A Model-based Framework for the Analysis of Software Energy Consumption

Lucio Mauro Duarte, Danilo da Silva Alves, Bruno Ramos Toresan
{lmduarte,dsalves,brtoresan}@inf.ufrgs.br
Institute of Informatics, UFRGS
Porto Alegre, RS, Brazil

Paulo Henrique Maia, Davi Silva
pauloh.maia@uece.br,jose.davi@aluno.uece.br
Group of Distributed Software Engineering, UECE
Fortaleza, CE, Brazil

ABSTRACT

Software is present in all types of devices, some of them with restrictions as to the amount of energy they can spend to execute software applications. For this reason, energy costs are becoming an important factor during software development and evolution. However, there is still little support for creating energy-efficient software. In this work, we introduce a possible framework for software energy costs evaluation based on model analysis. We model software as Labelled Transitions Systems (LTS) and annotate these models with energy costs, which can be obtained using existing tools. We can then apply graph-based algorithms to traverse the models to obtain information about energy consumption related to software behaviour, such as its most/least costly execution, the cost of a specific execution, and the average cost of executing the software. No existing tool currently provides all the necessary analyses, even though they are essential for energy-consumption evaluation. We have conducted a small experiment with our framework where we employed jRAPL to measure energy costs. We annotated the models with the collected energy costs using an extended version of the LoTuS tool, where we have also implemented some of the desired analyses. Based on this support and on our initial results, we believe developers could create software more energy-efficient and consider possible trade-offs related to time, space, and energy costs when producing new versions of their systems.

KEYWORDS

Model-based Analysis, Energy Consumption Evaluation, Labelled Transition System, Embedded Software

ACM Reference Format:
Lucio Mauro Duarte, Danilo da Silva Alves, Bruno Ramos Toresan and Paulo Henrique Maia, Davi Silva. 2019. A Model-based Framework for the Analysis of Software Energy Consumption. In *XXXIII Brazilian Symposium on Software Engineering (SBES 2019), September 23–27, 2019, Salvador, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3350768.3353813

SBES 2019, September 23–27, 2019, Salvador, BA, Brazil

ACM ISBN 978-1-4503-7651-8/19/09…$15.00
https://doi.org/10.1145/3350768.3353813

1 INTRODUCTION

Software is ubiquitous to all types of devices, some of them with restrictions as to the amount of energy they can spend to execute software applications. Mobile applications that quickly drain battery energy tend to be rejected by users [18], whereas corporations have also come to the conclusion that small inefficiencies in software can significantly affect its operation [30]. For this reason, energy consumption is now an important factor during software development and evolution [1] [21] [2] [32].

Despite the current importance of energy consumption analyses, there is still little support for designing energy-efficient software. In fact, developers find it still unclear how to produce and evolve energy-efficient software [30], mainly due to the absence of combined abstractions and tools to collect, model, and analyse energy consumption. Although there are some existing tools [23] [31] that allow the measurement of energy costs associated to code locations (this information is useful for identifying possible hotspots of energy consumption), developers still need to understand how such tools can be used to increase energy efficiency and how the identified hotspots affect the overall energy consumption of their software. Having energy information, analyses can be carried out using a model checking tool such as PRISM [20], as described in [3] [11], where a Markov chain can be used to model software considering probabilistic/stochastic behaviour, transition/state costs could be used to model energy costs, and questions about accumulated energy costs can be asked using a probabilistic temporal logic. Nonetheless, there is no possibility of visualising the model in the tool and the user still needs to gather the energy information to be used in the model. Moreover, knowledge on the logic used in the tool is required to ask the appropriate questions, which can only be about states (either the probability of being in a certain state or the accumulated energy cost when reaching a given state). However, many interesting questions are related to traces, such as the most costly execution and the average execution cost. These questions are not all covered by any current tool.

Considering all the ideas previously discussed, there is a need for a structured and well-defined idea of how to use and integrate available tools and how to provide the yet missing support some necessary analyses. In this work, we propose a model-based framework for analysing software energy consumption through the verification of energy-based properties, thus providing a novel approach for the development and evolution of energy-efficient software. We use *Labelled Transition Systems (LTS)* [17] as an abstraction to model software behaviour. Because an LTS is essentially a graph-like structure, we can apply graph-based algorithms to traverse the

LTS to obtain information about the software behaviour. Having energy costs associated to code elements, we could use these algorithms to also analyse energy consumption information, such as the software most/least costly execution and its average execution cost. No existing tool alone currently provides all these analyses and the verification of other properties of interest, even though they are essential for energy-consumption evaluation. Moreover, there is no guide as to how to develop software using available tools or a combination of them.

We present preliminary results where we instantiated our framework using jRAPL [23] to obtain energy costs associated to code elements of a simple program. As for the modelling and analysis, we have extended an existing free, open source LTS-analysis tool called LoTuS[1] [4] to include energy cost annotations and implement some of the desired analyses. Our results for this small experiment show this framework can produce information a developer could use to understand the energy consumption of their software and improve its energy efficiency. Moreover, the framework is flexible enough to support the combination of other tools that can provide the necessary information and analyses. Hence, each developer could follow the ideas described in each step of the framework to get the most out of their energy analysis using the tools at hand, as long as they give the appropriate support. Since our approach is compositional, the same process applied to the example program could be used for each component of a larger system, producing a model for each one of them, thus enabling support to multi-component systems and allowing further types of analyses.

The remainder of this document is divided as follows: Section 2 discusses the main related work; Section 3 introduces the proposed model-based framework, while Section 4 presents an example of instantiation and use of the framework; Section 5 brings some possible application areas of the framework; finally, we conclude and outline future research directions.

2 RELATED WORK

Software energy consumption research has so far been focused on measuring energy costs. The work described in [29] concentrated on detecting excessive or anomalous energy consumption in software, whereas the authors in [32] focused on optimising the energy consumption in IT resources knowing how much power an application is consuming. Further, the research presented in [31] described the development process of a profiler for measuring the energy consumption of source code points. In [28], critical energy areas were identified using a statistical method to associate responsibility for energy consumption to source code components. However, there is no support for understanding how this consumption affects the software behaviour as a whole. Analyses have been conducted on the influence of data structures on energy consumption [29] [14] [27], introducing a methodology to optimise Java programs and decrease its energy consumption replacing data structures for a more energy-efficient alternative. There have also been studies focusing on estimating energy costs with the goal of extending battery life in embedded systems [7] [16] and mobile devices [13] [24]. Nevertheless, none of the above studies employs a model as basis for their analyses, which limits their analysis capacity, since abstractions enable analyses that could not be or would be very difficult to be carried out directly on the actual software.

Some other approaches, e.g. [3] [11] model energy costs using Markov chains and use PRISM [20] to run quantitative analyses based on probabilistic information. The approach described in [3] uses the costs/rewards feature of PRISM to assigns costs to states/transitions. In [25], Power Consumption Automata are used to model the software and properties are written in terms of weighted linear temporal logic, which are checked against the automata, but no results are presented. The proposed framework works with execution traces and includes the collection of energy information to be added to the model. Moreover, model visualisation is an important feature. Even though PRISM can export models to be visualised using a graph description tool, such as GraphViz[2], the problem is that using two different tools to create and visualise models, in our opinion, makes it more complicated for developers to manually create their models and edit them more easily to produce the desired result. Another limitation of the aforementioned approaches is that analyses about paths are not supported. As PRISM uses Markov chains as underlying model, it carries out analyses based on probabilities and cost/rewards considering properties such as the probability of reaching a certain state, accumulated cost/reward of visiting a state, etc. Hence, questions that either require producing sequences of actions or evaluating specific executions (e.g., the most/least costly behaviour) could not be easily executed, if at all.

The framework proposed in this work describes how to combine gathering, modelling, and the analysis of software energy consumption information. We believe that any approach for analysing energy costs should contemplate all the described parts, which unfortunately is not the case of any of the aforementioned related work. In the development of our experiments, discussed in Section 4, a possible instantiation of the framework is carried out and shows how each part plays an important role in the process.

3 THE PROPOSED FRAMEWORK

This work proposes a model-based framework for analysing energy costs in software. It is divided into four phases, as depicted by Figure 1: (i) behaviour model specification; (ii) energy consumption measurement; (iii) model annotation; (iv) energy-based property verification. Each phase is detailed as follows.

3.1 Behaviour model specification

Initially, the user needs to create a behaviour model of their software. We propose our framework for a compositional development, where each component of the system is modelled and analysed in isolation, thus reducing modelling and analysis complexity and facilitating adjustments to achieve energy efficiency.

We propose Labelled Transition Systems (LTS) [17] to model software behaviour. LTS models are similar to state machines, which makes it easier for any developer to intuitively construct/understand them. They describe systems in terms of their possible states and transitions between these states, where these transitions are labelled with action names. These action names can be used to represent the execution of an element of the code for which we would like to

[1] gesad.uece.br/lotus

[2] https://www.graphviz.org

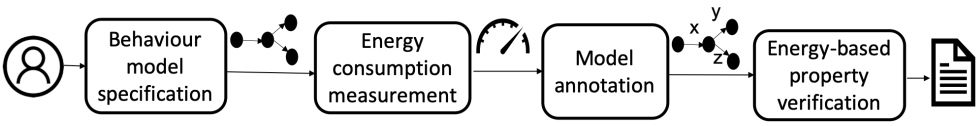

Figure 1: Proposed model-based framework

obtain an energy cost (e.g., a method call). Transitions of the model are annotated with the energy costs associated to each action.

Model construction can be carried out by manually relying on the knowledge of the system analyst or by using a specific model extraction approach, such as [10]. Each transition should represent the occurrence of a well-defined element of the code, which is associated to the action name labelling the transition and whose energy cost should be measured (method calls, for instance). Each state of the model, therefore, corresponds to a point where execution has not yet reached the next element of interest. A particular behaviour is, then, a sequence of actions allowed in the model, and its corresponding energy consumption is the sum of all costs labelling transitions of the behaviour path in the model.

3.2 Energy consumption measurement

In this phase, the energy consumption of the system under analysis is measured. It can be done using tools that model and/or simulate microarchitectures, such as Gem5 [5] and McPAT [22], or jRAPL [23], which collects energy information from Java programs. Whereas McPAT and Gem5 use abstractions to obtain estimates of energy information, jRAPL allows the annotation of source code with methods to collect energy information, so that the user can select what parts of the code should monitored for energy usage. Whatever tool is used, it should also be able to calculate the energy costs associated to the code elements used as transition labels in the model created in the previous phase. Collecting energy consumption information involves either running some simulations or executing the program a number of times.

3.3 Model annotation

The next phase consists of augmenting the system behaviour model with the energy consumption information of each element collected in the previous phase. This annotation can be performed automatically, i.e., using a script or intermediate solution that maps the system code to its behaviour model, or manually introduced by the user after the measurements. It can either be performed at runtime (in this case the costs may vary with time) or after the execution of the system. At the end of this phase, we obtain an energy-annotated LTS, where there is an energy cost associated to each transition of the model. After this phase, we end up with a model that is basically a graph (states and transitions) with weights (energy costs). Hence, graph-theory concepts and algorithms can be applied.

3.4 Energy-based property verification

The final phase comprises the specification of energy-based properties and their verification against the energy-annotated model produced in the previous phase. The types of properties we are interested in are presented in Table 1.

Those properties cover the most important types of energy consumption analysis, allowing users to choose and specify different properties according to their need. For instance, considering the list of properties presented in Table 1, the first property helps the user identify unitary costs per execution, while the second one gives them a broader scenario of the system energy consumption. The third property allows checking whether an execution satisfies energy-related non-functional requirements. The fourth property aids the user to find executions that may not be in conformance with expected energy costs or that are above the average consumption, which is calculated and informed in the fifth property. Even though some of these types of properties could be verified in existing tools, such as PRISM, no tool currently supports all them.

The overall time to analyse a real-world project using this framework depends on the type and size of the project and on the knowledge of the developer on using the model and tools at each stage. The bigger the software, the longer it may take to both model the system behaviour and collect energy consumption information. It is, therefore, desirable that all phases have tool support, so that user participation is restricted to providing inputs (i.e., selecting the code and its elements to be analysed and choosing the properties to be verified) and analysis of the results to improve the code, if necessary. This can be accomplished by combining multiple tools or using a single tool, which currently does not exist, as discussed in Section 2. Our idea is that the framework be flexible enough to be instantiated differently according to the user needs and tools at hand. Nevertheless, we believe that all phases should be carried out as described to obtain the results a developer can use to identify, and possibly fix, energy problems in their software.

4 INITIAL RESULTS

In this section we present an instantiation of the proposed framework and initial results of its application to a real system. An *instantiation* of the framework corresponds to choosing techniques/tools to implement each phase of the framework.

For our example, in the first phase, we used the open-source modelling tool LoTuS, which allows users to graphically specify both probabilistic and non-probabilistic behaviour models. For the energy consumption measurements, we used jRAPL. As mentioned before, jRAPL allows the annotation of Java source code to collect energy information at specific code locations. Moreover, it has already been used in other software projects, with minimum required knowledge on hardware aspects to be applicable. For the annotation phase, we extended LoTuS so that energy costs related to code elements could be added as transition labels along with action names that described these elements. Finally, as there is no tool that provides support for the analysis of all the necessary energy-based

Property List			
Prop.	Property Name	Input	Output
P1	Cost of an execution	Initial and final state/action	Energy cost value
P2	Cost of all executions	None	List of all possible executions ordered decreasingly by energy total cost
P3	Energy-consumption verification	An execution and a boolean expression	`true` or `false`, according to the satisfaction or not, respectively, of the boolean expression by the execution cost value
P4	List of executions with a threshold	Inferior and/or superior energy thresholds	List of executions satisfying the criteria
P5	Statistical properties	None	Average energy value of all executions, standard deviation and variation

Table 1: Energy-based properties

properties, we again extended LoTuS to allow property specification and verification, implementing the properties listed in Table 1 and their analysis for the constructed model.

The evaluation consisted of analysing a matrix multiplication algorithm[3], in which the matrices to be multiplied contain values that can be of type `Short`, `Int`, `Long`, `Float` or `Double`. The algorithm works receiving as input a choice of the required data type and randomly creating two matrices with size 1000 x 1000 containing only values of the selected type, which are then multiplied. The experiments[4] were carried out using a PC with processor Core i5 3 GHz, with 16 GB of RAM. Figure 2 presents the behaviour model for the analysed algorithm. The energy cost of each action (represented inside the curly brackets before the transition label) consists of an average of 10 executions and its value is presented in Watts (W).

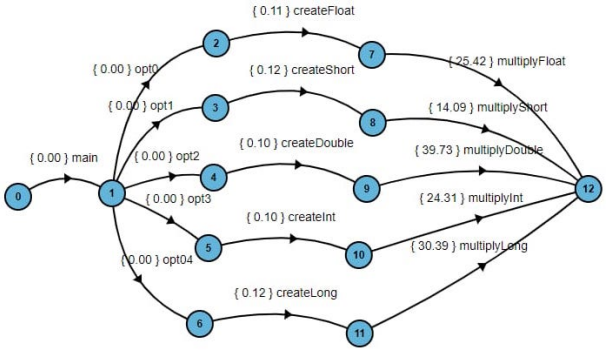

Figure 2: Model of the evaluated system

For this system, we were interested in the following questions: (Q1) What is the most and the least consuming type of matrix calculation? (Q2) What is the average energy consumption for any multiplication? (Q3) How many possible executions are above the average consumption? To answer these questions, we used analyses based on the properties described in the Table 1.

We verified property P2, which lists all executions and their respective energy costs, to answer Q1, Table 2 shows the executions ranked in decreasing order. The most and the least costly executions occur when the matrix has to deal with numbers of type Double (38.289W) and Short (14.210W), respectively.

Execution trace	Consumption (W)
main,opt2,createDouble,multiplyDouble	39.83
main,opt4,createLong,multiplyLong	30.51
main,opt0,createFloat,multiplyFloat	25.53
main,opt3,createInt,multiplyInt	24.41
main,opt1,createShort,multiplyShort	14.21

Table 2: Ranking of the execution costs

To obtain the average energy consumption, which responds to Q2, we verified property P5. After the verification, the obtained value was 26.895W. With this value, we verified property P4 in order to answer Q3. As a result, we found out that only two possible executions (40%) were above the average energy consumption: the ones dealing with numbers of types Double and Long.

As an instantiation of our framework, this experiment showed how we can implement each phase and obtain important information concerning relevant properties. Though our goal was just presenting an example of use of the framework, we should point out that the results can be heavily influenced by the choice of tools and programming language. In this case, we used a Java program, which means that part of the energy usage might be due to the Java Virtual Machine execution. Other programming languages with less execution overhead will probably generate results that can be actually attributed to the particular program. Moreover, different programming paradigms can also affect our modelling phase, as different types of code elements need to be considered, altering the energy measurement strategy as well.

In this experiment, we used jRAPL to collect energy information as it provides a simple way of marking the parts of the code to be considered. However, this could also create an execution overhead, as the measurement requires calling methods of an external library to gather information from hardware components. Apart from all that, the information provided by the results, though possibly not very precise, can help understand and maybe improve the code's energy efficiency, as modifications can be applied to reduce energy costs (e.g., replacing control structures or changing data structure types). In our experiment, for example, we can see that executing multiplications using types other than Long and Double,

[3] Adapted from jRAPL benchmarks available at http://kliu20.github.io/jRAPL.
[4] Data for this experiment can be found at https://github.com/danilodsa/MatrixEnergyMeasurement.

as expected, are more energy-efficient and, hence, only in 40% of the possible executions the program consumes above the average.

Our intention is to keep the framework as flexible as possible, hence the developer can determine what the transition labels represent. Depending on this choice, the developer has to guarantee consistency in the model (i.e., all transitions labels have to keep the same level of abstraction and represent the same type of element) and that the collected energy information matches the type of element represented in the model. However, we are aware that building a model might not be a trivial task and may affect the analysis results. Our idea was to present an example simple to understand, small enough to be fully described by a model that could be integrally presented in the paper, and for which we could provide analysis whose results would present some useful information.

Since we propose a compositional approach, the same process applied to the matrix program could be used for each component of a larger system, producing a model for each one of them. Hence, our small example describes the same procedure that could be applied to a multi-component project. Nevertheless, we plan, as future work, to propose a more general approach that takes into account a model construction phase, in which the model can be either manually constructed or automatically obtained using some model extraction or specification mining process, such as [10].

5 APPLICATION AREAS

In this section, we discuss some possible application areas for the proposed framework, highlighting how it can help users to address the inherent challenges of each area.

5.1 Component/Service-based development

A common way to develop modern software is to decompose it into a set of well defined, independent and intercommunicating components or services. More recently, microservices [26] have been gaining attention from both academia and industry, particularly due to its efficient manner to scale computational resources at runtime, thus becoming a trend as an architectural style for cloud-native applications [19]. A challenging and yet not addressed question regards how to build energy-efficient systems upon a set of components or (micro)services. In this direction, developers could benefit from our framework to model and exercise the possible architectural configurations based on the energy consumption of each available similar component, thus helping them choose the best configurations not only in terms of functional requirements, but also taking into account energy consumption.

5.2 Refactoring

Refactoring helps improve source code organisation through the application of structural modifications [12]. In the context of energy analysis, a developer could use the results of our framework to evaluate how applying a given refactoring would affect their local and/or overall energy consumption. In fact, the concern about energy usage adds a new aspect to software development: it is possible that refactoring a code, though beneficial from the aspect of code quality, could negatively affect its energy efficiency. Hence, the developer now has to consider the importance of each aspect, according to the specific application. Our framework can support this decision-making by providing results that can help weigh possible conflicting aspects. Different versions of a software could be compared and multiple changes considered.

5.3 Energy Optimisation

An interesting application would be the use of energy information to optimise software energy efficiency. Ideas about energy complexity analysis have already been presented [9], discussing how to determine, for instance, the minimum energy required to execute an algorithm. Using the results from applying our framework, developers could compare the energy efficiency of their solutions with possible lower/upper bounds. It would also be possible to discover what parts of the code are more inefficient and explore possible solutions to improve efficiency. Note that an energy-optimised code might not be the most optimised solution for a problem in terms of time and/or space, hence the developer has to consider side-effects. If a modification should be carried out, an AI-based approach could help decide how to apply it without undesired side-effects or, at least, help foresee implications of each alternative.

5.4 Self-adaptive systems

Self-adaptive systems are able to adjust their behaviour or structure in response to their perception of the environment and the system itself [8]. By using feedback control loops, such as the MAPE-K (Monitor, Analyse, Plan, Execute over a Knowledge base) [15], self-adaptive systems can monitor the system and environment properties, check whether they violate predefined requirements and, if so, choose and apply the best adaptation strategy that can improve the expected system quality. In this realm, energy consumption arises as a possible property that can be observed and lead a system to adapt at runtime in order to choose the most appropriate component (see Section 5.1) that keeps the system overall energy cost below a established acceptable threshold. Therefore, our framework can be used as a model@runtime [6] strategy that fits the first two phases of the MAPE-K loop.

5.5 Embedded Systems

If we can measure and analyse energy costs for a given software, we can do this process using different hardware components and configurations to see how they affect energy-efficiency. In particular, embedded systems [16] are systems where hardware and software are combined and, therefore, energy consumption should account for both types of components. Using simulators of architectures and microprocessors, one could execute a program using different settings and analyse how they influence energy efficiency in this composed scenario. Possible refactorings or optimisations might have a positive effect when running upon some device and not the desired result if some component is changed. Our framework could support gathering and analysing the necessary software energy information that, combined with hardware configurations, might indicate the best type of system for a certain type of application.

6 CONCLUSIONS AND FUTURE WORK

We have proposed a framework for the analysis of software energy consumption. Our approach comprises gathering energy information and adding it to model, which can be analysed for the verification of properties of interest. We developed an experiment where we used of a tool to collect energy information from the software under analysis and an extended version of an LTS-analysis tool to include this energy information in LTS models, which can be constructed and visualised in the tool. We have also implemented some basic energy-related analyses to generate relevant information about the energy-efficiency of the modelled software. Based on this support, we believe developers could create more energy-efficient software applications and also take into account possible trade-offs related to time, space, and energy costs when producing new versions of their systems.

Even though our preliminary results show important information about the energy consumption of the analysed application, we still need to further evaluate our framework using other applications and more complex systems containing, for instance, multiple components. We intend to explore the use of other tools to implement the framework to evaluate its flexibility and effectiveness, and software produced in other programming languages, as our experiment used a Java code due to the application of jRAPL. The framework also requires the introduction of new types of analyses. In particular, we plan to combine energy costs with probabilistic behaviour. This way, a developer could discover, for example, what is the probability of executing the most costly behaviour and decide whether it is worthy - or necessary - to make some change in the their code to improve software efficiency.

ACKNOWLEDGMENTS

This work is partially supported by CNPq/Brazil under the grant Universal 438783/2018-2.

REFERENCES

[1] Susanne Albers. 2010. Energy-efficient Algorithms. *Commun. ACM* 53, 5 (May 2010), 86–96. https://doi.org/10.1145/1735223.1735245

[2] asmeet Singh, Kshirasagar Naik, and Veluppillai Mahinthan. 2015. Impact of Developer Choices on Energy Consumption of Software on Servers. *Procedia Computer Science* 62 (2015), 385 – 394. https://doi.org/10.1016/j.procs.2015.08.423 SCSE'15.

[3] Christel Baier, Clemens Dubslaff, Joachim Klein, Sascha Klüppelholz, editor="van Breugel Franck Wunderlich, Sascha", Elham Kashefi, Catuscia Palamidessi, and Jan Rutten. 2014. *Probabilistic Model Checking for Energy-Utility Analysis.* Springer International Publishing, Cham, 96–123. https://doi.org/10.1007/978-3-319-06880-0_5

[4] Davi Monteiro Barbosa, Rômulo Gadelha de Moura Lima, Paulo H. Maia, and Evilasio Costa. 2017. Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-adaptive Systems. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (SEAMS '17).* IEEE Press, Piscataway, NJ, USA, 24–30. https://doi.org/10.1109/SEAMS.2017.18

[5] Nathan Binkert, Bradford Beckmann, and et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[6] Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. Models@run.time. *Computer* 42, 10 (October 2009), 22–27. https://doi.org/10.1109/MC.2009.326

[7] Fabio Salice Donatella Sciuto Carlo Brandolese, William Fornaciari. 2002. The impact of source code transformations on software power and energy consumption. *Journal of Circuits, Systems, and Computers* 11, 05 (2002), 477–502.

[8] Betty H. C. Cheng, Rogério de Lemos, and et al. 2009. *Software Engineering for Self-Adaptive Systems: A Research Roadmap.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–26. https://doi.org/10.1007/978-3-642-02161-9_1

[9] Erik D. Demaine, Jayson Lynch, Geronimo J. Mirano, and Nirvan Tyagi. 2016. Energy-Efficient Algorithms. In *ITCS '16 (2016 ACM Conference on Innovations in Theoretical Computer Science).* ACM, New York, NY, USA, 321–332. https://doi.org/10.1145/2840728.2840756

[10] Lucio Mauro Duarte, Jeff Kramer, and Sebastian Uchitel. 2017. Using contexts to extract models from code. *Software and Systems Modeling* 16, 2 (2017), 523–557. https://doi.org/10.1007/s10270-015-0466-0

[11] Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. 2014. Probabilistic Model Checking for Energy Analysis in Software Product Lines. In *MODULARITY '14 (MODULARITY '14).* ACM, New York, NY, USA, 169–180. https://doi.org/10.1145/2577080.2577095

[12] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[13] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. 2013. Estimating mobile application energy consumption using program analysis. In *35th International Conference on Software Engineering (ICSE 2013).* IEEE Press, 92–101.

[14] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle. 2016. Energy profiles of java collections classes. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE).* ACM, 225–236.

[15] IBM. 2005. *An Architectural Blueprint for Autonomic Computing.* Technical Report. IBM.

[16] R. Jayaseelan, T. Mitra, and Xianfeng Li. 2006. Estimating the worst-case energy consumption of embedded software. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06).* IEEE, 81–90.

[17] Robert M. Keller. 1976. Formal Verification of Parallel Programs. *Commun. ACM* 19, 7 (July 1976), 371–384.

[18] Hammad Khalid, Emad Shihab, and et al. 2015. What Do Mobile App Users Complain About? *IEEE Software* 32, 3 (May 2015), 70–77. https://doi.org/10.1109/MS.2014.50

[19] Nane Kratzke. 2018. A Brief History of Cloud Application Architectures. *Applied Sciences* 8, 8 (2018). https://doi.org/10.3390/app8081368

[20] Marta Kwiatkowska, Gethin Norman, and David Parker. 2002. PRISM: Probabilistic Symbolic Model Checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS 2012)*, P. Kemper (Ed.). 200–204.

[21] D. Li, B. Guo, Y. Shen, J. Li, J. Wang, Y. Huang, and Q. Li. 2016. Software Energy Consumption Estimation at Architecture-Level. In *2016 13th International Conference on Embedded Software and Systems (ICESS).* 7–11. https://doi.org/10.1109/ICESS.2016.35

[22] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* 469–480.

[23] Kenan Liu, Gustavo Pinto, and Yu David Liu. 2015. Data-Oriented Characterization of Application-Level Energy Optimization. In *Fundamental Approaches to Software Engineering*, Alexander Egyed and Ina Schaefer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 316–331.

[24] Andrea McIntosh, Safwat Hassan, and Abram Hindle. 2019. What can android mobile app developers do about the energy consumption of machine learning? *Empirical Software Engineering* 24, 2 (2019), 562–601.

[25] Shin Nakajima. 2015. Model Checking of Energy Consumption Behavior. In *Complex Systems Design & Management Asia*, Michel-Alexandre Cardin, Daniel Krob, and et al. (Eds.). Springer International Publishing, Cham, 3–14.

[26] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly Media, Inc.

[27] Wellington Oliveira, Renato Oliveira, Fernando Castor, Benito Fernandes, and Gustavo Pinto. 2019. Recommending energy-efficient Java collections. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR 2019).* IEEE Press, 160–170.

[28] R. Pereira. 2017. Locating Energy Hotspots in Source Code. In *39th International Conference on Software Engineering (ICSE 2017).* 88–90. https://doi.org/10.1109/ICSE-C.2017.151

[29] Rui Pereira, Marco Couto, João Saraiva, Jácome Cunha, and João Paulo Fernandes. 2016. The influence of the Java collection framework on overall energy consumption. In *Proceedings of the 5th International Workshop on Green and Sustainable Software.* ACM, 15–21.

[30] Gustavo Pinto and Fernando Castor. 2017. Energy Efficiency: A New Concern for Application Software Developers. *CACM* 60, 12 (December 2017), 68–75. https://doi.org/10.1145/3154384

[31] S. Schubert, D. Kostic, W. Zwaenepoel, and K. G. Shin. 2012. Profiling Software for Energy Consumption. In *2012 IEEE International Conference on Green Computing and Communications.* 515–522. https://doi.org/10.1109/GreenCom.2012.86

[32] V. K. Singh, K. Dutta, and D. VanderMeer. 2013. Estimating the Energy Consumption of Executing Software Processes. In *GreenCom, iThings and CPSCom 2013.* 94–101. https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.40