Tactics for Software Energy Efficiency: A Review

Jose Balanza-Martinez[1], Patricia Lago[1], and Roberto Verdecchia[2]

[1] Vrije Universiteit Amsterdam, The Netherlands
j.b.balanzamartinez@student.vu.nl, p.lago@vu.nl,
[2] Università degli Studi di Firenze, Italy
roberto.verdecchia@unifi.it

Abstract. *Context:* Over the years, software systems experienced a growing popularization. With it, the energy they consume witnessed an exponential growth, surpassing the one of the entire aviation sector. *Energy efficiency tactics* can be used to optimize software energy consumption.
Objectives: In this work, we aim at understanding the state of the art of energy efficient tactics, in terms of activities in the field, tactic properties, tactic evaluation rigor, and potential for industrial adoption.
Method: We leverage a systematic literature review based on a search query and two rounds of bi-directional snowballing. We identify 142 primary studies, reporting on 163 tactics, which we extract and analyze via a *mix* of qualitative and quantitative research methods.
Results: The research interest in the topic peaked in 2015 and then steadily declined. Tactics on source code static optimizations and application level dynamic monitoring are the most frequently studied. Industry involvement is limited. This potentially creates a vicious cycle in which practitioners cannot apply tactics due to low industrial relevance, and academic researchers struggle to increase the industrial relevance of their findings.
Conclusions: Despite the energy consumed by software is a growing concern, the future of energy efficiency tactics research does not look bright. From our results emerges a call for action, the need for academic researchers and industrial practitioners to join forces for creating real impact.

Keywords: Systematic Literature Review, Software Energy Efficiency, Tactics

1 Introduction

The energy consumption of software systems is an ever-increasing concern. Information and Communication Technology (ICT) consumes a staggering amount of electricity, estimated to produce between 2.1% and 3.9% of global greenhouse gas (GHG) emissions annually [7]. Data centers alone are estimated to produce 3% of the GHG, rivaling aviation at 2.5%, and having doubled in portion of global energy supply over the past 10 years [17].

Although hardware is the direct consumer of energy, software drives its energy consumption. As defined in a work by Jelschen *et al.* [14], which focuses on reengineering software to optimize its energy efficiency, at the highest level of abstraction of a computer system we find **application software**. Application software is the level of software that most software engineers address, namely the software developed for the end-users.Software engineers, now more than ever, must be aware of the energy consumption of the software they create, as the societal impact of their decisions can no longer be neglected.

A prime example of how application software drives energy consumption is google.com, Google's flagship search engine. Based on the latest figures officially released by Google, a search request on google.com consumes approximately 0,3 Wh.[3] Although this might seem like a negligible amount of energy, the search engine serves around 45,41 billion requests per month[4], *i.e.,* it consumes in total a staggering amount of 13,62 GWh per month. With this amount of energy, one could power 44.173 European homes per month[5]. Increasing the energy consumption of a search engine request by 0,1 Wh, would equate to driving the total monthly energy consumption up to 18,16 GWh per month, a 4,54 GWh increase. At such a massive scale, a seemingly negligible energy consumption increase at the application software level has the potential to increase the total energy consumption of a software application exponentially.

Despite the concerning global trends of software energy consumption, the literature on the topic is divided [27]. Software focusing on mobile and embedded devices received most attention [21], while application software received only a fraction of it [3, 27].

To worsen the situation, misleading biases and preconceptions often mentioned in the academic literature can confound readers. For example, a frequent yet wrongful assumption present in the literature is that software execution time and energy consumption are directly proportional, *i.e.,* that reducing execution time will reduce energy consumption [28]. This fact has been disproved multiple times in the recent literature [4, 19, 20]. Similarly, moving to the cloud or using green resources are often mentioned as holistic strategies to address software sustainability, while in reality such strategies do not provide any guarantee on the environmental sustainability of software applications [33].

By considering the state of practice, the outlook is also not promising. Most developers are not aware (yet) of the energy consumption of their software [26]. Developers that are aware of energy efficiency do not fully understand the energy consumption of their software, and lack concrete optimization examples and off-the-shelf solutions to address it [25, 33]. Developers who offer help on software energy optimization in popular knowledge bases tend to provide misinformed advice [30].

The **goal** of this paper is to identify existing energy efficiency application software tactics in the literature, in order to study the activity of the field, the characteristics of tactics, and their applicability in industrial settings. To achieve this goal, we use as **methodology** a systematic literature review [16] focusing on application software energy efficiency tactics. *Via* a mix of automated search and snowballing, we identify 142 primary studies (9 of which are extended versions). The primary studies are then analyzed by considering a classification framework comprising 9 parameters, such as execution environment, tactic type, abstraction level, and software development lifecycle (SDLC) stage.

The main **contributions** of this paper are:

- A rigorous *review* of the current tactics for application software energy optimization;
- A *data-driven framework* to classify application software energy optimization tactics;
- An assessment of the *industrial relevance* of energy efficiency software tactics.

The **audience** of this paper are (i) *researchers* interested in understanding the current state of the art of tactics for software energy efficiency and (ii) *practitioners* who consider applying tactics to improve energy efficiency of their software applications.

[3] `https://googleblog.blogspot.com/2009/01/powering-google-search.html`

[4] `https://www.statista.com/statistics/1201880/most-visited-websites-worldwide`

[5] `https://www.odyssee-mure.eu/publications/efficiency-by-sector/households/electricity-consumption-dwelling.html`

2 Related Work

While the literature includes several studies presenting reviews of software energy efficiency tactics, to the best of our knowledge, none of these studies specifically targets the level of application software. In addition, in none of the related literature the platforms, software development stage, and abstraction level of tactics is taken into account. Further considerations on the related work are reported below.

Frequently energy efficient software research considers embedded systems [14]. Such line of research focuses on low-level software and hardware optimizations, *e.g.,* dynamic voltage and frequency scaling, low energy power modes, hardware architecture techniques, and utilization of unconventional cores, *e.g.,* field-programmable gate array circuits [22].

One area closely related to application software is mobile computing, which has received plenty of attention in the academic body of literature. Zaman and Almusalli [36] provide a review of multiple energy efficiency tactics for smartphones at different system levels. However, the tactics they propose are mostly targeted towards tuning hardware components, low-level software, and operating systems to make them more energy efficient, without considering the application software domain. Naik and Chavan [24] focus on increasing the energy efficiency of smartphone hardware components *e.g.,* camera, and GPS. Similarly, Hans *et al.* [11] also present energy efficiency tactics for mobile applications. In both papers, the presented tactics focus exclusively on mobile applications, and are not applicable to devices that lack mobile specific hardware, *e.g.,* movement sensors.

The closest researches on reviewing energy efficiency application software tactics are the ones of Georgiou *et al.* [9] and Paradis *et al.* [27]. Georgiou *et al.* [9] present a review on techniques and tools to improve software energy efficiency. The authors describe different techniques and tools applicable at each development stage, and study the empirical evaluations conducted. In contrast to our work, the one of Georgiou *et al.* [9] (i) does not consider software abstraction levels, and (ii) does not consider the industrial relevance of tactics. Ignoring the abstraction level makes it harder for developers working at a specific level to select tactics. Presenting the existing empirical evaluations instead supports developers in understanding the efficacy and industrial applicability of tactics.

Paradis *et al.* [27] present the systematic literature review most closely related to our work. The authors perform a review of energy efficiency tactics, outlining strategies, and identifying potential issues for future work. There are two main differences w.r.t. this work. Firstly, similarly to the work of Georgiou *et al.* [9], Paradis *et al.* categorize tactics based solely on their definition. In this work, we classify tactics based on the context in which they can be applied, *e.g.,* abstraction level, target platform and SDLC. Secondly, the work of Paradis *et al.* differs in terms of number of primary studies considered. Paradis *et al.* review 39 primary papers, while this work is more encompassing, drawing results from 142 primary studies. In addition, Georgiu *et al.* do not report the individual tactic categorization, but rather a high level overview of each tactic type. Finally, the authors focus mostly on the verification stage of the SDLC, and do not conduct an systematic evaluation of the industrial relevance of tactics.

3 Definitions

3.1 System and Software Levels

In order to illustrate with care the tactics we target in this study, following we provide a brief overview of the system and software levels presented by Jelschen *et al.* [14]. Each level presents different opportunities to optimize the energy efficiency of software systems:

Hardware: Optimizations at this level focus on improving the hardware of computer systems to increase their energy efficiency by promoting better utilization of resources.
Low-Level Software: Optimizations at this level focus on improving the machine code transformations of source code. This is mainly done *via* compiler optimization.
Operating System: Optimizations at this level focus on improving the management of energy consumption by adjusting operating system functioning and settings, *e.g.,* by putting resources to sleep, or by scheduling resources.
Application Software: Optimizations at this level focus on improving the energy efficiency of software developed for the end-users, independently of operating systems and hardware capabilities. Such optimizations take into consideration application information that is unavailable at lower levels.

3.2 Energy Efficiency Optimization

We refer to the energy efficiency optimization of software as *green in software*, where the goal is reducing the energy consumption of software itself, and not *green by software*, where the goal is to use software to deliver energy efficient systems in other domains [5].

3.3 Platforms

In this paper, we consider the platforms targeted by application software. To avoid potential ambiguities, a definition of the different platforms, if any, considered by tactics is provided below. The definitions are based on the classification framework emerging from the coding process of this literature review.
Agnostic: Tactics that can be applied regardless of any platform, *e.g.,* the most energy efficient thread-safe data structure for the Java language.
Workstation: Tactics that can be applied on a single workstation such as a desktop or server, *e.g.,* measuring the energy consumption of the CPU of a single machine.
Distributed: Tactics that can be applied in a distributed setting, *e.g.,* cloud architectural patterns that reduce the energy consumption of cloud-native applications.
Mobile: Tactics that can be applied on mobile devices, *e.g.,* bundling sensor requests of an application to increase the energy consumption of the device.

3.4 Abstraction Levels

In this paper, we consider different abstraction levels of application software. This abstraction levels are defined by Buschmann *et al.* [1] as follows:
Architectural Level: The highest abstraction level. This level is concerned with overarching software components, layers, and their relation to the given context. Decisions at the architectural level express fundamental structural organization of software systems.
Design Level: Decisions at the design level are smaller in scope to those at the architectural level, but are at a higher level than programming language-specific idioms. The application of a design level decision has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem. Example are the "Gang of Four" design patterns [8].
Code Level: The lowest abstraction level. This level is concerned with implementation details at the source code granularity and describes how to implement particular aspects of components (and the relationships between them) with the features of programming languages. An example of this level of granularity are the refactoring techniques proposed in Martin Fowler's book, *Refactoring* [6].

3.5 Software Development Life Cycle

In this paper we categorize tactics based on the Software Development Life Cycle stages as defined by ISO-24748 [12], namely:

Requirements: Stage concerned with how requirements will be identified, traced, and managed.
Design: Stage concerned with defining, modeling, and describing the software system architecture and design.
Implementation: Stage concerned with how the various inputs into the software development effort will be implemented.
Verification: Stage concerned with how requirements, including non-technical requirements such as safety and security, will be verified and validated for the software system.
Maintenance: Stage concerned with how software defects and technical problems will be identified, recorded, and resolved.

4 Study Design

In this Section, we present the study design employed for this systematic literature review. We begin with our *Research Goal* and *Research Questions*, followed by complementary *Out of Focus Questions* to paint a clearer picture of the targeted focus. Next, we detail the employed *Search Strategy*, as well as the *Data Extraction* and *Data Synthesis* procedures, followed by our complementary *Study Replicability* package.

4.1 Research Goal

This study focuses on understanding the state of the art of energy efficient tactics, in terms of the activity of the field, the properties of tactics, the rigor of tactic evaluations, and their potential for industrial adoption. More specifically, by following the Goal-Question-Metric approach [2], our goal can be formalized as follows:

Analyze *tactics for software energy efficiency*
For the purpose of *classification and analysis*
With respect to *publication trends, properties, and potential for industrial adoption*
From the viewpoint of *software engineering researchers and practitioners*

4.2 Research Questions.

The main research question we address is:
RQ: *What is the state of the art of tactics for software energy efficiency?*
We refine the main research question in two research sub-questions:
RQ_1: *What are the characteristics of tactics for software energy efficiency?*
RQ_2: *Are tactics for software energy efficiency ready to be applied in industry?*

With respect to $\mathbf{RQ_1}$, as detailed in Section 2, characterizations of tactics for software energy efficiency targeted to several domains such as cloud architectures, embedded systems, hardware, low-level software already exist. Hence, in this study, we explicitly focus on tactics for software energy efficiency targeting application software, *i.e.,* tactics that can be deployed independently of the underlying hardware or operating system.

With respect to $\mathbf{RQ_2}$, we are interested in the potential applicability of tactics in an industrial setting. For a tactic to be applicable in practice, we need more than just empirical evaluations performed by researchers in a controlled environment; we need empirical evidence in real world settings to fully understand the prospective effects of each tactic.

4.3 Search Strategy

This study follows the literature search strategy shown in Fig. 1. The selected search strategy allows us to better control the characteristics and number of the potential primary papers at each stage. A description of each stage is presented below. The search strategy was executed by one researcher, while two other supervised the process and revised the results.

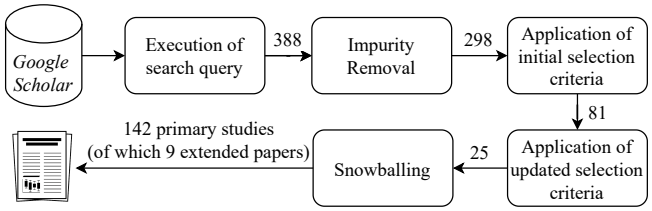

Fig. 1: Search Strategy (P: Primary Studies, S: Secondary studies, E: Extended)

Initial Search. For this investigation, we perform an initial automated search by leveraging the *Google Scholar* digital library. We opt to use Google Scholar based on multiple factors, namely (i) it has a vast aggregate of literature compiled across several publishers, *e.g.,* IEEE, ACM, Elsevier, Springer; (ii) systematic literature review guidelines suggest to use such digital library to conduct an initial automated search followed by a snowballing process [35]; (iii) it produces a higher yield of possible primary studies as opposed to other digital libraries, *e.g.,* IEEE Xplore, ACM Digital Library, Scopus; and (iv) query results can be automatically extracted.

The initial search is conducted by executing on Google Scholar the search query presented in Listing 1.1. The end date is set to the date the query is executed, namely *March 2022*. The start date is left unbounded, to mitigate potential threats to validity.

Listing 1.1: Google Scholar Search Query

```
1  TITLE: ("(power OR energy) (efficient OR efficiency OR consumption))" OR
       environmental OR green)
2  AND TITLE: (tactics OR strategies OR techniques OR tools OR patterns)
3  AND ("software (architecture OR development OR engineering)")
```

The query targets the keywords “power” or “energy” and “ efficient”, “efficiency”, or “consumption” in the title of the papers to identify studies focusing on energy efficiency. “Environmental” and “green” keywords are used to identify papers presenting energy efficiency tactics. The second query line targets the title keywords “tactics”, “strategies”, “techniques”, “tools”, and “patterns” to identify papers presenting software tactics. The design decision of utilizing such broad range of synonyms for the keyword “tactics” stems from the multitude of definitions present in academic literature regarding architectural tactics, and from the literature’s lack of a standardized characterization of architectural tactics [23]. Lastly we include the keywords “software” and “architecture”, “development” or “engineering”, searched throughout the full-text of the papers, to identify papers focusing on software engineering. We refrained to filter papers by using the keyword “application software”, in order not to exclude papers that targeted application level software under a more specific definition, like mobile application software or cloud computing. Instead, we assessed this criterion manually, by leveraging a specific inclusion criterion (I3, see Section 4.3).

The papers identified *via* the initial search could be in the form of primary studies, systematic literature reviews, systematic mapping studies, or loosely structured literature reviews. The purpose of including secondary studies in the automated query is to be as encompassing as possible, in order to lay a solid and comprehensive foundation for the subsequent snowballing process. While utilized for the snowballing, secondary studies are not considered for the data extraction, as further documented in Section 4.3.

Impurity Removal. After the 388 potential primary studies are obtained *via* the automated query, an impurity removal procedure is performed to filter out entries which

are not scientific peer-reviewed papers, *e.g.,* standards, patents, and master/doctoral theses. This procedure concludes with the identification of 298 papers.

Application of Selection Criteria. After the impurities are removed from the initial search results, we filter the remaining possible papers through a set of rigorous and *a priori* defined inclusion and exclusion criteria. A paper is included if it satisfied *all* of our inclusion criteria and *none* of the exclusion criteria. Several exclusion rounds are performed by first reading titles, then abstracts and conclusions, and finally a full reading of the paper, following and incremental reading depth [15]. Our inclusion (I) and exclusion (E) criteria are defined as follows:

I1- Studies focusing on the energy efficiency optimization. This criterion is used to include only studies focusing on the optimization of software energy efficiency.

I2- Studies presenting software tactics. This criterion is utilized to include only studies focusing on software tactics.

E1- Studies not focused on the perspective of a software engineer or software engineering researcher. This exclusion criteria ensures that tactics found are independently applicable by software engineers, without depending on external actors, *e.g.,* cloud providers or hardware manufacturers.

E2- Studies in the form of editorials, tutorial, short papers, and posters as they do not provide enough details for a thorough analysis.

E3- Studies that have not been published in English language, as their analysis is unfeasible in a timely manner without a translator specializing in software engineering.

E4- Studies that have not been peer reviewed, *e.g.,* pre-prints, technical reports, or gray literature, to ensure high quality of the considered papers.

E5- Duplicate or extensions of already included papers. When an extension of a paper is found, both papers are considered for the demographic analysis, but only the most mature version of the work is considered for data extraction.

E6- Papers that are not accessible, as, other than title and authors, we can not analyze the content of the paper.

After the application of the inclusion and exclusion criteria, we identify 81 papers which satisfy all inclusion criteria. However, we find that many papers target specific hardware and operating system, *e.g.,* embedded systems and wireless sensor networks. Therefore, we introduce a third inclusion criterion to further narrow down the selected papers exclusively to the application software level:

I3- Studies focused on application software, as defined by Jelschen *et al.* [14]. This inclusion criterion is utilized to select exclusively studies focusing on software developed for end-users, and is hence is applicable by a large number of software engineers.

Once this third criterion is applied, the number of primary studies resulting from the initial search amounts to 25.

Snowballing. To mitigate potential biases due to the search query used, and expand the primary study set, a recursive bidirectional snowballing procedure is adopted, till theoretical saturation is reached [35]. In total, two rounds of backward- and forward-snowballing are executed. The snowballing terminates with the inclusion of 133 additional studies.

Use of Secondary Studies for Snowballing. In this work, we design our automated query and selection criteria to include as many papers reporting secondary studies on tactics for software energy efficiency as possible. As discussed in Section 2, these literature reviews present a different focus w.r.t. this work. Nevertheless, given that these reviews fo-

cus on related subject matters, the reviews could accidentally capture primary studies containing tactics to be included in this review. This presents an opportunity to expand the search *via* a snowballing procedure. The secondary studies are exclusively used to enhance the snowballing process. In total, 15 secondary studies are identified. The complete list of secondary studies is documented in the replication package of this work (see Section 4.6).

4.4 Data Extraction

The purpose of the data extraction procedure is to create a classification framework for application software tactics, and to study the different facets of the tactics by following our classification framework. The classification framework takes into account *properties* and *industrial relevance* of the tactics, following the research questions proposed in Section 4.2.

Characteristics of Tactics for Software Energy Efficiency. Tactic characteristics can be separated into two groups: publication trends, and tactic properties. Publication trends help us visualize the state of the art in application software energy efficient research, and tactic properties helps us categorize each tactic based on specific criteria.
Publication Trends To identify the publication trends of tactics we evaluate three attributes, namely publication year, publication venue, and publication type.
Tactic properties To categorize types of tactics we employ a keywording process [29]. The keywording process consists of two steps, namely (i) collecting keywords from the full-text of primary studies *via* open coding, and combining keywords *via* constant comparison [10] to identify the context and nature of each tactic, and (ii) clustering of keywords into categories *via* axial coding to build a classification framework. The result of this process is a classification framework and the categorization of each tactic.

The parameters of the classification framework resulting from the keyroding are: Tactic Category, Execution Environment, Abstraction Level, Platform, and Software Development Stage. As one paper might present more than one tactic, the total number of tactics presented might be higher than the number of primary studies. Similarly, as a tactic can be mapped to more than one value of the classification framework parameters (*e.g.,*, a tactic might be mapped to more than one development stage), the total tactics per parameter might be higher than the number of tactics identified in the primary studies.

Potential for Industrial Adoption. To evaluate the potential for industrial adoption of each software tactic, we analyze the empirical evaluation of each primary study by applying the well-defined classification model introduced by Ivarsson *et al.* [13]. To objectively evaluate the rigor, the quality of the description of context, study design and execution, and validity of each study is analyzed. To evaluate industrial relevance, the description of the industrial context, subjects, application scale, and research method are considered, as further detailed in Section 6.

Use of Extended Papers. To limit potential conclusion bias, we do not consider for data extraction papers for which an extended version was found (see also Section 4.3). This process leads to the exclusion of 9 extended papers, resulting in a final set of 134 primary studies considered for the data extraction process. Note that, while not used for data extraction, extended papers are considered to study publication trends (see Section 5.1).

4.5 Data Synthesis

For our data synthesis procedure, we collate and summarize the data extracted to understand the current state of tactics for software energy efficiency [16]. In particular we use a combination of a descriptive synthesis (a descriptive analysis of the results) and content analysis (a categorization of results based on common characteristics).

4.6 Study Replicability

To ensure the replicability and scrutiny of this study, all raw and processed data resulting from each of the research phases is made available online.[6] In addition, we have included a reference table in the supplementary material of selected papers exemplifying each of the tactic groups discussed in Section 5.2.

5 Results RQ_1: Characteristics of Tactics for Software Energy Efficiency

In this section, we present the results corresponding to the characteristics of tactics for software energy efficiency, in terms of Publication Trends and Tactic Properties.

5.1 Publication Trends

Publication Year. An overview of the publication trends is presented in Fig. 2, where we can observe the rapid growth of publications from 2009, with a peak in 2015, and a steady decline in subsequent years. Caution needs to be used when interpreting the low number of publications in 2022, as the search query used for this study was executed in the beginning of such year, and hence the results might not be representative of the actual research output in 2022.

The publication years of the primary studies range from 2004 to March 2022. The earliest paper found was published in 2004. This paper focuses on the potential energy savings of offloading mobile computations to a cloud server compared to executing those computations locally. It is worth noticing that this paper considers mobile applications in the Android ecosystem, but does not directly mention application software, highlighting the dominance and maturity mobile computing has witnessed in application software energy efficiency research.

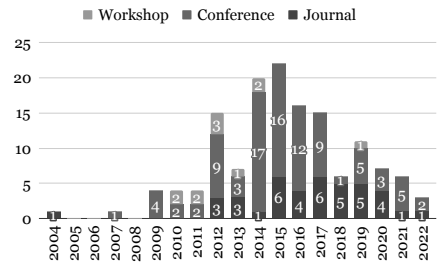

Fig. 2: Publication Trends

Publication Types. From Fig. 2 we can observe that the majority of the studies are published in conferences (91/142), while a substantial minority in journals (40/142), and only a handful in workshops (11/142).

Publication Venues. The publication venues where two or more primary studies were published are listed in Table 1. The venues with the most publications are the International Conference on Software Engineering (ICSE, 9 papers), followed by International Workshop on Green and Sustainable Software (GREENS, 5 papers). From the collected data a high variety of publication venues is observed, with only a small minority of venues presenting more than one publication on tactics for software energy efficiency (22/100).

5.2 Tactic Properties

In this section we present the properties the tactics for software energy efficiency identified in our review.

[6] https://github.com/ee-application-software/SEIS-2023-ee-application-tactics-rep-pkg

Venue	Venue	# of studies
IEEE/ACM International Conference on Software Engineering (ICSE)	C	9
International Workshop on Green and Sustainable Software (GREENS)	W	5
International Green and Sustainable Computing Conference (IGSC)	C	4
Information and Software Technology	J	4
IEEE Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER)	C	3
IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)	C	3
IT Professional	J	3
Journal of Systems and Software (JSS)	J	3
ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)	C	3
ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	C	3
Other	various	102

Table 1: Most popular venues (publications ≥3). W= Workshop, C= Conference, J= Journal

In total 163 tactics are identified from the primary studies. Out of the 134 studies considered (*i.e.,* by excluding the 9 extended papers for the data extraction), 24 proposed more than one tactic.

Execution Environment. The first parameter of tactics for software energy efficiency found through the keywording process is the **execution environment**. The execution environment of a tactic can be categorized as either ***dynamic***, if the tactic needs to be run during the execution of software applications, or ***static***, if the tactic needs to be executed during the development of software applications (*i.e.,* outside of its runtime environment).

Tactic Goal. The second parameter found during the keywording process is the **tactic goal**. The tactic goal can be either ***monitoring***, if the tactic allows developers to estimate the energy impact of their software, or ***optimization***, if the tactic consists of changing software characteristics (code, configurations, development environments) to improve energy efficiency.

Execution Environments and Tactic Goals. Supported by the parameters defined above (Sections 5.2 and 5.2) we classify tactics into four different groups, namely *dynamic monitoring*, *dynamic optimization*, *static monitoring*, and *static optimization*. An overview of the distribution of tactics among such groups is depicted in Fig 3, and is further characterized below.

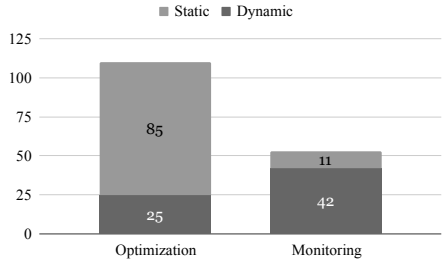

Fig. 3: Tactic Execution Environments and Goals

Static Optimization Static optimization focuses on improving energy efficiency of software outside runtime execution. This category represents the majority of tactics (85/163). Most tactics focus on the code level (43/85), followed by the architectural (26/85), and the design one (24/85). At the code level, tactics focus on data-structure implementation, and/or low granularity code optimizations (*e.g.,* control flow changes). At the architectural level, static optimization focuses on improving high level design decisions, *e.g.,* using of the most energy efficient programming language, software libraries, and development environments. At the design level, tactics focus on modifying existing design patterns, or selecting the most appropriate concurrency constructs. Regarding the software development stages, the majority tactics are performed during the implementa-

tion stage (37/85), followed by the maintenance stage (26/85), the design stage (23/85), the requirements stage (6/85), and the verification stage (1/85). Tactics at the design and implementation stages include novel programming practices, such as the use of genetic algorithms, search based optimizations, and evolutionary computing. Tactics at the requirement stage propose to add energy efficiency as quality attribute, and suggest how to balance it with other functional and quality attributes. Tactics at the verification stage propose improving testing techniques to make them more energy efficient.

Dynamic Optimization This group of tactics focuses on optimizing the energy efficiency of software at runtime. This group contains a smaller fraction of tactics (25/163) if compared to the static optimizations (85/163). Regarding the abstraction level of dynamic optimizations, the majority of tactics are employed at the architectural level (13/25), followed by the code level (11/25), and the design one (1/25). Tactics at the architectural level focus on dynamically choosing runtime environments for software components. Tactics at the code level self-adapt software components, *e.g.,* data structures, to increase energy efficiency based on runtime measurements. Tactics at the design level allow developers to change design elements during execution depending on runtime information, such, *e.g.,* input size. Regarding software development stages, these tactics only appear at the design stage (14/25) or implementation stage (12/25).

Dynamic Monitoring This group of tactics focuses on monitoring application software at runtime to estimate its energy consumption. It is the largest group of monitoring tactics (42/163). Tactics in this group most commonly measure entire applications (25/42), followed by code sections (14/42), and design elements (3/42). Across development stages, since dynamic measuring is performed by measuring the energy consumption of application software at runtime, all instances occur at the verification stage (42/42).

Static Monitoring This group of tactics focuses on measuring the energy consumption of application software outside of its runtime *via* static analysis and/or energy models. Only a handful of tactics are proposed to monitor applications based on static source code analysis (11/163). Analyses of this group can use information previously collected at runtime, *e.g.,* the runtime data upon which an energy model is built. Nevertheless, these tactics are exclusively executed during development stage, *e.g.,* by showing developers the energy consumption of lines of code *via* IDE extensions using an energy model. Regarding the considered abstraction level, static monitoring tactics mostly consider the code level (7/11), followed by the architectural (3/11) and the design one (1/11). Regarding the software development stages, most tactics are used at the verification stage (9/11), while only few at the design stage (2/11).

Abstraction Level. Regarding the abstraction level considered by the tactics identified in our review, an overview of their distribution is depicted in Fig 4. As can be observed in the figure, most tactics consider the code level (76/163), followed by the architectural (67/163), and the design one (29/163).

Software Development Stage. An overview of the software development stages of the tactics is depicted in Fig 5. Most tactics result to be employed at the verification stage (51/163), followed by the implementation (50/163), design (39/163), and maintenance stage (26/163). Energy efficient tactics to be used during the requirements stage appear to be only marginally explored (6/163).[7]

Platform. An overview of the platforms considered by the tactics is depicted in Fig. 6. Most tactics result to be either platform agnostic (68/163), or targeting mobile

[7] The sum of tactics across development stages is higher than 163, as some tactics are mapped to more than one development stage.

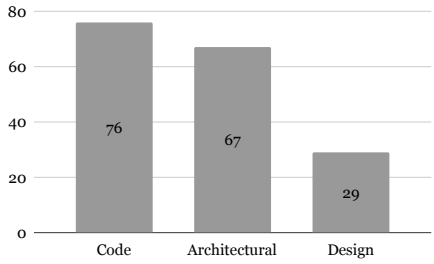

Fig. 4: Tactic Abstraction Levels

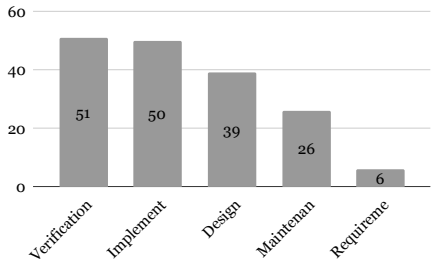

Fig. 5: Software Development Stages

computing (67/163). Only a smaller number of tactics regard cloud environments (17/163) and single workstations (11/163).

Main findings RQ1: *Energy Efficiency Tactic Characteristics*

- Starting from 2004, publications on energy efficiency tactics reached a peak of popularity in 2015, and then steadily declined.
- *Static optimization* tactics are most frequent, followed by *dynamic monitoring* tactics.
- The majority of tactics considers either the source code or architectural level, and only a smaller portion the design level.
- Most tactics can be used during the verification or implementation stage. Fewer tactics during the design or maintenance stage.
- Most tactics are platform agnostic or target mobile computing. Much fewer exist for cloud and workstation environments.

6 Results RQ_2: Potential for industrial adoption

In this section we document the potential for industrial adoption of the tactics identified in this review.

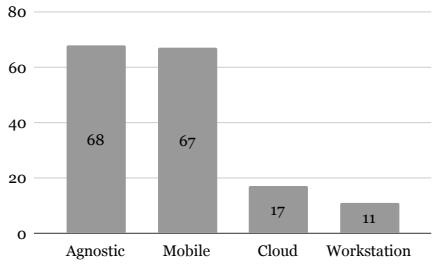

Fig. 6: Tactic Targeted Platforms

Industrial Involvement. To evaluate the industrial involvement in tactics for software energy efficiency, we leverage three categories: ***academic*** (if all the authors are affiliated to academia), ***industrial*** (if all authors are affiliated to industry), and ***mixed*** (if co-authors are from both academia and industry).

Primary studies produced exclusively by academic authors are the vast majority (119/134), while only a handful of papers have both academic and industrial authors (12/134), and few exclusively industrial authors (3/134).

Rigor and Industrial Relevance. To assess the readiness for industrial adoption of the software tactics, we analyze the evaluation of each primary study based on *rigor* and *industrial relevance* as defined by Ivarsson *et al.* [13] (see also Section 4.4).

Rigor [13] is defined as the precision of the research approach and its documentation. It is measured *via* three parameters, namely: (i) ***context***, *i.e.,* how well the context is presented, and if its description is sufficient to make objective comparisons with other

contexts, (ii) ***study design***, *i.e.,* the products, resources and processes used in the evaluation, and (iii) ***validity***, *i.e.,* any limitations or threats to the validity of the evaluation and the measures taken to limit them. The three parameters are measured with the values "*weak*", "*medium*", and "*strong*". We also include the "*no experiment*" value to categorize the studies not including any evaluation.

Industrial relevance [13] is defined as the realism of the environment in which the results are obtained and the research method used to produce results. It is measured by utilizing four parameters: (i) ***subjects***, *i.e.,* the subjects used in the evaluation, (ii) ***context***, *i.e.,* the context in which results are obtained, (iii) ***scale*** *i.e.,* the type of application used in the evaluation, and (iv) ***research method***, *i.e.,* the research method used, *e.g.,* laboratory experiments or case studies.The four parameters are measured with two values, "*contributing*" and "*non-contributing*", each representing whether the characteristic under scrutiny contributes to industrial relevance or not. To limit conclusion bias, we introduce a "*no experiment*" value to categorize papers that do not perform any type of evaluation.

Rigor. An overview of the primary study rigor is depicted in Fig. 8. Out of all primary studies, only a small portion does not report any type of tactic evaluation (7/134).

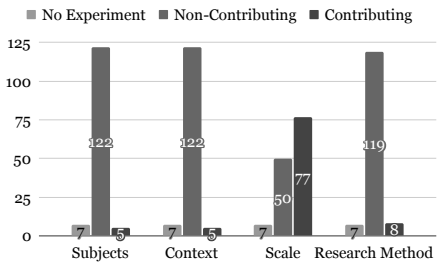

Fig. 7: Industrial Relevance

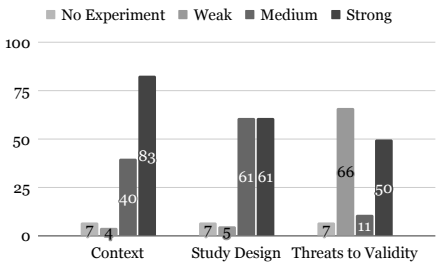

Fig. 8: Rigor

Regarding the *context*, most studies accurately describe the context (83/134). A smaller portion reports context with insufficient details to allow any comparison with other contexts (40/134), while only few papers do not describe the context at all (4/134).

Regarding the *study design*, primary studies present a medium or strong study design. The majority describe the study design to an extent to which the reader can compare the results to other studies (61/134), or mention the steps taken, but do not allow the reader to understand the measurements / statistical analysis performed (61/134). Only few studies do not mention the study design at all (5/134).

Regarding *validity*, studies result to be either a hit or miss. The majority does not discuss any threat to validity (66/134), while a similar number thoroughly describes them (50/134). Few papers mention threats without properly documenting them (11/134).

Industrial Relevance. Fig. 7 paints a different picture when considering the industrial relevance of primary studies. Regarding *subjects*, the vast majority of studies presents evaluations where the researchers are the subjects applying the tactics (122/134). Only few papers present instead evaluations where the subjects applying the tactics are industrial practitioners (5/134). Regarding *context*, the vast majority of papers presents

evaluations performed in laboratory settings or controlled environments (122/134). Only few studies consider a real-world industrial setting (5/134).

The industrial relevance category showcasing the highest number of "contributing" studies is *scale*. Most papers present evaluations on real world applications (76/134), while only a smaller portion utilizes *ad hoc* made software, mock ups, or benchmarks (50/134).

Finally, regarding the *research method* used, most primary studies use laboratory experiments and theoretical mathematical analyses (119/134), while only a much smaller fraction conducts evaluations using a research method that facilitate the investigation of real situations such as case studies and field studies (8/134).

Main findings RQ_2: *Potential for industrial adoption.*

- The field showcases a very limited industry involvement.
- The rigor used to evaluate tactics is usually sound. Most studies document with care their context and study design, albeit threats to validity of evaluations are often not discussed.
- Industrial relevance of tactics is scarce. Most tactics are developed and applied by academic researchers; they consider *in vitro* contexts, and use controlled experiments. Adequate scale of evaluations is mostly driven by the use of open source projects.

7 Discussion

The results collected *via* this systematic literature review provide a clear picture of the tactics for software energy efficiency landscape.
The topic began to become attractive in 2009 and, after reaching a peak of popularity in 2015, experienced a loss of interest (see Section 5.1). Multiple conjectures can be made on this trend, but the culmination of software energy efficiency tactics cannot be one of them. To date the topic still displays vast improvement possibilities, and is characterized by only marginally-explored areas (*e.g.,* tactics for cloud and edge environments [32]). An explanation could be the increasing complexity and fragmentation software applications experienced during the years, making tactics harder to be designed and evaluated. Another explanation could be the lack of a consolidated research foundation driving the topic. This conjecture might be further corroborated by the scattered publication venues (see Section 5.1), potentially indicating a missing unified research effort.

Regarding tactic properties (see Section 5.2), the high occurrence of static optimization and dynamic monitoring tactics might be due to the relative ease of developing such tactics, if compared to dynamic optimization and static monitoring ones. The trend towards focusing on *low-hanging fruits* is further supported by the most common abstraction levels used. By considering static optimizations, the vast majority focuses on code optimizations, *i.e.,* regard isolated adjustments which do not regard other software components or dependencies. Similarly, most dynamic monitoring tactics focus on monitoring entire applications, *i.e.,* do not require additional analyses to identify which software elements, or sections of code, are more energy greedy.

From the development stages of tactics (see Section 5.2) we can observe how their distribution is driven by the most recurrent tactic types. The high number of tactics utilizable during the implementation stage is primarily driven by dynamic monitoring tactics. Similarly, static optimizations mostly contribute to the implementation stage tactics.

From the distribution of tactics across platforms (see Section 5.2), most tactics are either platform agnostic or mobile-centric. This trend can be attributed to the emphasis on energy efficiency in mobile contexts. With the adoption of open source operating systems, *e.g.,* Android, and mobile hardware advancements, mobile platforms are

becoming evermore similar to workstations. Therefore, the majority of tactics are either platform agnostic (*i.e.,* can be applied across platforms), or consider mobile software. The low number of cloud tactics could be driven by the relatively recent consolidation of such platform, for which we expect a growing number of tactics in the future.

From the results of RQ_2 emerges that, while the rigor of tactic evaluation is high, a prominent lack of industrial involvement is present. The topic remains primarily an academic interest, with industrial parties displaying little investment. Such trend is reflected in tactic evaluation, which, due to the low representativeness of their subjects, contexts, and research methods, possess little industrial relevance. This creates a vicious cycle, in which academic efforts are hindered by the lack of industrial involvement, and industry cannot adopt academic solutions due to the low representativeness of the solutions. In order to break such vicious cycle, as suggested in recent literature [33], policy makers need to steer the way *via* regulations to make software sustainability a primary concern of industry. As a first possible step toward engaging practitioners, recently an open-source and open-access archive of tactics was made available online [18].

8 Threats to Validity

Despite our best efforts, the presented results might have been influenced by threats to validity. Following the threat classification of Runeson *et al.* [31], we discuss four aspects.

Construct validity. Our results could be influenced by the search query and digital library used. As mitigation strategy, we used a bi-directional exhaustive snowballing. To mitigate threats related to the literature selection and data extraction processes, we adopted consolidated guidelines for literature reviews [16] and snowballing [35], and used a systematic evaluation process to judge rigor and industrial relevance of tactics [13].

Internal validity. The identification of primary studies, the data extraction, and keywording processes might have been influenced by subjective interpretations, and hence prone to biases. Such threat might be remarked by the fact that the first author of this study was primarily responsible for such processes. To mitigate related threats, all steps were supervised by two other researchers, and any doubt or impediment was jointly discussed. In addition, all intermediate and final data was scrutinized and reviewed by all three researchers. Finally, the scientific quality of the primary studies could have influenced the results of the study. As done in similar work [34], rather than conducting a manual, and potentially subjective, quality assessment, we opted to include only primary studies published in venues employing peer-review processes. Therefore, we do not deem the quality of the primary studies to have noticeably influenced the results.

External validity. A common threat to external validity of SLRs is the representativeness of the identified literature. To mitigate related threats, our search query was purposely designed as encompassing as possible, leaving the identification of relevant studies up to a higher effort required for the manually scrutiny. Augmented *via* bi-directional snowballing, our final dataset comprised 142 primary studies, and acknowledged the presence of 15 secondary studies. While not claiming to be complete, we conjecture that the identified literature is representative of the current state of the art.

Reliability. To promote reliability, all intermediate and final data of our the literature selection, data extraction, and data analysis is made available online (see Section 4.6).

9 Conclusion

In this paper, we aim at characterizing the state of the art of tactics for software energy efficiency. To achieve our goal, we conducted a rigorous systematic literature review, leading to the identification of 142 primary studies and a total of 163 tactics.

Despite the energy consumed by software systems is a growing concern, the state of the research area, as studied through the lens of the literature, is not bright. The topic reached considerable popularity in 2015, but steadily declined afterwards. Potentially driven by analysis ease, most tactics focus on static optimization or dynamic monitoring, and consider either the energy consumed by entire applications or lines of code. Industry involvement is scarce, as reflected by the low industrial relevance of tactics. Tactics result to be, to date, mostly an academic concern, which is not transportable to practice.

From the results emerges a call for action to academic researchers and industrial practitioners to join forces and study how software sustainability can be improved – needed now more than ever before. Only by joining forces, by firmly bridging the current gap between academic research and industrial adoption, and by merging the current academic fragmentation, can the sustainability of software really be addressed.

References

1. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, Volume 1, vol. 1. John wiley & sons (2008)
2. Caldiera, V.R.B.G., Rombach, H.D.: The goal question metric approach. Encyclopedia of software engineering pp. 528–532 (1994)
3. Capra, E., Francalanci, C., Slaughter, S.A.: Is software "green"? application development environments and energy efficiency in open source applications. Information and Software Technology **54**(1), 60–71 (2012)
4. Capra, E., Francalanci, C., Slaughter, S.A.: Is software "green"? application development environments and energy efficiency in open source applications. Information and Software Technology **54**(1), 60–71 (2012)
5. Condori Fernandez, N., Lago, P.: The influence of green strategies design onto quality requirements prioritization. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 189–205. Springer (2018)
6. Fowler, M.: Refactoring. Addison-Wesley Professional (2018)
7. Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G.S., Friday, A.: The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. Patterns **2**(9) (2021)
8. Gamma, E., Helm, R., Johnson, R., Johnson, R.E., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH (1995)
9. Georgiou, S., Rizou, S., Spinellis, D.: Software development lifecycle for energy efficiency: techniques and tools. ACM Computing Surveys (CSUR) **52**(4), 1–33 (2019)
10. Glaser, B.G., Strauss, A.L.: The discovery of grounded theory: Strategies for qualitative research. Routledge (2017)
11. Hans, R., Burgstahler, D., Mueller, A., Zahn, M., Stingl, D.: Knowledge for a longer life: development impetus for energy-efficient smartphone applications. In: 2015 IEEE International Conference on Mobile Services, pp. 128–133. IEEE (2015)
12. ISO/IEC/IEEE: International Standard - Systems and Software Engineering–Life Cycle Management–Part 5: Software Development Planning. IEEE (2017)
13. Ivarsson, M., Gorschek, T.: A method for evaluating rigor and industrial relevance of technology evaluations. Empirical Software Engineering **16**(3), 365–395 (2011)
14. Jelschen, J., Gottschalk, M., Josefiok, M., Pitu, C., Winter, A.: Towards applying reengineering services to energy-efficient applications. In: 2012 16th European Conference on Software Maintenance and Reengineering, pp. 353–358. IEEE (2012)
15. Keshav, S.: How to read a paper. ACM SIGCOMM Computer Communication Review **37**(3), 83–84 (2007)
16. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Engineering (2007)
17. Knowles, B.: ACM TechBrief: Computing and climate change (2021)

18. Lago, P.: An Archive of Awesome and Dark Tactics. `https://s2group.cs.vu.nl/AwesomeAndDarkTactics/` (2022)
19. Li, D., Hao, S., Gui, J., Halfond, W.G.: An empirical study of the energy consumption of android applications. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 121–130. IEEE (2014)
20. Lima, L.G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., Fernandes, J.P.: Haskell in green land: Analyzing the energy behavior of a purely functional language. In: International conference on Software Analysis, Evolution, and Reengineering, vol. 1. IEEE (2016)
21. Manotas, I., Pollock, L., Clause, J.: Seeds: A software engineer's energy-optimization decision support framework. In: International Conference on Software Engineering (2014)
22. Mittal, S.: A survey of techniques for improving energy efficiency in embedded computing systems. International Journal of Computer Aided Engineering and Technology **6**(4), 440–459 (2014)
23. Márquez, G., Astudillo, H., Kazman, R.: Architectural tactics in software architecture: A systematic mapping study. Journal of Systems and Software **197**, 111,558 (2023)
24. Naik, B.A., Chavan, R.K.: Optimization in power usage of smartphones. International Journal of Computer Applications **119**(18) (2015)
25. Ournani, Z., Rouvoy, R., Rust, P., Penhoat, J.: On reducing the energy consumption of software: From hurdles to requirements. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (2020)
26. Pang, C., Hindle, A., Adams, B., Hassan, A.E.: What do programmers know about the energy consumption of software? PeerJ PrePrints **3** (2015)
27. Paradis, C., Kazman, R., Tamburri, D.A.: Architectural tactics for energy efficiency: Review of the literature and research roadmap. Hawaii International Conference on System Science (2021)
28. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Energy efficiency across programming languages: how do energy, time, and memory relate? In: ACM SIGPLAN International Conference on Software Language Engineering (2017)
29. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: International Conference on Evaluation and Assessment in Software Engineering (2008)
30. Pinto, G., Castor, F., Liu, Y.D.: Mining questions about software energy consumption. In: Proceedings of the 11th Working Conference on Mining Software Repositories (2014)
31. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering **14**(2), 131–164 (2009)
32. Toczé, K., Madon, M., Garcia, M., Lago, P.: The Dark Side of Cloud and Edge Computing: An Exploratory Study. In: Workshop on Computing within Limits 2022. LIMITS (2022)
33. Verdecchia, R., Lago, P., de Vries, C.: The future of sustainable digital infrastructures: A landscape of solutions, adoption factors, impediments, open problems, and scenarios. Sustainable Computing: Informatics and Systems p. 100767 (2022)
34. Verdecchia, R., Ricchiuti, F., Hankel, A., Lago, P., Procaccianti, G.: Green ICT Research and Challenges. In: Advances and New Trends in Environmental Informatics, pp. 37–48. Springer International Publishing (2017)
35. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: International conference on evaluation and assessment in software engineering (2014)
36. Zaman, N., Almusalli, F.A.: Smartphones power consumption & energy saving techniques. In: International Conference on Innovations in Electrical Engineering and Computational Technologies. IEEE (2017)