Bachelor's Thesis

FEATURE PERFORMANCE ANALYSIS: DIFFERENCES BETWEEN BLACK-BOX AND WHITE-BOX MODELS IN CONFIGURABLE SYSTEMS

MANUEL MESSERIG

February 16, 2023

Advisor:
Florian Sattler Chair of Software Engineering

Examiners:
Prof. Dr. Sven Apel Chair of Software Engineering
Second Examiner Affiliation 2

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT DES SAARLANDES

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

ABSTRACT

Nearly, all modern software systems are configurable, a major reasons for that is that we as the developer want to give the user the flexibility of configuring the system to their needs, by offering them to turn functionality on and off.

However, nowadays configurable systems offer various features which result in a large ammount of unique configurations. All of these different configurations result in different behaviours of the system, one of the metrics shareholder are intrested is the running times of system under each configuration and in what capacity each feature influcences the running time. To analyze the influence of these features we will introduce two different approaches, a white-box and black-box approach. We will use the data generated by those approaches to build performance-influence models and use them to analyze and compare both approaches.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

IDE Integrated Development Environment

1 INTRODUCTION

Modern software systems are designed to be configurable, we want to give the user flexibility, by offering them to turn functionality on and off. We also expect a configurable software system to satisfie the demand of multiple user by offering a single software system that contains multiple features. [1].

One example for such a software system would be the Linux kernel, the code base itself is over 6.000.000 lines of code containing more than 1.000 optional features [1]. All these optional features allow you to create a operating system that suits your needs. To effectively analyze such systems we introduce two different approaches one white box and a black box approach.

This results in a mutlitute of features that influence the system in different ways, to keep a overview of all these features and the ways they interact with one another we will use a feature model, which is in essence a tree that can contain different kinds of constraints to visualize the relationships between features in a configurable system [7].

In the black-box approach we only have a system that takes a input, in our case this would be a selection of multiple different features, the system will then be executed with our configuration, during the execution we can meassure various metrics, such as memory and energy consumption, however we will focus on the meassured exectuion time.

In our white-box approach, we have more information because we know the inner workings of the system itself, i.e., we know what code contributes to which feature and can therefore meassures the time we spend in each feature by summing up the time spend in the code when its executed.

Both of our approches produce different kinds of data, that we still need to compare and evaluate, for this we use performance- influence models. These model represent our configurable system as a polynomial, whereas each term represents either a feature or a interaction of features [9]. To generate these models we use the data produced by the white-box and black-box approach.

To show the validity of our approaches we establish a ground truth. For this we design a small configurable system to test both of our approaches, this system will include multiple features that partly interact with each other in different ways. Since we designed this system ourself we are aware of how each feature should influence the runtime of our system, therefore we generate a baseline performance-influence model that we use to compare our models generated by our white-box and black-box approach to.

After confiraming the validity of our approaches, we will use both of our approaches on real world systems, such as the compression tool XZ. We will use both approaches on the same data and reapeat the experiment 30 times for each configuration to reduce external factors such as meassuremest noise.

1 https://www.kernel.org/doc/html/v4.14/admin-guide/kernel-parameters.html

1.1 RESEARCH QUESTIONS

In this thesis the main focus is about comparing performance-influence models between white-box and black-box models. Before we can even compare these models, we need to check whether they are able to detect interactions between features, and if so, how accurate they are. If they can identify the interactions between the features, we can start to compare them.

First and foremost, we are interested in whether both models come to the same conclusions, after all, they have both analyzed the same system. In the case where they come to the same conclusion, we can already see that it is possible to use either of the two models to analyze a system, but from there we are still working out the advantages and trade-offs between the models so that the user can choose the one that meets their needs. If they do not reach the same conclusion, we analyze the reason for the differences between them and examine whether one model performs particularly poorly in certain cases and why this is so. We answer the following research questions:

RQ1: How accurately does white-box and black-box approaches detect feature interactions?

We will design a program that contains multiple different feature interactions. Afterwards, we use both white-box and black-box approaches to generate a performance-influence model for each approach. We compare both models against the expected time of our implementation.

RQ2: Do performance models created by our white-box and black-box reach the same conclusion?

We use our black-box and white-box approach to collect measurements that we use to build our performance-influence models, and then we compare these models with each other to see if they agree with one another.

RQ3: Can we identify the reasons for similarities or differences between performance models?

We compare the coefficients of each feature between the models, if they differ we start analyzing the reason for that.

2 FEATURE MODEL

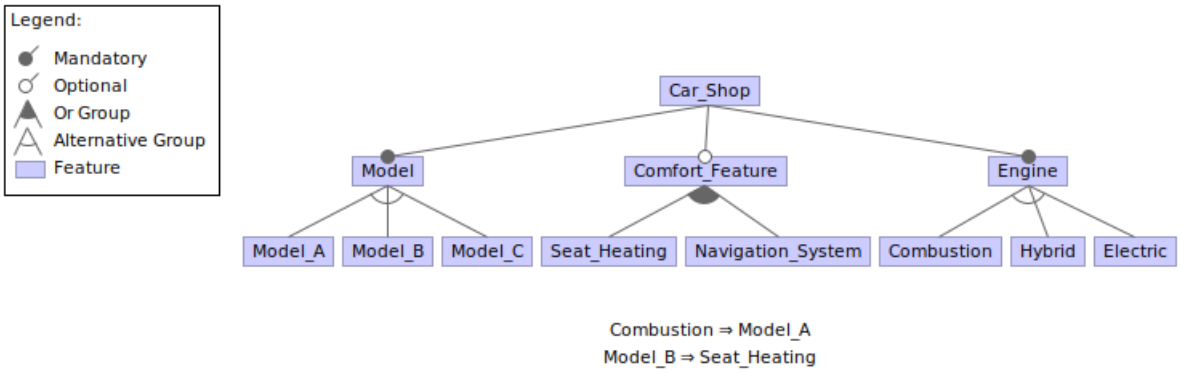

Figure 2.1: A feature model of a car shop

A configurable system often contains numerous features, all these features might interact with one another or have different dependencies. To help us keep a overview of the system we introduce feature model [2]. A example of a feature model for a car shop can be found in 2.1

A feature model is essentialy a tree, that models a configurable system, each node in that tree represent a specific feature whereas the root represent the system itself. If a feature is selected it implies that also the parent of that features needs to be selected as well, the further we decend in the tree the more specific features get. In 2.1 we see Engine is a feature where as the children a concrete types of engines, specificly Combustion, Hybrid and Electric.

Each feature may contain a grafical notation to indicate whether the feature is mandatory or optional, if the feature is mandatory, it is indicated with a black bubble and if its optional its optional its indicated with a empty bubble, in 2.1 we can see that Engine and Model are mandatory features, which make sense since both are necessary for each car, but Comfort_Feature a optional since they are not necessary for a car to function.

Besides mandatory and optional feature we also have alternative and choice groups. A parent can have one of these groups, a alternative group is denoted with a empty half circle and a optional group with a filled circle. If a choice group is used, one feature needs to be selected, but others can be selected as well, a choice group corresponds to the logical or operator. In a alternative group, one and only one feature can be selected, if more the one is selected the configuration is invalid. In 2.1 we see Engine has a alternative group, which makes sense since each car can only contain one Engine, where it makes sense that Comfort_Feature contains a choice group, you can have a navigation system and seat heating in a car without any conflics.

In addition to that, a feature model can contain different constraints that need to be satisfied, these constraints are defined as boolean algebra. In 2.1 we can see that there are

two constraints, $Combustion \implies Model\_A$ and $Model\_B \implies Seat\_Heating$, the reasons for such constraints could be that Model_B is a luxurious model where it only gets shipped with seat heating.

Every feature model can also be translated into compositional logic [2], therefore configuration of features is only valid iff it satisfies the propositional logic. We can therefore use a feature model to check wheter a configuration is valid, which is especially useful if we want to sample or enumerate all valid configurations.

3

PERFORMANCE-INFLUENCE MODELS

Both of white-box and black-box approach produce data that are different from each other, therefore we cannot directly compare them, for this reason we intoduce performance-influence models. We generated such model for each set of data produce by our both approaches, afterwards we are able to compare and analyze both model. We use the formal definitons from the paper "Performance-Influence Models for Highly Configurable Systems" by Sven Apel et al. [9].

A feature performance model is a polynomial that consists of multiple terms, whereas each term represent either a feature or a interaction of features and the influence they have onto the system itself. The sum of all terms represent the time the performance model predicts given a configuration of features. All our features are encoded as binary features, meaning they can either be turned on or off. Formally, let $\mathcal{O}$ be the set of all configuration options and $\mathcal{C}$ the set of all configuration, then let $c \in \mathcal{C}$ be a function $c : \mathcal{O} \implies \mathbb{R}$ assigning either 0 or 1 to each binary option. If we select a feature o, then $c(o) = 1$ holds, and if we choose to unselect a feature $c(o) = 0$. In general a performance-influence model is a function Π that maps configurations *mathcal* to a estimated time preditcion, therefore $\Pi : \mathcal{C} \implies \mathbb{R}$.

We differentiate between single features o denoted as ϕ_o and feature interactions denoted $i...j$ denoted as $\Phi_{i...j}$. With these definitions we can now define how a performance-influence model formally looks like:

$$\Pi = \beta_0 + \sum_{i \in \mathcal{O}} \phi_i(c(i)) + \sum_{i...j \in \mathcal{O}} \Phi_{i...j}(c(i)...c(j)) \tag{3.1}$$

Whereas β_0 denotes the base performance which refers to the time the system needs no matter the configuration, $\sum_{i \in \mathcal{O}} \phi_i(c(i))$ is the sum of each individuall feature and $\sum_{i...j \in \mathcal{O}} \Phi_{i...j}(c(i)...c(j))$ the sum of each feature interaction.

In Algorithm 1 we see a simple code snippet of some features influencing the runtime in different ways, in line 2 we define four features a,b,c and d, each of these features can be either True or False depending on the configuration selected. The code in line 5,7,9,11 will only be exectued if the respective features are active, if so the system sleeps for the specified ammount of time. In line 12 and 14 we have two feature interactions where as a and b need to be active for line 13 to be executed, we would attribute the time sepend in line 13 to the feature interaction a,b and not to one of the features alone. The performance-influence model for our system would look as follows:

$$\Pi = 1 + 1 \cdot c(a) + 2 \cdot c(b) + 1 \cdot c(c) + 2 \cdot c(d) + 2 \cdot c(a) \cdot c(b) + 0 \cdot c(c) \cdot c(d)$$

For simplicity we assume that it takes no time at all to execute the code and that no additional time is spend in a feature except the one stated in the *textitslee_for_seconds* function. The constant 1 here refers to β_0, which is the time we spend in our base feature in line 2. If we decide to turn on the features a,b and c the equation would look like this:

Algorithm 1 Feature Interaction

```
1:  procedure Feature Interaction
2:      a,b,c,d ← True or False
3:      sleep_for_seconds(1) #Lets the system sleep for 1 second
4:      if a then
5:          sleep_for_seconds(1)
6:      if b then
7:          sleep_for_seconds(2)
8:      if c then
9:          sleep_for_seconds(1)
10:     if d then
11:         sleep_for_seconds(2)
12:     if a and b then
13:         sleep_for_seconds(2)
14:     if c and d then
15:         sleep_for_seconds(0)
```

$$\Pi = 1 + 1 \cdot c(a) + 2 \cdot c(b) + 1 \cdot c(c) + 2 \cdot c(d) + 2 \cdot c(a) \cdot c(b) + 0 \cdot c(c) \cdot c(d)$$
$$\Pi = 1 + 1 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 + 2 \cdot 0 + 2 \cdot 1 \cdot 1 + 0 \cdot 1 \cdot 0$$
$$\Pi = 1 + 1 + 2 + 1 + 2 + 2$$
$$\Pi = 9$$

Therefore given the configuration a,b,c our performance-influence model would return a expected time of 9 second.

We create a performance-influence model from the date collected by both approaches, the challenge here lies in producing accurate data to create accurate model. Since we use both our approaches on the same data, we expect them to produce comparable performance-influence model, how we collect and work with the data we elaborate in chapter 4

4 BLACK-BOX MODEL

We speak of a black-box model when we the user are not aware of the inner workings of the system, like the name suggests we are in the dark on how the system works, for all we know is that given a input the black-box returns a output according to the systems specifications.

A well known example of a black-box system are neural networks, they start of with a problem and exampels on how to solve it, during several iterations, the network improves itself and is able to solve the problem more efficiently and accurately. While we know what problem the neural network is solving, if we would step into the code, we would not be able to understand the structure and reasoning behind it. For us humans, it is impossible to understand how all the neurons interact with each other and for what specific reason. [4].

However, since we want to compare black-box and white-box model, we need to do that on the same systems, so we cannot choose black-box system for our experiments, since the white-box model could not produce data in this case. We solve this problem by choosing a commonly available white-box system, such as xz, but even though we have acess to the code, we dont use that knowledge and look at the system from a black-box perspective.

4.1 DISADVANTAGES OF BLACK-BOX MODEL

When using the black-box model we encouter two major challanges, combinatorial explosion and collinear features, both need to be handled carefully in order to build an accurate performance-influence model.

COMBINATORIAL EXPLOSION One of the larger problems we face when using black-box models is the issue of combinatorial explosion, which refers to the effect that when features increase linearly, the number of possible configuration and thus system variants, increase exponentially [3].

Suppose we have a configurable system where each feature is a binary option that you can either turn on or off. We also define that in this system each feature is completely independent of another, i.e. the system has no constraints and turning a feature on or off has no affect on other features. The number of unique configurations this system can produce is 2^n, where 2 refers to the type of feature options allowed, binary in our case and n denotes the number of features. In Figure 4.1 we can see the effects of combinatorial explosion in such a system, since our system provides 15 features, we can see that the number of unique configurations equal to 32768.

Here is the problem, because 15 features is by no means a lot compared to the ammount of functions a real world systems like the linux kernel contains. All these different features might interact with each other in different ways, and for very small systems, we can certainly brute-force our way by benchmarking all possible configuration, however this does not scale, it is already not feasible for systems that have more then 16 features and therefore over 100.000 configurations. For this reason, we cannot fully expore the entire configuration

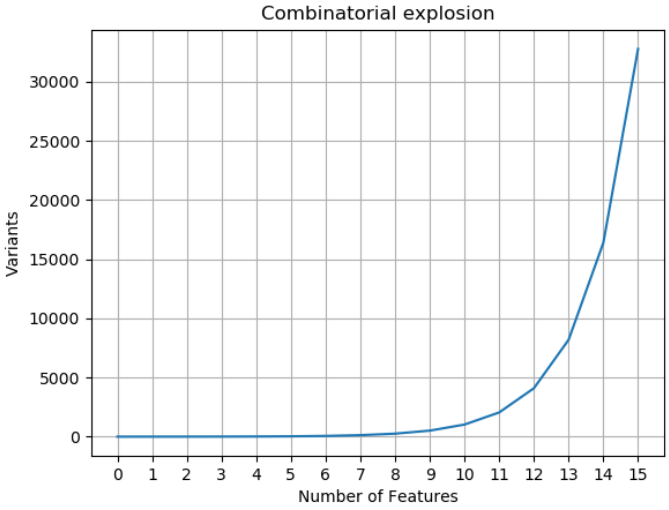

Figure 4.1: Combinatorial Explosion Graph

space and therefore need to select a subset that represent the system with a high accuracy. To do that state of the art black-box model use sampling strategies to find a suitable subset, such as, pair-wise sampling, most-enabled-disabled and random sampling. A in depht comparison between state of the art sampling strategies has been done by Medeiros et.al [8].

COLINEAR FEATURES Another problem for our black-box model arises when we have features that are correlated with each other, we call such features collinear. Let's make an example and revisit our code from Chapter 3 Algorithm 1 and let's extend it with the condition $c \leftrightarrow d$. We add this new condition from Algorithm 2 in Algorithm 1 after line 2.

Algorithm 2 Colinear Features

1: **if** $(c\ and\ \neg d) or (\neg c\ and\ d)$ **then**
2: $\quad c, d \leftarrow False$

Now, if our configuration does not contain c and d, the code in line 9 and 11 is unreachable. These colinear features make it difficult for our black-box model to correctly assign time to each feature, because if we are unaware of the inner workings, we might mistakenly think that the increase in time when feature c and d are activated is due to the interaction between them, but in fact we see in line 15 that they do not interact at all.

4.2 COLLECTING DATA

Let us now proceed on how we collect data. First, we analyze the configuration space using our feature model introduced in Chapter 2 and decide on the set of features we want to analyze. To deal with the effects of combinatorial explosion, we select a subset of the entire configuration space and analyze it using a brute force approach, rather using sampling strategies which would exceed the scope of this thesis.

In addition, we will only use valid configurations that satisfy the constraints of our feature model, to mitigate the influence that colinear features have or our model.

4.3 CREATING A PERFORMANCE-INFLUENCE MODEL USING MULTIPLE LINEAR REGRESSION

We use our black-box model by feeding it inputs and meassuring the time it takes the system to produce a output. From this very limited set of data, we now need to bulid our performance-influence model, which we do by using multiple linear regression. We use multiple linear regression beacause the formular we learn can be understood by us humans, unlike other approaches like neural networks. Moreover, to use multiple linear regression, we do not need to know anything about the inner workings of the system, which benefits our black-box model. [5]

We use the following formular of linear regression for matrices[5]:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 ... \beta_n x_n + \epsilon$$
$$y = X\beta + \epsilon$$

Where y is called dependendent variable and X is called independent variable y in our case is a vector with n elements containing the output of our black-box model, i.e. the time meassurement generated for each feature configuration in our feature set. What we are intrested in are the values of the coefficents β, since they quantify the influence of each feature or feature interaction on to the whole system. In addition, β_0 denotes the intercept, which for us represents the influence of the base code, meaning the part of the code that gets exectued regardless of the selected feature configuration. Our independent variable X is a $n \times m$ martrix, where n is the number of configurations used and m the powerset of all features. The reason we use a powerset here is to model all feature interactions in our matrix. The value of each feature in the matrix is either 1 if the feature is selected, or 0 if its not. If we have numerical features with l different options, we split this features into l binary features and encode them as an alternative group in our feature model. For all our meassurements we also have a possible error represented by ϵ. [6]

5

BACKGROUND

All relevant core information that is needed to understand this thesis should be explained in the background.

6 EXAMPLE CHAPTER

This chapter gives you some examples how to include graphics, create tables, or include code listings. Examples on how to cite papers from the literature can be found in Chapter 9.

6.1 ACRONYMS

This template makes advantage of the acronyms package to support acronyms. The first occurence of an acronym is replaced by its definition (e.g., Integrated Development Environment (IDE)). All other occurences are replaced by the acronym (IDE). The glossaries package also supports plural—IDEs.

6.2 GRAPHICS

In Figure 6.1, we give a small example how to insert and reference a figure.

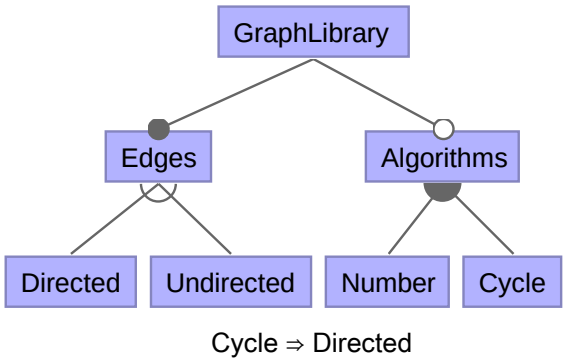

Figure 6.1: A feature model representing a graph product line

6.3 TABLES

Table 6.1 shows the result of a simple tabular environment.

Table 6.1: Mapping a feature model to a propositional formula

Group Type	Propositional Formula
And	$(P \Rightarrow C_{k_1} \wedge \ldots \wedge C_{k_m}) \wedge (C_1 \vee \ldots \vee C_n \Rightarrow P)$
Or	$P = C_1 \vee \ldots \vee C_n$
Alternative	$(P = C_1 \vee \ldots \vee C_n) \wedge \text{atmost1}(C_1, \ldots, C_n)$

6.4 CODE LISTINGS

In Listing 6.1, we give an example of a source code listing.

Listing 6.1: Java source code

```
class A extends Object {
        A() { super(); }
}
class B extends Object {
        B() { super(); }
}
class Pair extends Object {
        Object fst;
        Object snd;
        Pair(Object fst, Object snd) {
                super(); this.fst=fst; this.snd=snd;
        }
        Pair setfst(Object newfst) {
                return new Pair(newfst, this.snd);
        }
}
```

7 EXPERIMENTS

This chapter describes the main experiments that are conducted for this thesis.

8 EVALUATION

This chapter evaluates the thesis core claims.

8.1 RESULTS

In this section, present the results of your thesis.

8.2 DISCUSSION

In this section, discuss your results.

8.3 THREATS TO VALIDITY

In this section, discuss the threats to internal and external validity.

9 RELATED WORK

This chapter presents related work.

For example, **KG:SME06** investigated ...

ABKS:BOOK2013 analyzed ...

In earlier work [**KAK:GPCE09**, **ABKS:BOOK2013**], they have shown ...

10

CONCLUDING REMARKS

A

APPENDIX

This is the Appendix. Add further sections for your appendices here.

BIBLIOGRAPHY

[1] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-37521-7. URL: https://doi.org/10.1007/978-3-642-37521-7.

[2] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer Berlin Heidelberg, 2013, pp. 26–36. DOI: 10.1007/978-3-642-37521-7. URL: https://doi.org/10.1007/978-3-642-37521-7.

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer Berlin Heidelberg, 2013, pp. 243–244. DOI: 10.1007/978-3-642-37521-7. URL: https://doi.org/10.1007/978-3-642-37521-7.

[4] Chris M. Bishop. "Neural networks and their applications." In: *Review of Scientific Instruments* 65.6 (1994), pp. 1803–1832. DOI: 10.1063/1.1144830. eprint: https://doi.org/10.1063/1.1144830. URL: https://doi.org/10.1063/1.1144830.

[5] Alexander Grebhahn, Norbert Siegmund, and Sven Apel. "Predicting Performance of Software Configurations: There is no Silver Bullet." In: *CoRR* abs/1911.12643 (2019). arXiv: 1911.12643. URL: http://arxiv.org/abs/1911.12643.

[6] Jürgen Groß. *Linear Regression*. Springer Berlin Heidelberg, 2003. DOI: 10.1007/978-3-642-55864-1. URL: https://doi.org/10.1007/978-3-642-55864-1.

[7] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. CMU/SEI-90-TR-021. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990, pp. 35–37. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231.

[8] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. "A comparison of 10 sampling algorithms for configurable systems." In: *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. Ed. by Laura K. Dillon, Willem Visser, and Laurie A. Williams. ACM, 2016, pp. 643–654. DOI: 10.1145/2884781.2884793. URL: https://doi.org/10.1145/2884781.2884793.

[9] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. "Performance-influence models for highly configurable systems." In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. Ed. by Elisabetta Di Nitto, Mark Harman, and Patrick Heymans. ACM, 2015, pp. 284–294. DOI: 10.1145/2786805.2786845. URL: https://doi.org/10.1145/2786805.2786845.