Improved Prediction of Non-functional Properties in Software Product Lines with Domain Context

Max Lillack[a], Johannes Müller[b], Ulrich W. Eisenecker[a]

[a] {lillack, eisenecker}@wifa.uni-leipzig.de, [b] jmue933@aucklanduni.ac.nz

Abstract: Software Product Lines (SPLs) enable software reuse by systematically managing commonalities and variability. Usually, commonalities and variability are expressed by features. Functional requirements of a software product are met by selecting appropriate features. However, selecting features also influences non-functional properties. To satisfy non-functional requirements of a software product, as well, the effect of a feature selection on non-functional properties has to be known. Often an SPL allows a vast number of valid products, which renders a test of non-functional properties on the basis of all valid products impractical. Recent research offers a solution to this problem: the effect of features on non-functional properties of software products is predicted by measuring in advance. A sample of feature configurations is generated, executed with a predefined benchmark, and then non-functional properties are measured. Based on these data a model is created that allows to predict non-functional properties of a software product before actually building it. However, in some domains contextual influences, such as input data, can heavily affect non-functional properties. We argue that the measurement of the effect of features on non-functional properties can be drastically improved by considering contextual influences of a domain. We study this assumption on input data as an example for a contextual influence and using an artificial but intuitive case study from the domain of compression algorithms. Our study shows that prediction accuracy of non-functional properties can be significantly improved.

1 Introduction

Software Product Line Engineering (SPLE) allows to leverage the reusability of assets and to efficiently develop new products of a common domain on the basis of an existing reuse infrastructure. It comprises at least two core processes, domain engineering (development for reuse) and application engineering (development with reuse). The set of products realized by these processes is a Software Product Line (SPL) [CN07].

Commonalities and variability of a domain are usually documented with variability models of which feature models are the most common [CGR$^+$12]. In feature models commonalities and variability of a domain are described in terms of features. A feature is any aspect of a software product that is relevant to at least one stakeholder [CE00]. Usually, the main focus of feature models is to describe the functional properties of an SPL [CHS08].

A valid combination of features describing a software product is called a configuration. During the configuration process a user of a feature model (e.g. a developer) can select certain features according to her requirements. With the selection of a feature the remaining variability is reduced. The resulting configuration describes the functional properties of a valid product of the SPL.

Non-functional properties such as performance, resource usage and scalability can be just as important to users as functional properties. They are an important part of user requirements and have to be taken into account as well. However, from feature selection alone it is not apparent which non-functional properties a resulting software product will have. Thus, resulting software products have to be measured to assess their non-functional properties.

In SPLE it is not sufficient to assess the non-functional properties of a single product, rather, all products of an SPL have to be considered. In case of a high number of possible products it is practically impossible to measure every product on its own with methods from traditional software engineering. A solution to this problem, suggested by recent research (cf. Section 2), is to measure a sample of possible configurations of an SPL with a benchmark to predict the impact of features on non-functional properties of products not built before. Since presence or absence of feature combinations can affect the impact of features on non-functional properties, the measurements have to cover such *feature interactions* as well. The result of this process is a *performance model* which maps the set of configurations to non-functional properties of the resulting products [TP12]. For each non-functional property, the performance model contains the impact of each feature, their interaction with other features and a function to aggregate these values.

Products of an SPL are often executed in different runtime environments and exposed to different types of input data, which could influence their behavior. We call these influences *contextual influences*. Our hypothesis is that contextual influences can have an impact on the effect of features on non-functional properties of the resulting products.

In this case, including contextual influences into the measurement of non-functional properties can increase the precision of their prediction. In the following, we report on the results of a rather artificial but intuitive case study from the domain of compression algorithms with input data as one example of contextual influences.

The objective of this study is, first, to demonstrate how to incorporate contextual influences into the measurement of non-functional properties (Section 4) and, second, to test our hypothesis and to see whether considering contextual influences can be useful at all (Section 5). For our study we used synthetic input data to measure their effects isolated from other effects. This should allow to observe clearly whether contextual influences can have an effect and whether more extensive studies with real-world SPLs are worthwhile. A conclusion in Section 6 wraps up the paper and provides an outlook to future work.

2 Related Work

Within the last couple of years a reasonable amount of work on modeling, estimating and predicting non-functional properties for SPLs has been published.

Some work recognizes that contextual influences can have an effect on non-functional properties [KR10, HBR$^+$10]. However, their focus is on distinct properties of contextual influences, whereas we argue that the variability of contextual influences in a specific domain has to be considered to improve the precision of estimated performance models.

Besides work which discusses contextual influences on non-functional properties of software products, there is also work concerned with approaches to capture and model information about non-functional properties. A process to model non-functional properties is presented by Tawhid et al. [TP12]. They consider UML profiles for modeling non-functional properties. On that basis predictions about non-functional properties are possible. However, their approach to gather information about non-functional properties is neither automatic nor does it consider contextual influences.

Sincero et al. discuss that information about non-functional properties can enrich the configuration process of SPLs [SSPS10]. They present an approach to detect general influences of features on non-functional properties, but do not build a performance model on top of the gathered data. The approach is demonstrated with a subset of the features of the Linux kernel without considering any contextual influences on non-functional properties.

Recent work of Siegmund et al. [SKK$^+$12] and Guo et al. [GCA$^+$12] specifically focuses on building performance models for SPLs. Siegmund et al. present a process from gathering data to estimating a performance model and discuss techniques for the involved steps. The discussed technique for estimating the performance model is based on linear regression. Guo et al. follow a slightly different approach by applying classification and regression trees (CARTs). Both do not consider the effect of contextual influences on non-functional properties.

Most of the related work creates performance models based on functional features. They may therefore only be valid within the same context in which they were created. Since Siegmund et al. present the most comprehensive approach, we are going to extend their approach to consider contextual influences in estimating performance models for SPLs.

3 Background: Predicting Non-Functional Properties in SPLs

In their approach [SKK$^+$12, SRK$^+$12] Siegmund et al. select a subset of possible configurations of an SPL, generate corresponding products and measure their non-functional properties. They use the resulting sample to calculate the impact of each feature on non-functional properties. As a basic requirement the selection of configurations has to allow the isolation of each feature's effect. With the isolated effects of the features it is possible to predict non-functional properties for an arbitrary configuration by aggregating the effects of the containing features.

However, sometimes features do not only have an isolated effect on non-functional properties but interact with other features. In such cases the effect of one feature changes if an interacting feature is contained in a configuration, too. A performance model has to take feature interactions into account in order to realize sufficiently high prediction accuracy.

Feature interactions can be of different orders, and different techniques have to be applied to detect them.

A pairwise interaction (first order interaction) occurs if the presence of two features generates a different effect than the combined individual effects of the two features. For each feature there is a high number of other features with which they could possibly interact. To reduce the number of measurements, Siegmund et al. suggest the following heuristic: calculate a configuration with as many features as possible, measure the effect on a non-functional property and compare it with the expected effect on the non-functional property from the aggregated isolated effects of the contained features. If the measured effect does not deviate from the expected effect, the absence of a feature interaction is assumed. Otherwise, pairs of interacting features have to be identified.

This is done with the pairwise covering heuristic [POS+11]. The heuristic includes as many configurations as required to cover all combinations of two features. A measurement of a feature interaction between features A and B can be skipped if A implies B [SKK+12]. The measurement of A alone will always include the possible interaction effect. Such constraints reduce the set of measurements.

Second order interactions are present if the presence of three features generate another effect than the presence of only one or two of them. A high number of measurements are required to cover all possible second order interactions. Siegmund et al. consider second order interactions only if pairwise interactions exist between each of the three possible features. Based on the assumption that few features interact with many others, they also test features, which already have taken part in many pairwise interactions for second order interactions. Although higher order interactions are possible, they are not considered to limit the number of measurements and hence make the approach practicable [SRK+12].

The approach is extensively evaluated with case studies using SPLs from different domains (e.g. database, compiler and video encoding) and different non-functional properties (e.g. footprint and performance) [SKK+12, SRK+12]. The performance model is based on standard benchmarks applicable to the domains of the analyzed SPLs. The results show a high prediction accuracy in the evaluation with data of the utilized standard benchmarks.

4 Input Data in Predicting Non-functional Properties

Our hypothesis is: considering input data in the measurement can improve the prediction of non-functional properties. To study this, we will first have to clarify how to consider input data in the measurement of non-functional properties. Three questions will have to be answered. First, how to capture variability of input data of a domain. Second, how to link the variability of input data to the variability of an SPL. Third, how to measure the effect of input data on non-functional properties.

A common tool to capture the variability of an SPL is feature modeling. Since feature models are essentially designed to capture the variability of any relevant concept [CE00] they are useful to model the variability of possible input data of a domain as well.

Feature models consist of features which are hierarchically modeled in feature diagrams.

An example of a feature diagram can be found in our case study in Figure 3. The root of the hierarchy is one concept node. It represents all possible products of an SPL. It is refined with subsequent nodes, the sub-features. These sub-features describe common and variable parts of the products of an SPL. Features can either be solitary or grouped. Mandatory solitary features are present in every product containing the parent feature.

Since features of input data describe a different concern than features of an SPL, it is natural to model both concerns separately. This simplifies the modeling of the input data features and prevents the SPL feature model from being bloated. Both models can easily be related, for example, by feature model references and transparently processed afterwards.

The variability of SPLs is documented in feature models by a domain analyst during the analysis phase of domain engineering. In the same way the domain analyst can capture the variability of input data.

As with functional features, identifying relevant data features and their appropriate abstraction is a problem of the domain engineering phase. For example, domain analysis can show the size of the input data as an important property. As it is not feasible to model every possible size as a single feature, clustering can be applied to get a feature *Size* with sub-features such as *10kB* and *20MB* in an *or-group* as representatives for a class of input sizes.

During the configuration process features are selected, based on user requirements. It is not necessary to distinguish between functional features and data features because interactions between them are possible. E.g. selecting a data feature could require selecting a functional feature.

By capturing the variability of input data in a feature model, the adaptation of the approach of Siegmund et al. is straightforward. The variability caused by contextual influences can be treated as the variability of the SPL. The performance model has to be extended so that it does not only contain the estimates of the effects of the functional features but also estimates for the effects of the input data features. However, to actually measure the effect of input data on non-functional properties the measurement environment has to be extended so that it generates test data according to the selected features of the input data feature model.

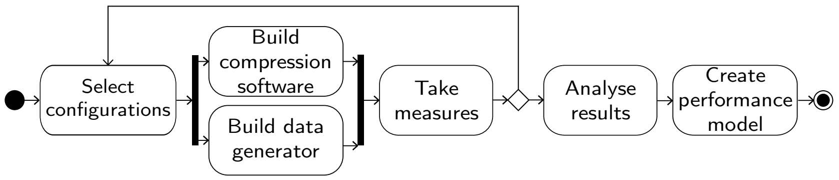

Figure 1: Activity diagram of measurement process

With that extension our measurement environment works according to the process depicted in Figure 1. The core of the measurement environment is a generator for products of the SPL, which are measured afterwards.

In contrast to the approach of Siegmund et al. [SKK$^+$12] we do not consider fixed input

data but generate different types of test data for different measurements. The products are executed with the test data and relevant non-functional properties are measured.
After all required measurements are collected they are analyzed and the performance model is created. A concrete implementation of this process is briefly described in Section 5.

For large SPLs it is infeasible to test every possible configuration because every test run needs time and other limited resources. Different configuration sets are built to isolate the effect of a single feature or a feature interaction from a complete configuration.

Based on the data produced by the measurement process, a performance model to predict non-functional properties of any configuration is constructed. We use a linear regression model as performance model. The model includes the effect of single features as well as interactions among any kind of features up to second order interactions. Features and their interactions are coded as binary variables (i.e. *dummy variables*) with the value 1 (present) or 0 (absent).

Statistical tools such as R[1] can be used to create the regression model with dummy coding. The regression model includes all features and valid feature interactions which can be identified by automatic analysis of feature models [BSRC10].

Siegmund et al. introduced a heuristic, where three-wise interactions are considered for three features which interact pairwise [SKK+12]. This heuristic requires the reliable identification of pairwise interactions and assumes a low number of second order interactions. Depending on the structure of a feature model, a complete pairwise interaction detection could require a large amount of measurements.

We use a heuristic which increases the number of measurements of *interesting* features, i.e. features whose estimated direct effect or pairwise interaction effects are rather extreme. Based on the measurements of the pairwise set, we select results which exceed two predefined quantiles (e.g. 15% and 85% quantiles). Additional measurements are selected by replacing a sub-feature of a parent feature (e.g. in an or-group) with the other features of the same group holding the other features of the configuration constant. This heuristic is specific for each non-functional property.

With only little or no domain knowledge the heuristic can be adjusted to the requirements of the measurement environment. By adjusting the threshold set by the quantiles the number of measurements can be regulated. A higher number of configurations increases the time for measurements but improves the prediction quality. A second parameter is the selection of features to vary, the more features are fixed the less configurations need to be measured.

5 Study

In our study we test the hypothesis: considering input data can increase the precision of a performance model. Furthermore, we demonstrate our approach. In order to see the effect

[1]http://www.r-project.org

of input data on non-functional properties isolated from other effects we have designed an artificial SPL with lossless compression algorithms. The artificial nature of the study does limit its expressiveness but increases the clarity of the results. So, even if the study does not proof our hypothesis, its results provide a first indication whether our hypothesis should be further investigated with real-world SPLs.

We designed *Compressor SPL* (Figure 2) based on an already implemented compilation of compression tools[2] and an input data feature model (Figure 3). The Compressor SPL is similar to the *ZipMe* SPL [AKL12] but realizes more variants. The input data feature model comprises variability of input data, which is relevant for non-functional properties.

We considered three exemplary non-functional properties of lossless compression algorithms in the study: *compression time*, *memory usage* and *compression ratio*. Compression time is the time to finish one compression task with pre-specified input data. Memory usage is the amount of memory required during the compression. Compression ratio is the ratio between the size of the compressed data and the size of the input data.

We selected those three properties since they can be relevant in a decision for a specific compression algorithm. Though a low compression ratio is clearly desirable, a user can sacrifice ratio for improved speed (e.g. in time-critical applications) or lower memory consumption (e.g. in embedded systems). However, the selection of relevant properties solely relies on the addressed domain and has to be done in the domain analysis phase.

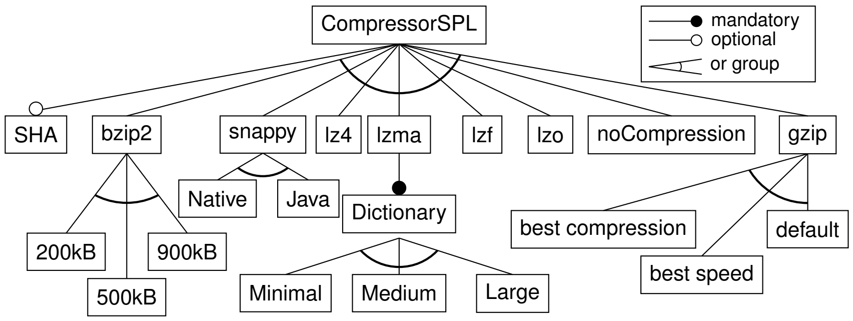

Figure 2: Feature Model of Compressor SPL

Figure 2 depicts the feature model of the Compressor SPL. It is basically a selection of different compression algorithms, implementations and configuration options. For example, the algorithm *bzip2* can be configured with different block sizes of which *200kB*, *500kB* and *900kB* are used as representative features. The block size is a popular user-configurable parameter which can have a great influence on performance [Sal07].

As another example the *snappy*[3] algorithm can have the feature *Native* or the feature *Java* to compare different implementations of the same algorithm.

Additionally there is an optional feature *SHA* to calculate a checksum of the input data using the SHA-256 hash function shared by the algorithms.

[2]https://github.com/ning/jvm-compressor-benchmark

[3]http://code.google.com/p/snappy

Figure 3 depicts the feature model of the input data of the domain. The model contains 14 features. It comprises two main features *Size* and *Content*. *Size* has six sub-features rep-

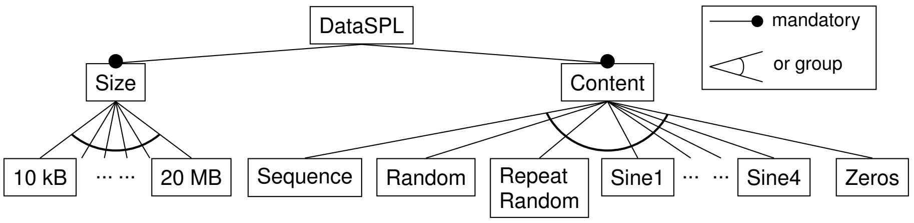

Figure 3: Feature Model of Data SPL

resenting different input data sizes from 10kB up to 20MB. The feature *Content* describes what kind of data is generated. It has several sub-features which represent prototypical structures of input data, which are differently difficult to compress. A *Sequence* is input data that contain sequentially changed bytes. The first thousand bytes are zeros, the next thousand bytes are ones and so on. *Random* is a sequence of random bytes. As a variant of *Random RepeatRandom* repeats small chunks of random bytes. *Sine1* to *Sine4* create non-trivial sequences based on different sine waves. *Zeros* contains only zeros.

For a transparent handling of the *Compressor SPL* feature model and the *Data SPL* feature model, we joined both in an artificial feature model, *Measurement SPL*, which describes the variability of the domain. For the measurements we needed to look not only at the compression software but also at the complete product including the input data. Considering the whole SPL we have many differences and commonalties leading to the possibility of first and second order interactions. To efficiently estimate the influence of such interactions common heuristics such as the pairwise interaction heuristic can be applied.

We used *FeatureHouse* [AKL12] to implement a generator for the *Measurement SPL*. The generator generates test data and compression programs for any sample. To configure, execute and control measurement runs we used *Apache Maven*[4] and the *Jenkins*[5] continuous integration server. This environment allows flexible and fully automatic measurement runs.

5.1 Methodology

To see whether considering input data in estimating a performance model can improve the precision of the estimated performance model we have to compare the precision of a performance model estimated with and without input data consideration.

For that we first estimated a performance model without considering input data. Second, we estimated a performance model considering input data. Finally, we applied both performance models on the three sets of configurations given by our heuristics and compared the

[4]http://maven.apache.org
[5]http://jenkins-ci.org

estimated non-functional properties (x) with the realized non-functional properties. These sets differ in their sizes and their ability to infer feature interactions. The deviation of the predicted value (p) to the realized value (r) is the *prediction error* (e)

$$e = |x_p - x_r|.$$

We use absolute differences, as they are easier to interpret and more robust for small values than relative differences. We compared the prediction errors of the performance model without input data (w/o) and the performance model with input data (w)

$$\Delta e = e_{w/o} - e_w.$$

A positive value means that the error of the performance model with input data is smaller than the performance model without input data. We tested the significance of this difference using the non-parametric Wilcoxon signed-rank test according to the test hypotheses:

H0 The prediction error of the model estimated with input data is equal or higher than the prediction error of the model estimated without input data.

H1 The prediction error of the model estimated with input data is less than the prediction error of the model estimated without input data.

We performed the measurement runs on a desktop PC with an Intel Core 2 Duo 3.0 GHz processor, 6 GB RAM and Windows 7 as operating system installed. Our Compressor SPL relies mostly on Java for which we used the Oracle Java Virtual Machine 1.7.

For this rather small case study it was possible to measure every configuration within a few hours on our experimental setup to create the performance models. However, as we will see later, good prediction can already be reached with a small subset of measured configurations. This aspect is important for the scalability of the proposed solution and is therefore explicitly addressed in our study.

We generated the performance models on the basis of three different sets of sample configurations. The *minimal* configuration set contains enough configurations to use regression analysis to estimate the impact of each feature. The *pairwise* configuration set extends the minimal configuration set to include additional configurations to observe interactions between two features. The *three-wise* configuration set extends the pairwise configuration set to observe interactions between three features.

Which configurations are part of the respective sets had to be calculated. Johansen et al. [JHF12] developed algorithms and tools to calculate the set of pairwise configurations which we used to build the set in our case study. The test of all possible three-wise interactions would require a high number of measurements. Hence, heuristics had to be applied to reduce the number of required configurations. We applied our three-wise heuristic described in Section 4 to calculate configurations in order to identify important second order interactions.

The measurement of compression ratio and memory usage is deterministic. Therefore, we were not faced by any threat of biased measurements for these non-functional properties.

However, the non-functional property compression time is vulnerable for biased measurement, which we tried to reduce by several strategies. First, we had a warm up phase, in which we ran the configuration twice without taking time to allow the virtual machine to apply optimizations. Then, we ran the test again four times and took the average runtime as our result. A look at the standard deviation of the four measurements showed that we got stable results. Second, we executed the performance measurements as the exclusive operation in the execution environment. Third, we set the memory of the virtual machine large enough to prevent disturbing effects from the Garbage Collector and executed all operations in memory so that disk or network I/O will also produce no disturbing effects.

5.2 Results

In the next section we present the results of our study separated by the measured non-functional properties. First, we shortly discuss the different sizes of the configuration sets, which are required to apply the discussed interaction detection approaches. Second, we report for all three considered non-functional properties the prediction errors $\overline{e}$ for both performance models and their differences $\overline{\Delta e}$.

The required sizes of the configuration sets are reported in Table 1. To evaluate the impact of every feature (including data features) a *minimal* set of 28 configurations is required. The *pairwise heuristic* already requires a total number of 150 measurements.

Table 1: Size of the configuration sets

Configuration set	Size	Relative size
Minimal	28	0.02
Pairwise	150	0.1
Three-wise (heuristic)	349 - 443	0.24 - 0.31
Three-wise (complete)	729	0.51
All	1440	1

The *three-wise heuristic* leads to 349 measurements for ratio (443 for time and 372 for memory), which equals 24% (31%, 26%) of all possible configurations. The configuration set of the three-wise heuristic is a superset of the pairwise set. It contains 199 to 293 *additional* configurations compared to the pairwise set. The difference between both sets cannot be generalized as it depends on the feature model as well as on the results of the pairwise measurements.

The three-wise heuristic configuration sets are specific for each non-functional property. The more non-functional properties are considered in a study the more configurations are required to identify three-wise interactions for the considered non-functional properties. However, the size of the sets does not grow linearly because the sets share a large amount of configurations (between 56% and 61% of the configurations).

As shown in Table 1 only 2% of all possible configurations are enough to create a performance model which considers direct effects of features on non-functional properties. The more interaction effects have to be captured the more configurations are required. In case of the three-wise heuristic nearly half of the possible configurations had to be measured. Compared to the fact that all possible configurations have to be considered without the use of this heuristic, this is still a good improvement.

Based on the generated configuration sets, we built performance models with and without considering variability in the input data. The results of our evaluation are summarized in the Tables 2 – 4.

Table 2: Regression for compression ratio

	Without input data		With input data			
Configuration set	$\overline{e}$	σ	$\overline{e}$	σ	$\overline{\Delta e}$	p_{wrt}
Minimal	0.91	0.50	0.09	0.25	0.82^{***}	<0.001
Pairwise	0.35	0.33	0.07	0.19	0.28^{***}	<0.001
Three-wise heuristic	0.30	0.35	0.05	0.15	0.25^{***}	<0.001

*** $p < 0.001$

Table 2 shows the results of the performance models for compression ratio. The prediction accuracy has greatly improved once data features were considered. Even with the minimal set the prediction quality can be improved to a much better but still not perfect average error of 0.09 compared to the average error of 0.91 without input data. The three-wise heuristic with input data gives the best results with an average error of 0.05 for the performance model with input data compared to an average error of 0.30 for the performance model without input data. For all configuration sets the study showed a significant improvement by considering input data.

Table 3: Regression for memory usage (in kB)

	Without input data		With input data			
Configuration set	$\overline{e}$	σ	$\overline{e}$	σ	$\overline{\Delta e}$	p_{wrt}
Minimal	507.2	2313.53	545.00	2366.67	-37.78	1
Pairwise	497.70	2057.33	47.53	183.27	450.17^{***}	<0.001
Three-wise heuristic	496.46	2052.99	47.43	188.22	449.03^{***}	<0.001

*** $p < 0.001$

Table 3 shows the prediction of the non-functional property memory usage. For the minimal configuration set the prediction with input data cannot be assumed to be better than without input data. However, the more information is provided by the considered sample the better the predictions of the model with input data get. This is not the case for the performance model without input data. Even with many more measurements the prediction error has only slightly improved. The model without input data is not able to consider a major influence on the dependent variable. For the larger configuration sets the precision model with input data is significantly more precise than the model without input data.

Table 4 provides a different image than the results of previous non-functional properties. In the minimal and pairwise configuration sets the performance model with input data leads to a higher prediction error than the simpler model without input data. The direct and pairwise interaction effects provide no good explanation of the observed values. The two smaller configuration sets provide no information of second order interactions. The direct and pairwise effects are overemphasized. Only if second order interactions are considered with the three-wise heuristic the model with input data provides results which are significantly better than the model without input data.

Table 4: Regression for compression time (in seconds)

	Without input data		With input data			
Configuration set	$\bar{e}$	σ	$\bar{e}$	σ	$\overline{\Delta e}$	p_{wrt}
Minimal	4.20	24.54	4.58	23.96	-0.38	1
Pairwise	6.42	23.91	12.57	38.41	-6.15	1
Three-wise heuristic	4.22	24.55	1.62	12.78	2.60^{***}	<0.001

$^{***}\ p < 0.001$

5.3 Discussion

For all three non-functional properties we can show significant improvements compared to respective models without input data. In case of compression time the performance model with input data needs a three-wise heuristic configuration set to achieve significantly better results. This suggests that some second order interactions between the features of the SPL have a large influence on compression time and have to be considered to achieve precise prediction results. Our study showed that the shape of the input data has a significant effect on non-functional properties of the Compressor SPL. Based on our study we can assume that in some domains contextual influences can have a noticeable effect on non-functional properties.

The measurement effort is higher if input data is considered as an additional source of variability than without it. Using heuristics in the selection of configurations mitigates this effect.

However, the results should be seen under the light of some possible threats to their validity. The measurement of performance metrics can be subject to a variety of errors. We only use these measurements for comparison and not the absolute values so that they should not affect our conclusion. The presented predictions still have errors. The SPL contains pairwise and second order interactions of which not all are covered by our prediction model. This has to be seen as a trade off between measurement effort and accuracy.

The used SPL was especially created for this study and is designed to make the interaction between data features and features highly visible. The selected data features represent extreme values within the range of possible realistic values. For example, completely randomized data on the one extreme and data of only zeros on the other extreme will show extreme behavior of the compression ratio. Nevertheless patterns of the modeled data will also appear in real data, yet, not in pure form but in combination with other patterns of data. This restricts the generalizability of our study but still gives a good indication that contextual influences can have an effect on non-functional properties.

Despite the fact that our case study is based on an artificial SPL we can see that input data can indeed have an effect on non-functional properties. The results of our study are somewhat extreme and cannot be extrapolated to arbitrary cases. However, we have shown that an effect can exist. To improve the generalizability of our findings the next step has to be a study with real-world SPLs.

6 Conclusion

With the example of input data we showed how contextual influences can be included into the measurement of non-functional properties of SPLs. We applied our approach in a synthetic case study from the domain of lossless compression algorithms to study whether input data has an effect on three instances of non-functional properties.

Our study shows that input data has an effect on the precision of measured performance models. We see this as an indicator that considering contextual influences can have a positive effect on the precision of performance models. This is a prerequisite for future work to validate the approach using realistic SPLs. A natural choice of study candidates are the publicly available and realistic SPLs described by Siegmund et al. [SKK$^+$12, SRK$^+$12].

To conclude, in our synthetic case study we found that contextual influences can improve prediction of non-functional properties. This is a first indication that the accuracy of prediction models can be improved if contextual influences are considered. It is therefore worthwhile to carry out further extensive case studies with real-world SPLs.

References

[AKL12] Sven Apel, Christian Kästner, and Christian Lengauer. Language-Independent and Automated Software Composition: The FeatureHouse Experience. *IEEE Transactions on Software Engineering (TSE)*, 2012.

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems*, 35(6):615–636, 2010.

[CE00] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, USA, 2000.

[CGR$^+$12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wasowski. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 173–182, New York, NY, USA, 2012. ACM.

[CHS08] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What's in a Feature: A Requirements Engineering Perspective. In *Proceedings of the Theory and Practice of Software, 11th International Conference on Fundamental Approaches to Software Engineering*, FASE'08/ETAPS'08, pages 16–30, Berlin and Heidelberg, 2008. Springer.

[CN07] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, Boston, 6 edition, 2007.

[GCA$^+$12] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. Variability-Aware Performance Modeling: A Statistical Learning Approach. Tech. Rep. GSDLAB-TR 2012-08-18, Generative Software Development Laboratory, University of Waterloo, 2012.

[HBR$^+$10] Jens Happe, Steffen Becker, Christoph Rathfelder, Holger Friedrich, and Ralf H. Reussner. Parametric performance completions for model-driven performance prediction. *Performance Evaluation*, 67(8):694–716, 2010.

[JHF12] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. An Algorithm for Generating t-wise Covering Arrays from Large Feature Models. In *Proceedings of the 16th International Software Product Line Conference (SPLC 2012)*, Salvador and Brazil, 2012. ACM.

[KR10] Lucia Kapova and Ralf Reussner. Application of Advanced Model-Driven Techniques in Performance Engineering. In Alessandro Aldini, Marco Bernardo, Luciano Bononi, and Vittorio Cortellessa, editors, *Computer Performance Engineering*, volume 6342 of *LNCS*, pages 17–36. Springer, Berlin and Heidelberg, 2010.

[POS$^+$11] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Pairwise Testing for Software Product Lines: A Comparison of Two Approaches. *Software Quality Journal*, 2011.

[Sal07] David Salomon. *Data Compression*. Springer, New York, NY, USA, 2007.

[SKK$^+$12] Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting Performance via Automated Feature-Interaction Detection. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2012.

[SRK$^+$12] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3):487–517, 2012.

[SSPS10] Julio Sincero, Wolfgang Schröder-Preikschat, and Olaf Spinczyk. Approaching Non-functional Properties of Software Product Lines: Learning from Products. In *Proceedings of the Asia Pacific Software Engineering Conference*, APSEC 2010, pages 147–155. IEEE Computer Society, 2010.

[TP12] Rasha Tawhid and Dorina Petriu. User-friendly approach for handling performance parameters during predictive software performance engineering. In *Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering*, ICPE '12, pages 109–120, New York, NY, USA, 2012. ACM.