Green Configurations of Functional Quality Attributes

Jose-Miguel Horcas
Universidad de Málaga
Andalucía Tech, Málaga, Spain
horcas@lcc.uma.es

Mónica Pinto
Universidad de Málaga
Andalucía Tech, Málaga, Spain
pinto@lcc.uma.es

Lidia Fuentes
Universidad de Málaga
Andalucía Tech, Málaga, Spain
lff@lcc.uma.es

ABSTRACT

Functional quality attributes (FQAs) are those quality attributes that, to be satisfied, require the incorporation of additional functionality into the application architecture. By adding an FQA (e.g., security) we can improve the quality of the final product, but there is also an increase in energy consumption. This paper proposes a solution to help the software architect to generate configurations of FQAs whilst keeping the energy consumed by the application as low as possible. For this, a usage model is defined for each FQA, taking into account the variables that affect the energy consumption, and that the values of these variables change according to the part of the application where the FQA is required. We extend a Software Product Line that models a family of FQAs to incorporate the variability of the usage model and the existing frameworks that implement FQAs. We generate the most eco-efficient configuration of FQAs by selecting the framework with the most suitable characteristics according to the requirements of the application.

KEYWORDS

Quality Attributes, Energy Consumption, FQA, SPL, Variability

1 INTRODUCTION

Quality Attributes (QAs), or non-functional properties of an application (e.g., security, usability,...), can be improved by appropriately modifying the software architecture of the application. For instance, the performance of a web application can be improved by adding a caching component that serves stored data faster. This additional functionality, which is incorporated into the software architecture to satisfy or improve the application's quality is known as a Functional Quality Attribute (FQA) [3, 4, 8]. By adding an FQA like security we can improve the quality of the final product, but there is also an increase in energy consumption [5].

Each application requires a customized configuration of each FQA. For instance, an application may require integrity and encryption to satisfy security, while another may require only encryption, or may require a different encryption algorithm. Moreover, there are several frameworks and third party libraries that provide different implementations of FQAs ready to be reused, such as the Java Security package, the Apache Commons library, and the Spring Framework. Although there is much variability in FQAs, not all variants and implementations fulfill the system's quality requirements in the same way. Each framework provides different degrees of quality, at the expense of a higher or lower use of resources, and therefore different energy consumptions.

The software architecture should not only guarantee a certain quality but also that the generated software product is energy-efficient. However, analyzing relationships between FQAs and energy consumption has still not received enough attention [5]. In this paper, we focus on enhancing the energy efficiency of the FQAs added to applications, so the main goal is to generate the most eco-efficient FQA configurations (i.e., that consume the least energy), which fulfill the application's requirements.

Since software architects are not normally aware of the energy consumption of architectural solutions [10], our approach will automatically generate the FQA configurations taking into account the energy consumption of the different variants. We also consider that the same FQA can be injected into different places of the application, which can have different usage models. Our usage model defines the variables and parameters that affect the energy consumption of the FQA, whose values will depend on the part of the application where the FQA is injected.

In previous work, we defined WeaFQAs [4], a Software Product Line (SPL) process that models an FQA family, and generates and injects FQA customized configurations into the application architecture. However, when customizing the FQAs it is recommendable to also consider the influence of the new components on other non-functional QAs, like the energy efficiency in this case.

In this paper, we extend the WeaFQAs approach to: (1) define each FQA as a clonable characteristic, to allow different configurations of the same FQA to be added to different parts of the application architecture; (2) add the usage model of each FQA to the variability model, to allow the software architect to instantiate it according to the necessities of the application point where it is injected; (3) extend the WeaFQAs variability model with the frameworks and libraries that implement the FQAs modeled, explicitly specifying the configurability degree allowed by each of them, and (4) generate the most eco-efficient configuration by selecting the most appropriate framework for each FQA and customizing it to the application's requirements.

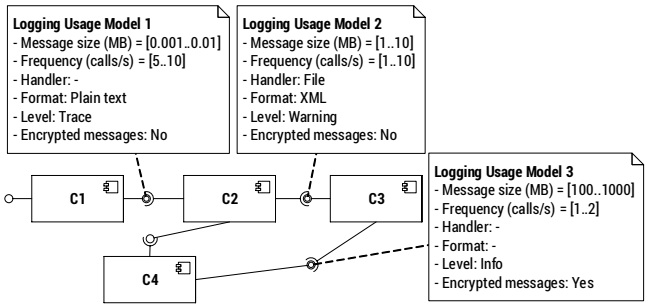

Figure 1: Examples of usage models for the logging FQA.

The remainder of this paper is structured as follows. In Section 2 we characterize the FQAs and define the usage model for the FQAs. Section 3 extends our previous work with the usage models and the FQA frameworks. Section 4 explains how to generate eco-efficient configurations from the experimentation results. Section 5 evaluates and discusses our proposal. Section 6 presents the related work, and Section 7 concludes the paper.

2 FQAS AND ENERGY EFFICIENCY

Before analyzing and selecting the most eco-efficient configurations of the FQAs, we first need to characterize how FQAs influence energy consumption. For each functionality of the FQAs, for example encryption, hashing, or logging, we identify the variables that affect the energy consumption, and the possible values. For instance, the logging functionality of the usability FQA has a high degree of variability. Messages can be logged in different formats (e.g., plain text and XML), or can be sent to different outputs or handlers such as console, file, or database. Messages can have also different levels of severity (e.g., trace, debug, and warning), and can be encrypted if necessary. Thus, the usage model of an FQA is defined as the set of variables that affect the energy consumption of that FQA, and the values that each variable can take [1, 10].

The energy consumption of an FQA in a particular application is determined by the usage model of the FQA in the different places of the application architecture where the FQA has to be injected. The characterization of the FQAs allows us to build a generic usage model that can be instantiated by the software architect multiple times in an application, in different parts. To illustrate this, Figure 1 shows the usage model of the logging functionality for three different component interactions.

Assigning concrete values to the usage model variables can be a difficult task, since the values can be unknown to the software architect. Moreover, the software architect may want to analyze the energy consumption of all possible variants of the usage model. In those cases, our proposal will support the partial instantiation of the usage model, as shown in `Logging Usage Model 1` and `Logging Usage Model 3` in Figure 1, where the handler of the message (i.e., console or file) is not provided in the former case, or the format (i.e., plain text, XML) in the latter.

The effect of each variable of the usage model in the energy efficiency of the FQA is pre-calculated through experimentation. As part of our proposal we measure the energy consumption of the frameworks implementing FQAs, for the different values (treatments) of each variable (factor) [11]. An example of this experimentation is presented in Section 4.1, where we show the results of

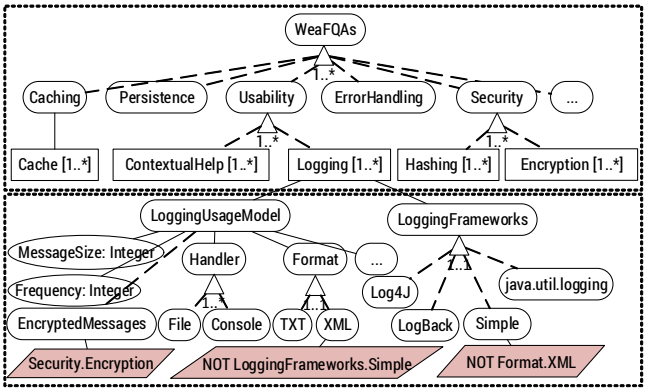

Figure 2: Variability model extended.

assessing the energy consumption of the logging functionality for several Java frameworks. Note that the specific values of energy consumption are not relevant for the characterization of FQAs, but they are relevant for the energy consumption trends of the FQA when the values of the variables change. For this reason, in this paper we assume that the FQA characterization and the analysis of the energy consumption can be performed independently of a specific application. Note, that any experimental measure of energy consumption must be considered as an estimation only, since the power consumed by an application will be different in successive executions. This is because there are many uncontrollable factors that affect power consumption (e.g., temperature, garbage collection), so identifying power consumption tendencies is enough.

The next section details how we have extended the variability model of WeaFQAs to include the information of usage models, the frameworks that implement the functionalities of the FQAs, and the clonable features of the FQAs.

3 EXTENDING THE VARIABILITY MODEL OF WEAFQAS

The extended WeaFQAs variability model is shown in Figure 2. The variability model specifies a complete family of FQAs (e.g., security and usability) and the functionalities of each of them (e.g., encryption and logging). First, we have modified the features of the variability model that represent the functionalities of the FQAs to make them clonable, as, for example the `Logging [1..*]` feature. A clonable feature (features with cardinality [1..*]) allows that feature to be instantiated multiple times, and all its sub-features can be configured differently for each instance [2]. Second, we extend the variability model by adding, for each functionality of the FQAs, one feature to model the usage model (`LoggingUsageModel`) and another that models the frameworks and libraries that implement the functionalities of the FQAs (`LoggingFrameworks`). Children of any usage model feature are those variables of the FQA that affect the energy consumption. For example, for logging, the usage model is composed by the variables `message size`, `frequency`, `format`, and `handler`.

Note that some variables of the usage model are not supported for all frameworks, so in those cases we define cross-tree constraints in the variability model to limit the possible selections. For example, not all frameworks provide the possibility of logging the message in XML format, such as the simple implementation of SLF4J

(Simple), and thus, we specify the constraint `Format.XML implies NOT LoggingFrameworks.Simple` to prevent that framework from being selected when the format is XML. Similarly, we explicitly define the dependency with the encryption functionality when the log messages need to be encrypted (`EncryptedMessages implies Security.Encryption`). This creates an additional instance of the encryption functionality to be configured.

4 GENERATION OF ECO-EFFICIENT CONFIGURATIONS

Using the WeaFQAs variability model, the software architect can generate a complete or partial configuration of the FQAs according to the application's requirements. In this paper, the goal is to help the software architect to generate the most eco-efficient configurations. To do so, first we need to evaluate how the variables of the usage models affect the energy consumption, independently of the application (Section 4.1). Second, we need to integrate the energy information gathered with the variability model (Section 4.2). Finally, the most eco-efficient configurations are automatically selected based on the energy information (Section 4.3).

4.1 Experimentation: energy consumption

To illustrate the details of our proposal, we show the results of evaluating the energy consumption of the logging functionality of the usability FQA with different Java frameworks, based on the characterization described in Section 2.

The experiments were conducted on a desktop computer with Intel Core i7-4770, 3.40 GHz, 16 GB of memory, Windows 10 64 bits with the high performance option activated, and Java JDK 1.8. Measurements of energy were obtained using the IPPET (*Intel Platform Power Estimation Tool*)[1] tool that estimates the energy consumption of the CPU at the process level.

Experiments to characterize the logging functionality consisted in delivering a set of log messages using four different implementations of the SLF4J (Simple Logging Facade for Java)[2] logging API, the java.util.logging package directly provided by the Java JDK, the Log4J[3] and the LogBack[4] frameworks, and a simple implementation of SLF4J[5]. We logged plain text messages varying the message size from 100 Bytes to 1 GByte in powers of 10. Each variable or characteristic from the usage model was evaluated independently, the other variables remained fixed. Each experiment was performed ten times, and we took the average value.

Figure 3 shows the results of the energy measurements (in Joules) of the logging functionality for different message sizes, using two different handlers: console and file. On the one hand, we can observe that the most eco-efficient framework is, in general, the simple implementation of the SLF4J API, while LogBack is the framework with the most energy consumption in all cases (except for messages of 1 GB). We also observe in Figure 3 that for messages of a huge size (1 GB) it is more eco-efficient to send the messages to file and not to the console. Log messages of this size are not common in desktop or mobile applications, but in larger-scale applications such

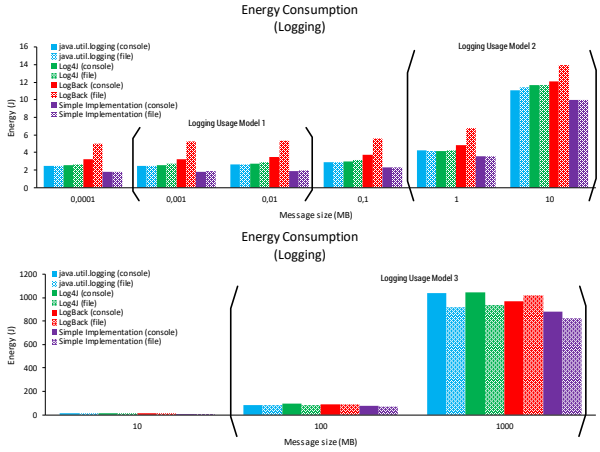

Figure 3: Energy consumption of the logging FQA.

as Google farms[6] and CERN data centers[7], where the complete status of applications must be logged frequently, they are more common. In these data centers the energy efficiency is even more important. Log4J and LogBack frameworks consume more than others because their purpose is to provide full functionality and a great performance regardless of energy consumption.

4.2 Integrating the energy information in WeaFQAs

In order to integrate the energy information with the variability model, we formalize the variability model as a Constraints Satisfaction Problem (CSP).

A CSP problem is defined by a triplet (X, D, C), where X is the set of variables, D is a set of domains for the variables, and C is the set of constraints that must be satisfied. We map the variability model to these concepts. X variables are the features of the variability model. To integrate the information of the energy consumptions, we define an additional variable for each different configuration that represents its energy consumption value. Domain D is $\{0, 1\}$ for the variability model's features to indicate whether the feature has been selected or not in a configuration. The domain of the variables that represent energy consumption values is the set of real numbers $\mathbb{R}$. The constraints include the tree and cross-tree constraints of the variability model. Additionally, we also define as constraints the selections made by the software architect in a configuration of the variability model. Each constraint that represents a configuration is associated with an energy consumption value. By resolving the CSP for each partial configuration of each FQA, we can obtain the most eco-efficient configuration of each FQA.

4.3 Eco-efficient configurations

From the results obtained in the experimentation and the information formalized in the previous section, our proposal is able to select the most eco-efficient global configuration of the FQAs adapted to the application's requirements. First, for each FQA required by the application, the software architect needs to identify the join points (or interactions) in the application architecture where the

[1] https://goo.gl/noZsYk
[2] https://www.slf4j.org/
[3] http://logging.apache.org/log4j/1.2/index.html
[4] https://logback.qos.ch/
[5] https://www.slf4j.org/apidocs/org/slf4j/impl/SimpleLogger.html

[6] https://www.google.com/about/datacenters/
[7] http://information-technology.web.cern.ch/about/computer-centre

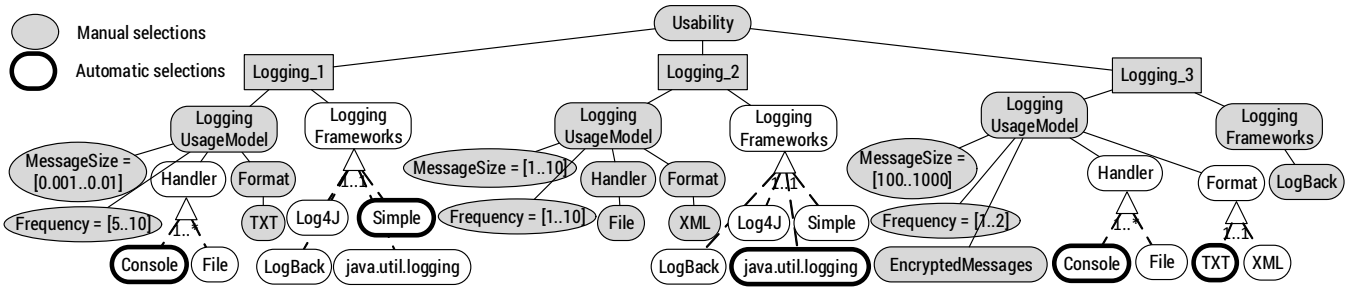

Figure 4: Configuration of the logging FQA instances.

FQA will be incorporated. Second, for each join point the architect creates an instance of the usage model for the injected FQA. This implies having to provide the values of the variables that compose the usage model — i.e., specifying how the FQA will be used in that specific part of the application.

When the software architect provides a complete configuration for an FQA (i.e., they provide all values for the usage model and select a particular framework according to the application's requirements), the configuration generated may not be the most eco-efficient. So, in our approach it is more useful to partially instantiate the usage models, so we can then automatically complete the most eco-efficient configuration. Our approach will automatically select those features that were not selected by the software architect with the values that minimize the energy consumption (e.g., select the most eco-efficient framework for that usage model). It may be the case that the software architect may want to use a specific framework, but they do not know how to configure the internal characteristics of it. In this case, our approach generates the most eco-efficient configuration taking into account the characteristics and variables of that particular framework.

To select the most eco-efficient configuration, we minimize Equation 1. The energy consumption of a global configuration of the FQAs is the sum of the different configurations (`config`) of each instance (i) of the FQA functionalities (f). The energy consumption of each instance is calculated from the usage model of the FQAs, using the values obtained in the experimentation (Section 4.1). The solution of the objective function can be obtained using any optimization technique such as programming constraints or evolutionary algorithms.

$$\texttt{Energy\_consumption}(FQAs) = \sum_{f \in FQAs} \sum_{i=1}^{N} \texttt{Energy\_consumption}(config_i(f)) \quad (1)$$

Figure 4 shows a configuration of the variability model with three instances for the logging functionality customized with the values of the usage models shown in Figure 1. In the first interaction (`Logging Usage Model 1`), the software architect has defined the message size as between 0.001 MB and 0.01 MB, and a frequency of 5 to 10 messages per second. Messages will be logged at trace level in plain text and without encryption. However, the architect has not specified the handler (console or file) nor the framework to be used, so our proposal will automatically select the most eco-efficient handler (i.e., console) for the values provided, and the most eco-efficient framework that provides the chosen characteristics: the simple implementation of SLF4J. In the second instance (`Logging Usage Model 2`), the architect has provided a complete configuration, but has not selected the framework to be used. In this case our proposal will select the library java.util.logging because this is the most eco-efficient option that fulfills the constraints of the variability model — i.e., it is the most eco-efficient framework that allows logging messages of between 1 and 10 MB to file in XML (see constraint in Figure 2). Finally, the architect has provided a partial configuration for the third instance (`Logging Usage Model 3`), selecting the option for encrypting the message and a specific framework: LogBack. In this case, our proposal will select the console handler and the plain text format (`TXT`) that is the most eco-efficient configuration when the LogBack framework has been selected.

5 EVALUATION

In this section we evaluate our proposal. First, we demonstrate that the generated configurations are correct and the most eco-efficient for each usage model. Then, we discuss the lessons learnt from our proposal and how the validity of our evaluation could be threatened.

5.1 Correctness Evaluation

We can resolve the CSP problem defined in Section 4.2 using a CSP solver to guarantee that our proposal generates the same valid configuration and that those configurations are the most eco-efficients. To do so, we complete the CSP problem by defining an objective function that minimizes the energy consumption of the FQAs' configurations, satisfying all the constraints defined in the variability model. To resolve the problem, we have used Choco[8], a Java library for constraint programming. Configurations generated by Choco satisfy the usage models introduced by the software architect, guaranteeing that the configurations generated by our proposal are the most eco-efficients.

5.2 Discussion

In this section we discuss the lessons learnt from our proposal and how the validity of our evaluation could be threatened.

Measurement of the energy consumption in applications. The real energy consumption of the generated FQA configurations can vary due to several factors that are independent of the usage model, for example, the load of the system in a particular moment, or the degree of resource sharing (e.g., network usage, disk). The effect of these external factors on the energy consumption of the FQAs is unknown and sometimes unpredictable. However we think it is possible to consider these external factors in our approach by adding new external variables to our model, apart from the usage model, and performing the experiments based on real or expected execution contexts of the final application.

[8] http://choco-solver.org/

***Trade-off* between quality attributes.** Although in this paper we focus on energy efficiency, FQAs also affect other non-functional properties such as performance, level of security or usability. Our proposal can be used to generate those configurations that optimize other quality attributes, after characterizing the variables that affect the quality attribute and modifying the usage model accordingly. Moreover, our proposal can be extended to perform a trade-off between some quality attributes [5].

Determinism of the proposal. Our proposal is based on the data obtained from the experimentation to generate the most eco-efficient configuration. By performing the same experiments on another computer, the actual energy values may vary due to many factors (mainly hardware), but the comparative results should remain correct. In specific cases where the execution environment is very different from a general-purpose computer, such as supercomputers or special-purpose computers, the experimentation results may be very different and, therefore, our proposal will not generate the same configurations. However, our proposal will generate the most eco-efficient configurations for a specific environment.

6 RELATED WORK

There are some existing approaches that have identified the necessity of managing FQAs [3, 4, 8, 9]. For instance, in [3], the authors argue that most of the so called non-functional requirements are not really non-functional since they describe part of the functional behavior of the system.

In [8], the authors apply an inductive research method to identify FQAs. They identify functionalities related to the usability FQA that affect the software architecture, and define patterns to implement those functionalities in different applications. However, the authors only focus on usability, and our approach is more generic since it supports any kind of FQA (e.g., security, persistence, usability,...). In fact, usability functionalities identified in [8] can be added to our approach to improve its variability model. In [9], the authors present CORE (Concern-Oriented REuse), a software development paradigm where every kind of software characteristic, from base application functionality to non-functional properties, are encapsulated in reusable units called *concerns*. Although they do not directly manage the concept of FQAs, they identify and encapsulate as concerns the functionality of the FQAs.

Finally, in [4], WeaFQAs is defined. WeaFQAs combines SPLs and aspect-orientation in a generic process to model, configure, and automatically inject the FQAs into the software architecture of the applications. However, in WeaFQAs the FQAs are customized just based on the functional requirements of the application, additional information such as the energy consumption of different configurations is not considered. In addition, WeaFQAs does not consider the possibility of configuring the same FQAs differently in different parts of the application — e.g., encrypting in different places of the application but using a different algorithm, key size, mode, padding, or any other parameter. Finally, WeaFQAs does not take into account existing frameworks that implement FQAs.

However, none of these approaches [3, 4, 8, 9] analyze the energy consumption of different variations of the software architecture. This is because they include neither the support to represent the usage model, nor the configurable characteristics of the frameworks that implement the FQAs. These are essential to analyze the energy consumption of a given architectural configuration.

The relevance of reasoning about energy efficiency at the architectural level is to be able to compare the energy consumption of the different architectural configurations of the same applications (architectural patterns, design variations, deployment, etc.). There are some approaches which focus on the definition of architectural tactics [5] and design patterns [6] driven by the energy. There are also new architectural description languages (ADLs) that include energy profiles and analysis of energy consumption [7, 10]. In any case, the experimentation consists of estimating the energy consumption of the application code to analyze the effects of applying a specific architectural pattern or design [5, 6].

7 CONCLUSIONS AND FUTURE WORK

In this paper, a proposal to generate eco-efficient configurations of functional quality attributes (FQAs) has been presented. The proposal is useful because it helps the software architect to incorporate the most appropriate configuration of the FQAs at each part of the application where the FQA is required, optimizing energy efficiency based on a usage model definition. We have evaluated the energy consumption of different frameworks that implement FQAs.

As future work we intend to extend the proposal to generate FQA configurations with a trade-off between energy efficiency and performance, among other quality attributes.

ACKNOWLEDGMENTS

This work is supported by the projects Magic P12-TIC1814 and HADAS TIN2015-64841-R (co-financed by FEDER funds).

REFERENCES

[1] Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2009. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82, 1 (2009), 3–22.
[2] Krzysztof Czarnecki and Chang Hwan Peter Kim. 2005. Cardinality-based feature modeling and constraints: a progress report. In *International Workshop on Software Factories at OOPSLA'05*. ACM, ACM, San Diego, California, USA.
[3] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. 2016. Are "Non-functional" Requirements Really Non-functional?: An Investigation of Non-functional Requirements in Practice. In *International Conference on Software Engineering (ICSE)*. 832–842.
[4] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2016. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software* 112 (2016), 78–95.
[5] Erik Jagroep, Jan Martijn van der Werf, Sjaak Brinkkemper, Leen Blom, and Rob van Vliet. 2016. Extending software architecture views with an energy consumption perspective. *Computing* (2016), 1–21.
[6] A. Noureddine and A. Rajan. 2015. Optimising Energy Consumption of Design Patterns. In *International Conference on Software Engineering*, Vol. 2. 623–626.
[7] B. Ouni, H. Ben Rekhissa, and C. Belleudy. 2012. Inter-process communication energy estimation through AADL modeling. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. 225–228.
[8] Francy D. Rodríguez, Silvia Teresita Acuña, and Natalia Juristo Juzgado. 2015. Reusable Solutions for Implementing Usability Functionalities. *International Journal of Software Engineering and Knowledge Engineering* 25, 4 (2015), 727–756.
[9] Matthias Schöttle, Omar Alam, Jörg Kienzle, and Gunter Mussbacher. 2016. On the Modularization Provided by Concern-oriented Reuse. In *Modularity*. 184–189.
[10] Christian Stier, Anne Koziolek, Henning Groenda, and Ralf H. Reussner. 2015. Model-Based Energy Efficiency Analysis of Software Architectures. In *European Conference on Software Architecture (ECSA)*. 221–238.
[11] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*.