Many of

these causes are hardware-related. Changes to hardware design to solve these problems are not uncommon. However, to avoid lead times and save on costs, many problems are fixed using a less ideal solution in the software. Though hardware redesigns currently do not happen solely to improve energy consumption, they are used for future product iterations. Optimization is stopped when the energy consumption requirement is met. Long-term tests or use cases are used to determine if this is the case. Lastly, almost all developers find understanding the energy related behavior of and finding energy consumption hotspots in embedded systems software a useful step in optimizing its energy consumption.

4.3 Information

We will now discuss the results that pertain to the sub research question: *Which information do embedded systems software developers already use to optimize the energy consumption of their software?*

4.3.1 Information seeking subgoals

Many subgoals generate information which is then subsequently used in other subgoals. Participants measured the energy consumption of the system or its components in order to learn about the initial energy consumption or the effect of an optimization. Furthermore, they compared it to expected energy consumption to determine whether further energy consumption improvements were needed, to locate the cause of an issue, or to find an opportunity to save energy. Participants also looked up or estimated energy consumption of components, and of the whole system by calculating a use case, to select hardware that meets the energy consumption requirement, to be able to identify problematically high energy consumption and/or determine whether further energy consumption improvements are needed.

We will list information sources that participants consulted that directly result from the aforementioned subgoals as well as others. We first discuss information sources that participants used to obtain energy measures. Next, we discuss other information sources that participants use when optimizing embedded systems for energy consumption.

4.3.2 (Energy) measures

Almost all participants often use energy measures and also deem them very useful. In the questionnaire, P7 somewhat agreed and all other participants strongly agreed with the statement that "Energy estimations or measurements are useful for optimizing energy consumption of embedded systems software". Participants deem both estimations and measurements to be useful, though measurements seem to be favored. P4 states that measurement is "very good", "useful" and claims that it is "most important" as it "gives hard data". P8 mentions that measuring gives "an indication of future development" and P2 states that measurement equals knowledge and that some form of measurement is always needed for power-critical applications. Furthermore, P4 expresses that estimations can help to identify scenarios with high energy consumption and P9 mentioned that relying on estimations is very helpful to keep up the development pace when the full setup cannot be run.

Datasheets and estimations All participants except for P1 and P8 mentioned consulting datasheets during their optimization process. P6, P7, and P9 remarked consulting datasheets to learn about the intended working of the component. P6 and P9 specifically looked up how a component could be put to sleep. P2, P3, P5, P6, and P9 mentioned using datasheets to check the energy consumption of hardware components. P2, P3, P5, and P6 remarked that they used this information during the hardware design process. P5 and P9 mentioned using the energy consumption reported in the datasheets to validate energy consumption measurements. In the questionnaire, P5 explains using it for this purpose because "hardware often behaves different than theoretically designed". P6 instead used their power budget (which they created using estimations) to validate energy consumption measurements. In contrast, P7 admitted that they never adjusted the goals they set based on the use case that they created and forgot about them.

In another project, P2 selected a processor by taking into account the sleep modes it could go into. P6 also mentioned taking possible sleep modes into account for choosing a microprocessor

and for other components as well. P2 also mentioned the use of a tool to estimate the power consumption of a chip which could prove useful in hardware selection. Similarly, P3 plans to use a tool to estimate the expected power consumption in a personal project. Lastly, P1 also mentions using datasheets and estimation tools in the questionnaire.

Measurements and visualizations All participants used power measurements during their optimization process. To obtain these measurements, P1 and P8 mentioned using a power meter and P5, P6, and P7 said that they used a multimeter. Furthermore, P1, P3, and P7 mentioned using an oscilloscope and P7 mentioned specifically using a logic analyzer. P7 used the logic analyzer to measure energy consumption over time of different components simultaneously and to interpret protocols to learn about the inner state of their embedded system. Lastly, P2, P4, P6, P7, and P9 all used Nordic's Power Profiler Kit (PPK) to measure power consumption over time. P4 specifically mentioned using its first version and P7 and P9 using its second version.

P4 and P9 both mentioned using the PPK to learn how much power was consumed on average over a period of time. P2, P4, and P9 also mentioned learning about the duration of high power consumption using the PPK. Similarly, P1 used data loggers and long-term loggers on an oscilloscope to derive timing information. To learn about timing, P7 and P9 indicated using server logs of the embedded system. Server logs were also used by P5 and P7 to obtain battery level measures. P7 automatically created a line graph from these battery level measurements to show its decline over time. Furthermore, as previously mentioned in section 4.2.1, P8 used measurements of how long the system could function on one battery charge.

P8 also ran data processing algorithms on a PC to check whether changes to them would reduce their execution time and therefore make them consume less energy. P8 also used unit tests to obtain these measurements. P1 also mentioned using unit tests, but instead to check the workings of the software. Along those lines, P7 and P9 mentioned using a debugger. Lastly, P3 planned to use a code analysis tool that provides suggestions to reduce power consumption.

4.3.3 Optimization resources

P6 mentioned checking the documentation of multiple Real-Time Operating Systems (RTOS) to choose between them. P2 mentioned researching which optimizations their RTOS supported and mentioned consulting a former colleague with experience with that RTOS and energy consumption optimization.

To optimize Bluetooth parameters, P2 and P4 checked the Bluetooth communication parameter standard of Apple. P2 also mentioned checking a less stringent standard earlier on.

To summarize

Many subgoals that developers pursue are ways to uncover information. When optimizing for energy consumption, developers deem energy measures to be very useful. They heavily use datasheets when optimizing for ultralow power but are also sometimes hindered by the inaccuracies in them. Moreover, few developers use estimation tools, but all collect power measurements using physical measurement tools. Some also use these and other tools (e.g. server logs) to obtain timing information and use server logs also for battery measurements. Lastly, RTOS documentation and Bluetooth communication parameter standards help developers apply optimizations using their respective technology.

4.4 Challenges

We will now discuss the results that pertain to the sub research question: *Which challenges do embedded systems software developers face when optimizing the energy consumption of their software?*

4.4.1 Tooling

Several participants expressed challenges related to tooling. P1 and P8 expressed criticism of simulation tools, stating that theory does not match well enough with practice which has interrupts and other disturbances. P8 mentioned that simulations "can help to take first steps" but also stated that they cost a lot of time to create and maintain, are not cost-efficient, and that there are

other tools that measure performance that provide more insight. P1, P4, P5, and P7 expressed challenges related to measurement tools. P1 and P4 expressed that specific tools are needed to measure wide power ranges. P4 used the PPK from Nordic and stated that it could accurately measure between 0,01 microampère and 20 miliampère which perfectly suited their case, but that other tooling would be needed in projects where hundreds of watts would need to be measured. P1 also stated that a regular multimeter would be unable to (precisely) measure the high power consumption that they were dealing with. Similarly, P5 mentioned that a regular multimeter would not allow them to find the cause of an issue and that the multimeter that they used could measure up to single microampères, which limited their precision, and that their hardware supplier had more advanced tooling. In the questionnaire P5 also mentioned that "low power applications are continually getting 'lower power'" and therefore measuring "tends to get harder and special equipment is needed". Furthermore, P4 and P7 mentioned that some tooling is very expensive and P4 added that it is sometimes difficult to find where they are sold. Lastly, P4 expressed that the first version of Nordic's PPK crashed often but that the second version is a lot more trustworthy. However, P7 reported that if the second version is used for a longer time it can still miss samples or stop drawing a graph at all.

4.4.2 Testing

P1 and P6 mentioned that software is needed to assess the hardware design. P5 and P7 indicated that it is difficult to draw conclusions from the battery level reported by their product and that measurements can fluctuate. P5 mentioned that conclusions can be drawn only over longer periods of time and P7 said that it only reliably tells them when the battery is almost empty as the curve then drops sharply. P5 explained that they perform a calculation to try to account for this non-linearity and P7 is still thinking of something similar. P8 mentioned that battery charge capacity also varies and thus affects battery life.

P7 and P9 mentioned that the connection with a debugger breaks when the power supply is turned off or the microcontroller is put into a deep sleep mode, which prevents them from debugging during these behaviors. Furthermore, P4 called it a challenge to devise of which behaviors the energy consumption should be measured to calculate a use case. P1, P4, P7, and P9 all experience problems with their current test setups. P4 would prefer a real-life setting to their current artificial test setups. Along similar lines, P9 mentioned that some variables such as which LTE mast their product connects to are constant during development, and field tests are needed to test their influence. During field tests, P7 misses detailed power measures over time such as they would get from Nordic's PPK during development. They reiterate this point in the questionnaire. Furthermore, P1 would prefer more automated testing to improve consistency and documentation. They admit that only for some tests it would be cost-efficient to automate them. Similarly, P6 mentioned that they chose not to automate their version testing as that would be too time-intensive. Lastly, P8 expressed that it is more difficult to locate a component that consumes a lot of energy in an embedded system than on a PC. P8 explained that you can alter software that runs on a PC easily to obtain timing information, but that this cannot be done as easily on an embedded system as that would interrupt the timing which it relies on.

4.4.3 Optimization and hardware

P4 and P9 expressed that if you "go all out" and "try to push your hardware to extremes" you "encounter more hardware bugs" and that "even though things meet certain standards in theory, they go less smoothly in practice.". Indeed many participants encountered challenges whilst optimizing.

Many participants encountered problems in deactivating and reactivating hardware components. P7 and P9 had to give up on turning off a flash chip and P4, P6, and P9 encountered sleep issues. The causes and solutions to these and other problems can be found in Table 5. Lastly, P7 also noted that the creation of an automatic way to turn off component energy supply was difficult to get completely right.

P4, P8, and P9 encountered difficulties whilst optimizing the communication of their embedded system. P4 needed to set up a Bluetooth connection with another module as well as a phone and could not change the parameters for both types of connections separately. Therefore, they had to conform to Apple's Bluetooth communication standard for both connections which prevented further energy consumption savings. Furthermore, P8 and P9 encountered some issues whilst

creating a custom communication protocol and P8 tried to bundle the revision of their protocol in a single update to minimize time investment for their clients.

As already mentioned in section 4.2.1, P6 voiced that unexpected issues are identified during the optimization process. P4 shared this view and emphasized the necessity of measurement. P3 added that the feasibility and effect of optimizations is also uncertain. Furthermore, P4 mentioned in the questionnaire that knowing what parts of a system consume energy and evaluating the result of an improvement are both not trivial. In the questionnaire, P1 and P8 mention that measurement is essential to know the effect of optimizations. Lastly, P8 mentioned that sometimes companies stop the optimization process early, even though a need for further energy saving is expected in the near future, and continuing the process a bit longer would have been more efficient.

As mentioned earlier in section 4.2.1, P6 and P8 noted that specialized hardware makes the embedded system more complex. P6 mentioned that this in turn complicates validating the power consumption of the system and P8 mentioned that it increases the chance of something going wrong of which the cause would be difficult to find. Furthermore, P4 discovered that Electro Static Discharge (ESD) can increase power leakage which significantly affect battery life in low-power systems. P8 also noted that prototypes can also easily break because of short-circuiting which further problematizes a shortage of prototypes and measurement tools which they experienced in the companies they worked at. Though this can be solved by buying more measurement tools and hardware components, P5 and P7 indicated that shortages of components are another challenge in developing embedded systems with low energy consumption. P1 raised that components also show slight deviations and P6 and P8 mentioned that long lead times are another limiting factor during development.

4.4.4 Conflicting goals

P9 mentioned that "everything is of course always a tradeoff" and indeed optimizing for energy consumption can also conflict with other goals during the development of an embedded system. Participants mentioned that it could conflict with minimizing cost, pursuing simplicity and maintainability, and maximizing user experience. As already mentioned in section 4.2.1, P6 decided against including a specialized hardware component to maintain simplicity and reduce development costs. P6's quote that "a solution in software is often cheaper, but often not the most optimal solution" also implies that cost prevents making hardware changes to optimize for energy consumption. A sentiment that P8 agreed with when stating that, due to costs, they had never seen hardware changes being made solely to optimize energy consumption. P1, P4, P7, and P8, however also raised that energy consumption optimization can lower cost. P7 and P8 mentioned that costs can be reduced by reducing the number of batteries or size of batteries and/or solar panel (in the case of an energy harvesting device). P7 also reiterated this in the questionnaire.

P8 also raised that applying software engineering optimizations can make code faster but also less readable and maintainable. Lastly, P4, P5, and P6 raised that optimizing for energy consumption can negatively impact other performance measures and therefore user experience. Optimizations can destabilize connections (P2), lengthen response delays (P4,6), make data less real-time (P5,6), and lower output frequencies (P8). Conversely, P4 raised that optimizing for energy consumption can also enhance user experience by reducing the number of times that batteries need to be recharged or replaced.

To summarize

Developers currently experience challenges whilst optimizing embedded systems software for energy consumption. Some developers find simulation tools too time-intensive and inaccurate to use. Additionally, some developers regret that measurement tools support only narrow power ranges. Furthermore, some developers experience difficulties with current test setups and desire ones that cost less manual effort and provide more ecologically valid and detailed data. Lastly, according to some developers, lower energy consumption may come at the cost of simplicity and maintainability as well as increase cost and negatively affect user experience. However, some expressed that it might also decrease cost and enhance user experience.

4.5 Needs

We will now discuss the results that pertain to the sub research question: *Which needs do embedded systems software developers have when optimizing the energy consumption of their software?*

As already mentioned in section 4.4.2, P7 expressed a need for detailed power measures over time such as they got from Nordic's PPK during development. Furthermore, P4, P8, and P9 would ideally like to see for every component how much power it draws or energy it consumes. P8 mentioned that knowing how much time a piece of code costs to execute is already valuable information. P9, however, notes that measuring for every component how much power it draws or energy it consumes would require extensive hardware support which is difficult and probably not worth the trouble. As already explained in detail in section 4.2.1, they state that it depends on how low the energy consumption should be, but that hardware components can already be recognized sufficiently well from a power consumption measurement of the whole system over time due to the difference in magnitude and timing of their power consumption. Ideally, P1, P4, and P7 would like power/energy not to be measured by external tools but by the embedded systems itself. This would allow P4 and P7 to test their product with realistic parameters in the field whilst getting detailed measurements without using artificial test setups. Ideally, P4 would like to see these measurements be used to detect high-energy events which could trigger an alert in safety critical systems as something might be broken. Similarly, P1 would like these measurements to allow the embedded system to do a self-check and even calibrate itself automatically to deal with deviations in components. In the questionnaire, P4 added the condition that these in-product measurements should not only be possible without additional hardware but also without additional cost or performance loss.

As already discussed in section 4.4.2, P1 would prefer more automated testing to improve consistency and documentation. P8 also expressed a desire for more systematic testing, but as a way to compare the performance of different algorithms. Lastly, a couple of positive and negative experiences of P1, P5, and P6 indicated a need of them to have good communication between the producers of the hardware and the software of the embedded system.

To summarize

Some participants want measurements of individual components as well as in-product measurements.

4.6 Energy measure needs

We will now present our results regarding what kind of energy measures participants desire to be collected and how they want them to be related to the source code. Though related to SQ2 (which was addressed in section 4.3) and SQ4 (which was addressed in section 4.5), these questionnaire results more directly aim to answer the main research question.

4.6.1 Kind of energy measures

In the questionnaire, participants were asked various questions about which kind of energy measures should be collected.

Power or energy When asked whether measures of power draw or energy consumption are more useful when optimizing, many participants favored energy consumption measures. P5 and P8 found measures of power draw to be slightly more useful, whereas P1, P2, P4, and P9 considered energy consumption to be slightly more useful and P3, P6, and P7 found energy consumption to be much more useful.

P1 expressed that both measures are almost equally important. P4 mentioned that energy consumption often suffices, but that power is also very relevant in many cases (e.g. to avoid high temperatures or deal with power supply limits). In contrast, P8 expressed that, for the full product, energy consumption is more important. P9 mentioned that energy consumption is more accurate and P3 and P7 explained that for battery powered devices the consumption over a long period of time is most relevant. Furthermore, P7 expressed that "power usage over time is critical, because peaks in consumption might otherwise not be visible". P2, P8, and P9 acknowledge, however, that power draw is measured more easily and P2 mentioned that energy consumption can be calculated

from power draw. P5 even advocated measuring voltage and current separately to know which one is off, in case of an anomaly, and be able to benchmark against rated current values which come with components and datasheets. P5 also repeated their desire for current measures later on in the questionnaire.

Executions When asked whether energy measures should be collected for a single, several, or all possible execution(s), 6 participants (P2,3,4,5,8,9) selected "several" and the remaining three participants (P1,6,7) selected "all possible". P1, P6, and P9 expressed that they ideally want energy measures for all possible code paths. P4, P6, and P8 however, mention that this would be too impractical. P8 states that "best practice is to generate a set of relevant test cases that cover (80%?) of the normal use cases" and that "the outliers typically also benefit from optimizations in these cases". Furthermore, P9 mentions that for "exceptional code paths" they "only need to validate that eventually it'll get back to a reasonable state". In contrast, P6 says that "due to budget constraints, only high risk use cases are verified". P2 also mentions that "you select for scenarios which theoretically cost the most power" and that "also edge cases need to be taken into account". However, P2 also expresses that "simulations of normal use are most useful if you want to validate the energy consumption". Furthermore, P5 mentioned to "take an average of multiple runs to account for fluctuations outside of your control" and P7 expressed that it is better to have more samples because "there are always slight variations in power consumption across multiple products in the field". Lastly, P4 mentioned that energy measures from a single execution are only useful if the worst case is known à priori. This lack of preference for collecting energy measures for a single execution is surprising. Given that during their reenactment and/or interview P4 as well as P2 and P9 showed that they already collect energy measures for single executions. To locate the cause of abnormally high energy consumption or to calculate the use case, they let the system execute a part of its functionality and use the Power Profiler Kit to see the change in power over time. So this question was possibly a little too abstract to uncover this information.

CPU When asked whether energy measures for the central processing unit or external hardware components would be more useful, participants seemed quite divided. P4, P8, and P9 slightly favored CPU, P2, and P6 slightly favored external hardware components, and P1, P3, P5, and P7 strongly favored external hardware components.

From their explanation in the open question, it became clear that all participants who indicated a slight preference actually felt that the answer heavily depends on the project. P2, P6, P8, and P9 all expressed that measures should be obtained for all energy-intensive components. Along similar lines, P7 expressed that "all components that draw power from the battery are important to optimize energy consumption" and, as alluded to in section 4.2.2, P1 explained that in their project, most power is consumed by the external hardware. Later in the questionnaire, however, P1 states that they would still find it nice to have better insight in the power usage of the CPU. P1 and P3 both stated that the CPU is running at constant speed and therefore consumes a relatively constant amount of power. P3 repeats this point several times in the questionnaire, also mentioning that therefore optimizing code does not always lead to less energy consumption. P5 mentions that the consumption of the CPU probably closely matches what is described in the datasheet and P4 and P5 both state that its energy consumption is easily validated. Lastly, P3 and P5 mention that the impact of external components on energy consumption is less straightforward as they can be enabled or disabled (P3) and can have different applications/electric schematics (P5).

Hardware combinations P1, P6, and P8 indicated that the software they write always needs to function on one specific hardware combination. P2 and P3 indicated that it should often function on one specific hardware combination and P4, P5, and P7 answered that it should often function on multiple different hardware combinations. P3 and P4 strongly agreed that seeing energy measures for different hardware combinations would be useful and P2, P5, P7, and P8 somewhat agreed to this statement.

P2 mentioned that due to shortages sometimes components are replaced. Similarly, P7 expressed that seeing measures for different combinations is especially useful when having multiple hardware revisions. P3 raised that software is largely reused for different hardware combinations. Furthermore, P4 and P5 indicated that having these measures beforehand would be useful. Scattered across multiple points in the questionnaire, P4 explains that hardware has a large influence

on energy consumption and therefore simulating up front without owning the hardware could influence hardware decision-making given that these simulations/estimations match with reality. Also, P5 expressed doubts about estimation quality, stating that they still want measurements to validate the energy consumption on the actual hardware. Lastly, P9 explained that they try to create different modules and that measures could be collected for them. Still, they consider it useful to validate the specific combination of modules using measurement.

Usage scenarios P3, P5, and P9 indicated that the software they write should often work under one usage scenario. P7 and P8 indicated that it should often work under multiple different usage scenarios and P1, P2, P4, and P6 answered that it should always work under multiple different usage scenarios. P5, P8, and P9 somewhat disagreed that seeing energy measures for different for different usage scenarios would be useful, whereas the other participants strongly agreed with the statement.

P5, P8, and P9 as well as P3 expressed that it depends on the project. P3 mentioned that when multiple scenarios are relevant, energy consumption should be monitored if it impacts usability. P5 and P9 stated that most projects only have a single scenario. P8 mentioned that only in a few cases they experienced "that a specific scenario deviated far enough from the mean that it warrants specific optimization attention" and that "in many cases the shared components are the dominant factor". The other participants focused on the fact that multiple different scenarios were relevant in their projects. P4 reiterated that the use case plays a major role in their products and P7 mentioned that in their case the outside environment heavily impacted the energy consumption. P1 stated that their embedded systems are used in "many different scenarios (environments)" and that "energy measures are important for all these different scenarios". In contrast, P6 gave an example from their own product to illustrate that power usage is more important for more frequent scenarios. Lastly, P2 argued that multiple different scenarios are always relevant by stating that "expecting a user to do only one thing with your software/hardware is the worst thing one can do".

To summarize

Developers indicated preferences for certain kinds of energy measures. They indicated that power draw is more easily measured, but overall seem to favor energy consumption measurements as it includes the time dimension. Though some developers prefer energy measures to be collected for all possible code paths, most want energy measures to be collected for several executions as they either believe that collecting measurements for all possible code paths is impractical or that not all code paths are necessary or relevant to consider. According to most, it depends on the project whether energy measures of the CPU or external hardware components are more relevant. However, energy measures for external hardware components seem to be more relevant for most projects. For developers dealing with multiple different hardware combinations, energy measures for these different combinations seem promising. According to some developers, this could guide the initial hardware design as well as hardware revisions provided that the measures are accurate. Lastly, embedded systems seem to regularly need to work under different scenarios and if so, developers deem it useful to have energy measures for these different scenarios.

4.6.2 Relation to source code

In the questionnaire, various questions were also asked about how energy measures should be related to the source code

Abstractions Figure 3 shows how every participant ranked 6 different program abstractions in descending order of how useful it would be if energy measures were related to them when optimizing.

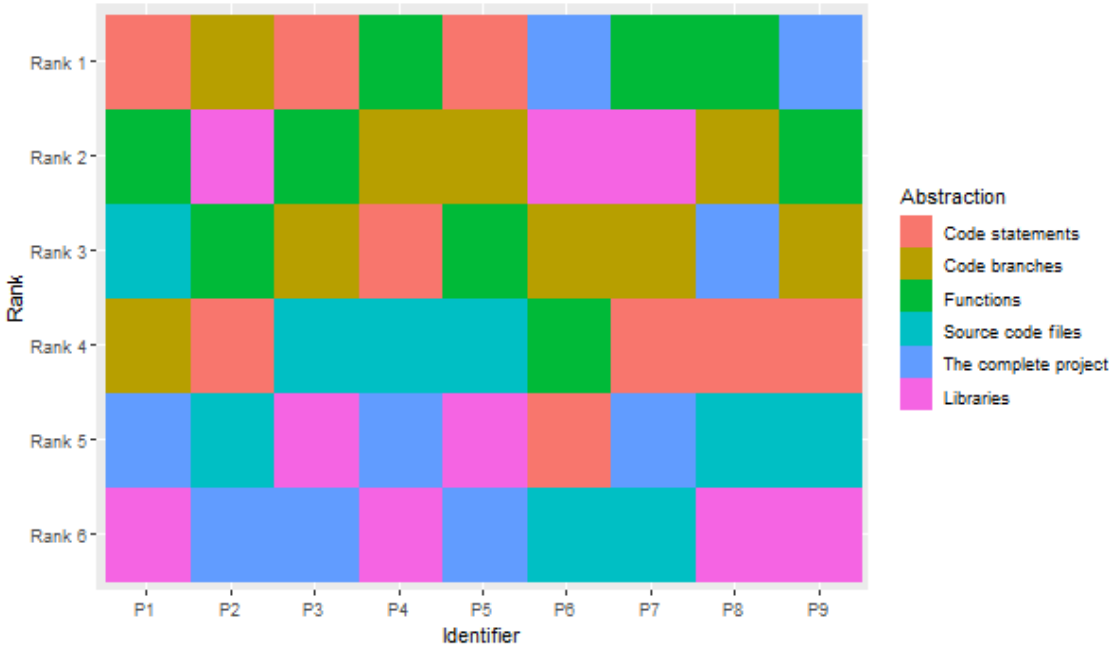


Figure 3: Response of every participant to the question: "Please rank the following program abstractions in descending order of their usefulness for optimizing the energy consumption of embedded systems software."

Most participants deem it more useful when energy measures are related to low-level program abstractions (code statements, branches, and functions) than to high-level program abstractions (source code files, the complete project, libraries).

P4, P7, and P8 clearly favor functions. P4 mentioned that functions are a sweet spot, providing enough level of detail as well as being sufficiently informative. P7 agreed that functions are not too big and stated that they can be optimized individually. Lastly, P8 mentioned that "organizing your code to work efficiently is done by measuring function times". P1, P3, and P5 clearly favor code statements. They stated in their explanation that optimizing starts with the smallest piece of code (P1), that measures provided at this "lowest level" make it easier to tune the power consumption of a MCU (P3), and that "Most of the time, you can save on energy on the smaller things. Like configuring a peripheral or register slightly different." (P5). Other participants rank code statements quite low. In their explanation, P6 mentioned that "premature optimization" on code statement level should be prevented (which they also reiterate later on in the questionnaire) and P8 raised that "individual statements are typically optimized by a compiler". In explaining their answer to a later question, P6 did mention that "power usage by code statements is useful when it can detect while(true) loops requiring 100% CPU". Code branches are consistently ranked quite high, but it is unclear why, as only P8 mentioned branches in their explanation, stating that reducing them helps for modern CPUs. Lastly, P2 mentioned that code blocks or configurations are disabled to isolate components during optimization and P5 raised that even though energy can be saved on the smaller things, "it all starts with a well thought out design".

High-level program abstractions (source code files, the complete project, and libraries) are often ranked below low-level ones. With respect to libraries, P1 mentioned that they hardly use them as the internal code is unknown. P7 however expressed that libraries and especially third-party ones are "quite overlooked" which explains why they ranked them second. Though P3 ranks libraries second to last here, they are more enthusiastic about relating energy measures to them after being given the option to relate energy measures to multiple abstractions and have them be shown simultaneously using their hierarchical relation. They then state that libraries are inefficient but fast to implement and that workarounds can be programmed more easily if problems can be traced back to statements in libraries. It is unclear why P2 ranked libraries so high (especially because they did not select them in a next multiple-choice question). P6 might have ranked the complete project as well as libraries high due to their top-down strategy and the ranking of P9 should be interpreted with caution as they later admit that they found it difficult to rank the program abstractions. We will discuss both these points later on. Lastly, with respect to the full

project, P8 stated that it is "the ultimate test of how things fit together".

Multiple abstractions Participant's responses centered around "somewhat agree" when presented with the statement that it would be useful if energy measures were related to multiple different program abstractions. P1 and P2 somewhat disagreed with this statement, P5, P6, P7, and P8 somewhat agreed and P3, P4 and P9 strongly agreed.

Figure 4 shows which program abstractions the participants would want energy measures to be related to. P4 chose all abstractions and P6 only selected the complete project and libraries. The selections of the other participants appear quite consistent with the ranking that they gave before. Again, low-level program abstractions (especially functions) seem more popular than high-level ones.

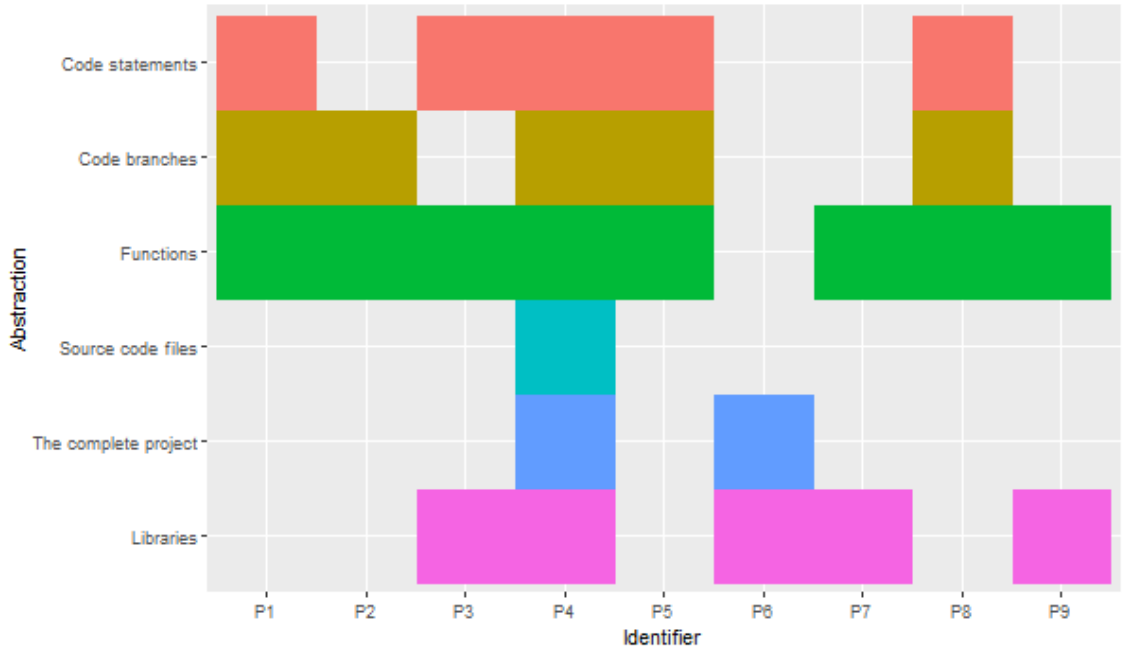


Figure 4: Response of every participant to the multiple choice question: "If the energy measures are related to multiple different program abstractions, these should be:"

When asked whether it would be useful if the energy measures related to these multiple program abstractions were shown simultaneously using their hierarchical relation, P2, P5, and P6 agreed somewhat and the others agreed strongly.

P4 and P6 explained their answers by reiterating the explanation which they gave for their ranking. Namely, that they want to take a top-down approach. Selecting all abstractions allows P4 to "start at the top abstraction level" and "drill down to the lowest level". In contrast, P6 explained that they "start by selecting hardware and software components suitable for low power management" for example a RTOS. Similar to P4, P5 wants to "narrow down where the (excessive) energy consumption is occurring". However, they selected only code statements, code branches, and functions because they believe that energy consumption measurements related to higher levels would be meaningless because "it also incorporates code that might not be used in the project but are part of a library for instance.".

P2 mentions that this splitting into components can be easily done if "the program is set up so that each class or code block has a clear and single responsibility". Later they state that this split can be done on a higher level given a clear separation of concerns in the code and that therefore a "good architecture helps". They clarify that for them "having a view on what the power draw is when bluetooth is enabled with the potentiostat enbled and with the potentiostat disabled would be most useful, any lower level then that is overkill". P9 seems to desire energy measures on a similar level. Stating that they selected libraries, but would rather have energy measures related to a 'system' e.g. one that controls the radio or the accelerometer. They state that this "may go across the bounds of the abstractions ... mentioned" but state that systems have a dependency on

one another and that therefore "showing a hierarchy is probably good". P1 mentioned seeing this dependency between the listed abstractions.

P3, P7, and P8 are positive in their explanation. P7 expressed that they think that "giving an hierarchical relation would make for an easy to understand overview on where the most power is consumed in the firmware" and P3 stated that "this makes it easy to find the problems and tune these". P8 mentioned that "this helps a lot" and mentioned that a tool called Performance Validator does the same thing for PC software times.

Show with source code In contrast to the previous questions/statements about relating energy measures to program abstractions, responses to the next statement are very divided. When presented with the statement that when optimizing it would be useful if the energy measures are shown close to or in conjunction with the source code in a code editor, P1, P5, and P6 strongly disagree, P2 somewhat disagrees, P4, P7, P8, and P9 somewhat agree and P3 strongly agrees. Some positive comments and criticisms are clearly specifically related to this statement. Others seem to regard the idea of relating energy consumption to program abstractions which underlies this as well as previous questions/statements.

P1, P6, P8, and P9 make comments about the feasibility of relating energy measures to program abstractions. P1 mentions that they "can not think of a way this should work" which would also explain their somewhat disagreement to a previous statement about relating energy measures to multiple different program abstractions. P8 is also critical, stating that "it's very hard to connect energy consumption to a few specific lines of code". Similarly, P9 states that in embedded systems a lot of power draw is not directly related to the code statement by statement. They also mentioned that they found it difficult to rank the program abstractions as their perspective is "really on the hardware" and from that perspective it is not the function but the hardware component that consumes power. Lastly, they raise the question of how power draw would be attributed if a function turns on a piece of hardware but does not shut it down. P6 also comments on the feasibility, stating that in the case of a RTOS, code power usage "would require the simulation of the entire application in order to deduce CPU usage of software components". They mention that "most RTOS's support this type of runtime view" and that for example "Segger systemviewer shows CPU usage per task". Despite their criticism, P8 mentions that "it would be nice" and P9 states that they think that in embedded systems it is less useful in general, but it is still useful in specific cases. Also P7 is somewhat enthusiastic, saying that it is a "nice to have".

Other comments seem to pertain more specifically to how energy measures are shown. P3, P4, and P7 make positive remarks. P3 mentions that "this makes it easy to create a overview of the current state and factors that can be tuned for the programmer", P7 says that "it could give a clear insight on what operations are the most expensive" and P4 calls it "a user friendly solution". P7 expresses some criticism, however, stating that "it would be nice to see some sort of heat map of your source code, but this doesn't necessarily have to be in the same editor that you use to code". Lastly, though P4 called it a user-friendly solution, they also stated that other solutions may be more effective. Specifically, they mention that source code is very flat and has no time axis and that they think that much data could better be presented in a graphical way.

P8 also hinted towards potentially more effective solutions, stating that "typically the relevant thing is the *change* in energy consumption following a change in code, not the absolute numbers" (they repeat this statement later in the questionnaire). P5 and P6 also mention that other steps in the optimization process, namely upfront design, may be more important. P5 already ended their explanation after the abstraction ranking question by stating that "it all starts with a well thought out design" and answered here that "software cannot dictate actual power consumption" and that "a strong factor is hardware, circuitry and component selection". Similarly, P6 stated that "in general, most benefits can be made by upfront hardware and software design" and later mentioned again that "design for low power in hardware and software" is useful.

Alternative abstractions Some participants introduced alternative abstractions for which energy measures might be collected. As already explained in sections 4.6.1 and 4.6.2, P9 wants energy measures to be collected for modules or related to 'systems' which each control one hardware component. This might be related to the higher level on which P2 wants to split (as mentioned in section 4.6.2). Furthermore, P3 would find it useful when power consumption was monitored for the different sleep modes of the MCU and the different power domains of for instance the UART, RF, and Timers since, according to P3, these domains can often be tuned.

To summarize

When energy measures are related to source code, most developers prefer them to be related to low level abstractions and especially to functions. Developers also deem it somewhat useful if energy measures are related to multiple program abstractions and somewhat to very useful if they are shown simultaneously using their hierarchical relation. Developers are quite divided about whether it would be useful if these energy measures are shown close to the source code. Some expect it to give a clear overview whereas others expect other efforts to be more worthwhile. The strong disagreement of some also seems to be caused by their skepticism of the underlying assumption that energy measures can be reliably related to program abstractions of embedded systems source code.

5 Discussion

In this discussion section, we first identify the limitations of our study. Then, we relate our findings to earlier work whenever applicable. Last, we give recommendations to researchers, tool makers, vendors, and embedded systems software developers.

5.1 Limitations

5.1.1 Self-reporting of past events

As explained in section 3.3.1, to allow the feasibility of the study, a retrospective approach was taken to learn about the current practices of embedded systems software developers when optimizing for energy consumption. This retrospective approach also helps participants to put their experiences into perspective and focus on the most important parts. Unfortunately, it might have also introduced inaccuracies. Participants might have disproportionally emphasized non-frequent or emotion-laden (e.g. frustrating) events or only given examples that fit with their views. Given our interest in the perspectives of developers themselves as well as the challenges they faced, this is not very problematic. Lastly, as we already touched upon in section 3.3.1 the unexpected longer duration of the optimization processes might have concealed some important details. Therefore, future work could try to use observational methods to falsify or verify the results from this study.

5.1.2 Abstract questioning

To prevent participants from focusing on visualization design ideas, care was taken not to present any graphical representations. However, as we have already pointed out in section 4.6.1, this might have made the question about the number of executions for which energy measures should be collected vague and difficult to imagine. Future work might try to make prototypes or even fully functional techniques for these different options to make them more concrete. Having these proof of concepts could then also take away worries of participants regarding the feasibility of relating energy measures to source code.

5.1.3 Response bias

Though response bias cannot be completely ruled out, extra care was taken to increase the chance of truthful and accurate responses. Participants were put at ease and given many opportunities to ask questions. Though interview and questionnaire questions were evaluated to ensure that they were not leading, some statements in the questionnaire could have been made more neutral. Fortunately, this did not seem to have led to acquiescence bias, as disagreement was also expressed by participants. The use of other multiple-choice as well as open questions also aimed to relieve this possible issue. Lastly, questions were carefully ordered to prevent answers to previous questions from distorting answers to future questions. For example, general questions such as whether energy measures are useful in general preceded specific questions such as if energy measures for the CPU or external components would be most useful.

5.1.4 Sampling

Sample size The aforementioned conclusions have to be taken with caution. As already mentioned in section 3.2, saturation was not fully reached and other alternative views might have been

left undiscovered. As also already mentioned in section 3.2, the sample size did not allow for a statistical analysis of the quantitative results. However, as predicted, trends could be identified quite well and most importantly also participants' reasoning was uncovered. Clear hypotheses could be formulated using the results of this study which future work could seek to test statistically

Population Though the experiences and perspectives of embedded systems software developers might overlap with software developers in other application domains, our findings cannot be generalized to this larger group of software developers due to the strong focus on hardware when optimizing embedded systems for energy consumption. Furthermore, our findings suggest a distinction between the development of embedded systems with an energy requirement (e.g. because the system is battery-powered) and those without such a requirement (e.g. because the system has a constant power supply). Thus in the latter case, the experiences and perspectives of embedded systems software developers might differ from our findings. Lastly, the fact that all participants described the optimization of relatively complex embedded systems indicates that our findings might not generalize to the optimization of relatively simple embedded systems.

5.2 Relation to earlier work

Next, we relate our findings to earlier work where possible.

5.2.1 Workflow

The findings of our study are consistent with earlier work on the workflow of embedded systems software developers. Manotas et al. [8] mentioned that, for embedded systems, many practitioners rely on the hardware, not the software to reduce energy consumption. Indeed we found that embedded systems software developers believe that the selection of hardware is an important step toward optimizing energy consumption. Furthermore, many of the steps that we identified that embedded systems software developers take in practice are also mentioned in existing guidelines that we described in section 2.3. Lastly, our findings that embedded systems regularly need to work under different scenarios and that if so, developers deem it useful to have energy measures for these different scenarios fits well with the finding of Manotas et al. [8] that developers often consider usage scenarios when evaluating energy usage.

5.2.2 Subgoals and visualization contexts

Some of our findings can be related to the subgoals and visualization contexts of Isaacs et al. [24]. As a first, developers consider understanding the energy related behavior of embedded systems software and finding energy consumption hotspots in it useful steps in optimizing its energy consumption. This provides evidence for the global comprehension subgoal of Isaacs et al. [24] in the context of energy consumption optimization of embedded systems software. Furthermore, the fact that embedded systems software developers use (component) energy measures to find problems and locate their causes also shows potential for visualizations of these measures to aid in the Problem detection and Diagnosis and attribution subgoals of Isaacs et al. [24]. Lastly, P4's comments that "source code is very flat and has no time axis" and that they think that "much data could better be presented in a graphical way" suggest that non-software contexts might also be considered for visualizations of energy measures. For example, a hardware context might be considered which fits with the hardware perspective that P9 usually takes.

5.2.3 Challenges and tools

Previous work that studied the perspective of software developers concluded that optimizing for energy consumption is challenging [2], [6]–[9]. Our results reinforce this notion for embedded systems software development specifically and reveal specific challenges that these developers face. From our findings, the lack of tools which Pinto et al. diagnosed [6] seems less of an issue for embedded systems developers. This might be due to the history of the embedded systems domain which has long dealt with power constraints and heavily relied on measurement tools. However, this issue might have also been alleviated more recently as many of the tools identified in this study (especially software-based ones) were available only after the paper of Pinto et al. [6] was published. The fact that optimizing for energy consumption remains challenging even in the

presence of measurement and other tools corroborates the findings of Pang et al. [7]. However, the observation of Pinto and Castor that tools are course grained [2] still seems to be relevant as we found that developers manually isolate components to measure their energy consumption.

Furthermore, usability remarks in our findings (criticism about measurement tools crashing and positive remarks about visualization ideas) fit with the call for usability by Manotas et al. [8] and Ournani et al. [9]. Interestingly however, though Ournani et al. [9] and Hindle [27] advocated integration within IDEs, a comment of P7 revealed that not all developers require energy measure visualizations to be in the same editor that they use to code. Furthermore, the comment of P8 that the change in energy consumption following a code change is typically relevant and not the absolute numbers, reinforces the view of Ournani et al. [9] and Hindle [27] who advocate following the evolution of energy consumption through commits using multi-version analysis. The finding that many developers measure energy consumption after applying optimizations to see whether energy consumption has decreased, further strengthens this view. Lastly, it is interesting to note that the transition from external to internal energy measurements which we described for PCs and mobile devices in section 2.4, is also desired for embedded systems by at least some developers.

5.2.4 Objectives and knowledge

Though Manotas et al. [8] found that in software development energy consumption requirements are often desires and not specific targets and often are not stated in terms of energy consumption or battery-life, our findings suggest that the development of battery-powered embedded systems is an exception to this. Furthermore, our findings corroborate the conclusion of Manotas et al. [8] that energy consumption requirements or goals are not set for embedded systems with a constant and unlimited power supply. Combined with our finding that company priorities shift once energy consumption objectives are met, this explains why developers have advocated for these objectives in the study by Ournani et al. [9]. Furthermore, our finding that developers use datasheets and sometimes also other forms of documentation or consult colleagues, reinforces the findings of Manotas et al. [8] that developers believe they can learn about energy consumption from these sources. Whereas previous studies [2], [6]–[9] diagnosed low awareness of energy consumption of software developers, our findings suggest that embedded systems software developers are an exception as they have an extensive process for optimizing for energy consumption and know which components are most energy intensive.

5.3 Recommendations

Last, we provide recommendations for researchers, tool makers, vendors, and embedded systems developers.

5.3.1 For researchers

Investigate currently used tools Future work could seek to investigate tools that embedded systems software developers currently use when optimizing for energy consumption (see Table 7 in Appendix B for all tools identified in this study). Researching how these tools are used, could reveal additional requirements and functionality about what information developers consult and how this is visualized. This work could investigate tools that collect energy measures such as the Power Proflier Kit or simulation tools such as the power optimizer, but also recommender tools like the ULP Advisor.

Create prototypes Future work could also create prototypes for monitoring tools using our findings about what type of energy measures should be collected and how these should be visualized. Testing multiple prototype alternatives with participants could help to refine knowledge about different aspects that did not become completely clear from our questionnaire. For this purpose, also new ways could be created to collect energy measures and reliably relate them to the source code. Having proof of concepts would help to alleviate concerns of developers about the feasibility. Alternatively, these prototypes could also explore non-software contexts (e.g. a hardware context) for visualizations. Furthermore, these prototypes could display the change in energy consumption rather than the absolute values.

5.3.2 For tool makers

Creators of existing measurement and simulation tools can aim to address the problems with their tools identified in our study. Furthermore, like researchers, tool makers can use our findings about what type of energy measures should be collected and how these should be visualized to guide the development of their tools.

5.3.3 For vendors

Embedded system vendors can aim to reduce the number of hardware bugs that hinder the optimization for energy consumption. Documentation of hardware bugs or other limitations in datasheets would also help developers to diagnose and resolve these problems more quickly. Additionally, making the energy consumption measurements in datasheets more realistic could help to better manage developers' expectations. Lastly, by reducing lead times, vendors can help enable hardware redesigns for reduced energy consumption.

5.3.4 For embedded systems software developers

embedded systems software developers that wish to optimize for energy consumption can adopt the workflow and suggestions documented in this study. When little time is available, using sleep modes and optimizing the most energy-intensive components could already help to greatly reduce the energy consumption of embedded systems. If more time is available, hardware and software can be carefully chosen, optimizations can be devised and energy measures can be estimated or measured in advance and measured after each implementation and optimization to track the energy consumption during the optimization process.

6 Conclusion

In this thesis, we have tried to take a first step towards problem-driven energy visualizations to support embedded systems software developers in optimizing for energy consumption. As explained in section 3.1, we have specifically engaged in the discover stage of the nine-stage design framework. We have uncovered which information embedded systems software developers need when optimizing for energy consumption as well as which subgoals they pursue, which information they currently use, which challenges they face, and which other needs they have.

The reenactments and interviews gave us insight into the current workflow of embedded systems software developers: their current subgoals, challenges, information, and needs. However, except for showing that energy measures are important, these methods did not directly give us many insights for visualizations. As explained in section 5.1.1, we might have missed visualization-relevant details due to our choice of methods as well as the length of the optimization processes that participants reenacted. Furthermore, as explained in section 5.1.4, it is important to note that, without further research, the uncovered workflow can only be generalized to the development of complex embedded systems that have a set energy consumption requirement.

In contrast to the reenactments and interviews, the questionnaire found clear preferences for what kind of energy measures should be collected (see section 4.6.1) and how they should be related to source code and visualized (see section 4.6.2). As discussed in sections 5.3.1 and 5.3.2, researchers and tool developers can create prototypes of energy consumption visualizations to verify and refine these preferences and take away skepticism about the possibility of relating energy measures to program abstractions. As discussed in section 5.2.2, these prototypes could also experiment with other visualization contexts and subgoals as the reenactments and interviews indicated their possible usefulness.

The insight of the reenactments and interviews in the current workflow of developers also provided recommendations for hardware vendors as well as embedded systems software developers themselves (see sections 5.3.3 and 5.3.4). Furthermore, as discussed in section 5.2, the workflow revealed that these developers have more knowledge and tools than previously expected and our other findings are very consistent with earlier work. Lastly, the workflow which we identified is also important for energy visualizations as these tools should be integrated well into this current workflow to ensure their adoption.

Thus, by uncovering the existing practices and information needs of embedded systems software developers, we have been able to give recommendations to researchers, tool makers, vendors, and

developers themselves to help embedded systems meet stricter energy consumption requirements and have lower greenhouse gas emissions.

Acknowledgements

We would like to thank Fernando Castor for our early conversations and his thought-provoking questions. Furthermore, we want to thank Evanthia Dimara for her guidance with respect to the methodology. We want to thank Bram Abbekerk, Gido Hakvoort and especially Johan Korten for sharing their expert knowledge and contacts. Additionally, we wish to thank Lauren Beehler and Roy Schoonwater for their feedback and assistance. Lastly, we want to thank all the participants who took part in our study for sharing their time and perspectives.

References

[1] C. Marantos, L. Papadopoulos, C. P. Lamprakos, K. Salapas, and D. Soudris, "Bringing Energy Efficiency Closer to Application Developers: An Extensible Software Analysis Framework," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 2, pp. 180–193, Apr. 2023, Conference Name: IEEE Transactions on Sustainable Computing, ISSN: 2377-3782. DOI: 10.1109/TSUSC.2022.3222409.

[2] G. Pinto and F. Castor, "Energy efficiency: A new concern for application software developers," en, *Communications of the ACM*, vol. 60, no. 12, pp. 68–75, Nov. 2017, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3154384.

[3] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations," *Patterns*, vol. 2, no. 9, p. 100 340, Sep. 2021, ISSN: 2666-3899. DOI: 10.1016/j.patter.2021.100340.

[4] S. Georgiou, S. Rizou, and D. Spinellis, "Software Development Lifecycle for Energy Efficiency: Techniques and Tools," *ACM Computing Surveys*, vol. 52, no. 4, 81:1–81:33, Aug. 2019, ISSN: 0360-0300. DOI: 10.1145/3337773.

[5] J. Balanza-Martinez, P. Lago, and R. Verdecchia, "Tactics for Software Energy Efficiency: A Review," en, in *Advances and New Trends in Environmental Informatics 2023*, V. Wohlgemuth, D. Kranzlmüller, and M. Höb, Eds., Cham: Springer Nature Switzerland, 2024, pp. 115–140, ISBN: 978-3-031-46902-2. DOI: 10.1007/978-3-031-46902-2_7.

[6] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, New York, NY, USA: Association for Computing Machinery, May 2014, pp. 22–31, ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597110.

[7] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What Do Programmers Know about Software Energy Consumption?" *IEEE Software*, vol. 33, no. 3, pp. 83–89, May 2016, Conference Name: IEEE Software, ISSN: 1937-4194. DOI: 10.1109/MS.2015.83.

[8] I. Manotas, C. Bird, R. Zhang, *et al.*, "An empirical study of practitioners' perspectives on green software engineering," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16, New York, NY, USA: Association for Computing Machinery, May 2016, pp. 237–248, ISBN: 978-1-4503-3900-1. DOI: 10.1145/2884781.2884810.

[9] Z. Ournani, R. Rouvoy, P. Rust, and J. Penhoat, "On Reducing the Energy Consumption of Software: From Hurdles to Requirements," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '20, New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–12, ISBN: 978-1-4503-7580-1. DOI: 10.1145/3382494.3410678.

[10] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating source line level energy information for Android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ser. ISSTA 2013, New York, NY, USA: Association for Computing Machinery, Jul. 2013, pp. 78–89, ISBN: 978-1-4503-2159-4. DOI: 10.1145/2483760.2483780.

[11] R. Verdecchia, A. Guldner, Y. Becker, and E. Kern, "Code-Level Energy Hotspot Localization via Naive Spectrum Based Testing," en, in *Advances and New Trends in Environmental Informatics*, H.-J. Bungartz, D. Kranzlmüller, V. Weinberg, J. Weismüller, and V. Wohlgemuth, Eds., ser. Progress in IS, Cham: Springer International Publishing, 2018, pp. 111–130, ISBN: 978-3-319-99654-7. DOI: 10.1007/978-3-319-99654-7_8.

[12] M. Couto, T. Carção, J. Cunha, J. P. Fernandes, and J. Saraiva, "Detecting Anomalous Energy Consumption in Android Applications," en, in *Programming Languages*, F. M. Quintão Pereira, Ed., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2014, pp. 77–91, ISBN: 978-3-319-11863-5. DOI: 10.1007/978-3-319-11863-5_6.

[13] M. Klinik, B. v. Gastel, C. Kop, and M. v. Eekelen, "Skylines for Symbolic Energy Consumption Analysis," en, in *Formal Methods for Industrial Critical Systems*, M. H. ter Beek and D. Ničković, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 93–112, ISBN: 978-3-030-58298-2. DOI: 10.1007/978-3-030-58298-2_3.

[14] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440–459, Jan. 2014, Publisher: Inderscience Publishers, ISSN: 1757-2657. DOI: 10.1504/IJCAET.2014.065419.

[15] H. Roebbers, "Achieving Ultra Low Power in Embedded Systems Understand where your power goes and what you can do to make things better," en. [Online]. Available: https://www.researchgate.net/publication/332878839_Achieving_Ultra_Low_Power_in_Embedded_Systems_Understand_where_your_power_goes_and_what_you_can_do_to_make_things_better_2019.

[16] M. Sedlmair, M. Meyer, and T. Munzner, "Design Study Methodology: Reflections from the Trenches and the Stacks," en, *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2431–2440, Dec. 2012, ISSN: 1077-2626. DOI: 10.1109/TVCG.2012.213.

[17] K. Tanaka, Ed., *Embedded Systems - Theory and Design Methodology*, en. InTech, Mar. 2012, ISBN: 978-953-51-0167-3. DOI: 10.5772/2339.

[18] M. Pedram, "Power optimization and management in embedded systems," in *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '01, New York, NY, USA: Association for Computing Machinery, Jan. 2001, pp. 239–244, ISBN: 978-0-7803-6634-3. DOI: 10.1145/370155.370333.

[19] K. Georgiou, S. Xavier-de-Souza, and K. Eder, "The IoT Energy Challenge: A Software Perspective," en, *IEEE Embedded Systems Letters*, vol. 10, no. 3, pp. 53–56, Sep. 2018, ISSN: 1943-0663, 1943-0671. DOI: 10.1109/LES.2017.2741419.

[20] A. Noureddine, "PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools," in *2022 18th International Conference on Intelligent Environments (IE)*, ISSN: 2472-7571, Jun. 2022, pp. 1–4. DOI: 10.1109/IE54923.2022.9826760.

[21] R. Pereira, T. Carção, M. Couto, J. Cunha, J. P. Fernandes, and J. Saraiva, "SPELLing out energy leaks: Aiding developers locate energy inefficient code," *Journal of Systems and Software*, vol. 161, p. 110 463, Mar. 2020, ISSN: 0164-1212. DOI: 10.1016/j.jss.2019.110463.

[22] H. A. de Souza, M. L. Chaim, and F. Kon, *Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges*, Nov. 2017. DOI: 10.48550/arXiv.1607.04347.

[23] K. Eder and J. P. Gallagher, "Energy-Aware Software Engineering," en, in *ICT - Energy Concepts for Energy Efficiency and Sustainability*, G. Fagas, L. Gammaitoni, J. P. Gallagher, and D. J. Paul, Eds., InTech, Mar. 2017, ISBN: 978-953-51-3012-3. DOI: 10.5772/65985.

[24] K. E. Isaacs, A. Giménez, I. Jusufi, *et al.*, "State of the Art of Performance Visualization," en, 2014. DOI: 10.2312/EUROVISSTAR.20141177.

[25] A. Blandford, D. Furniss, and S. Makri, *Qualitative HCI Research: Going Behind the Scenes*, en. Morgan & Claypool Publishers, Apr. 2016, ISBN: 978-1-62705-760-8. [Online]. Available: https://dx.doi.org/10.1007/978-3-031-02217-3.

[26] L. G. Militello and R. J. B. Hutton, "Applied cognitive task analysis (ACTA): A practitioner's toolkit for understanding cognitive task demands," *Ergonomics*, vol. 41, no. 11, pp. 1618–1641, Nov. 1998, Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/001401398186108, ISSN: 0014-0139. DOI: 10.1080/001401398186108.

[27] A. Hindle, "Green Software Engineering: The Curse of Methodology," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, Mar. 2016, pp. 46–55. DOI: 10.1109/SANER.2016.60.

Appendices

A Assumptions

Table 6 lists all assumptions that the visualizations of Li et al. [10], Verdecchia et al. [11], Couto et al. [12] and Klinik et al. [13] make with respect to which information is visualized.

Category	**Assumptions**	[10]	[11]	[12]	[13]
EC or Power draw	Energy Consumption	✓	✓	✓	
	Power draw				✓
Executions	Averaged over a fixed set of executions (derived from test cases)	✓	✓	✓	
	All possible executions				✓
CPU or External	CPU energy measures	✓	✓	✓	
	External component energy measures				✓
Code proximity	Energy measures should be shown close to or in conjunction with the source code in a code editor	✓			✓
Program abstractions	Code statements	✓			✓
	Branches		✓		
	Methods		✓	✓	
	Classes			✓	
	Packages			✓	
	Application			✓	
Multiple	Multiple program abstractions		✓	✓	
	Hierarchy information			✓	

Table 6: All assumptions that the visualizations of Li et al. [10], Verdecchia et al. [11], Couto et al. [12] and Klinik et al. [13] make with respect to which information is visualized.

B Tools

Tool	Brand	Tool type	Participant
SDM 3065x digital multimeter	Siglent	Measurement	P6
Logic 8 Logic analyzer	Saleae	Measurement	P7
Power Profiler Kit	Nordic	Measurement	P2,4,6,7,9
Power Profiler Kit 1	Nordic	Measurement	P4
Power Profiler Kit 2	Nordic	Measurement	P7,9
Power consumption calculator	STM32	Estimation	P3
Communication power consumption estimator	Nordic	Estimation	P2
ULP Advisor	Texas Instruments	Recommender	P3
Zephyr integrated in nRF SDK	Nordic	OS	P2,6

Table 7: Tools mentioned by at least one participant