Hyper Explanations for Feature-Model Defect Analysis

Marc Hentze
marc.hentze1@volkswagen.de
Volkswagen Aktiengesellschaft
Germany

Tobias Pett
t.pett@tu-braunschweig.com
Technische Universität Braunschweig
Germany

Thomas Thüm
thomas.thuem@uni-ulm.de
Universität Ulm
Germany

Ina Schaefer
i.schaefer@tu-braunschweig.com
Technische Universität Braunschweig
Germany

ABSTRACT

Proprietary formats, missing analysis tools, the lack of continuous toolchains and their complexity itself impair the maintainability of industrial variability models, making them prone to *defects*. Also, automated analysis of variability models is still not common in industry. To gain detailed information about the defects in a variability model, it can be converted into a standardized *feature model* to apply automated analyses that expose present defects. However, resolving those defects can be challenging, as their cause can be complex, especially for large feature models. To mitigate this, an *explanation* can be generated which identifies feature model parts that are involved in a specific defect. Although those explanations provide valuable information, handling a high number of defects at the same time remains a tedious task, as there is no prioritization that states which defect is to be handled first to achieve the best progress.

In this paper, we propose a concept to automatically derive that prioritization based on *hyper explanations*, which are generated by aggregating the information provided by the explanations of single defects. Hyper Explanations allow to derive a prioritization not only for defects but also for defect-creating constraints. We applied our concept to industrial feature models and discussed the results with domain experts, which led to an unexpected conclusion: The models contain *intentionally* created dead features. We further evaluate the usage of intentionally dead features in industry and discuss how to handle them together with actual defects.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Software defect analysis*; *Feature interaction*.

KEYWORDS

feature model defects, defect explanations, hyper explanations, prioritization of defects, disabled features

ACM Reference Format:
Marc Hentze, Tobias Pett, Thomas Thüm, and Ina Schaefer. 2018. Hyper Explanations for Feature-Model Defect Analysis. In *Proceedings of VaMoS '21: 15th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS '21).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Today, car manufacturers offer highly customizable vehicles that can be configured to satisfy the individual needs of their customers. Therefore, available configuration options need to be modeled along the constraints that define how those options can be combined. Despite the central relevance of the resulting variability model, its maintenance is subject to multiple aspects that make it prone to errors.

While the lack of a continuous toolchain affects its integrity, heterogeneous and proprietary formats [4] hinder the applicability of standard, automated analyses [1] [6] [16] and impair the model's verification. Moreover, simultaneous modifications of multiple stakeholders cause frequent changes, whose side effects are hard to capture. In addition, the model's complexity further increases its vulnerability to issues [4].

Hence, the resulting *variability models* contain high numbers of errors, which are classified into different kinds of *defects* [5]. Converting those models into standardized *feature models* [7] allows to use standard tooling for detailed analyses which expose those defects and provide valuable insights. By applying those analyses to existing variability models, we revealed high numbers of configuration options that cannot be selected and other defects. Although explanations [8] provide useful information about the cause of those defects, handling and analyzing each individual explanation in isolation turns out to remain cumbersome.

To improve the management of many defects, in this paper we propose the concept of *hyper explanations*, an extension of *defect explanations* [8], that allows to derive a prioritization of defects based on their respective *severity* and helps to decide which defect is to be handled first. We argue that this concept can improve the simultaneous handling of high numbers of defects by identifying and prioritizing erroneous model parts that are potentially responsible for multiple defects. We evaluate our concept by applying it to industrial variability models and discuss the obtained results with domain experts, which brought interesting insights about the reasons behind the exposed defects.

The contributions of this paper are as follows:

- We introduce the concept of *hyper explanations* and use it to derive a prioritization of defects and defect-creating constraints.
- We evaluate this concept by applying it to industrial variability models and confirm the presence of multiple dead features. Further, we show that hyper explanations are suitable to identify feature model parts that are responsible for multiple defects.
- Based on the results, we identify the technical reasons behind dead features in industrial variability models with domain experts and discuss mechanisms of creating them.

2 BACKGROUND AND RUNNING EXAMPLE

In this section, we introduce the running example that is used to outline the background needed for understanding the concepts and mechanisms explained in the following sections. Further, we explain the structure of the used *feature models* and outline the approach of *defect analyses*. Moreover, we introduce the concept of *defect explanations*.

2.1 Feature Models

To improve the development and maintenance of variability containing software intensive systems, they can be managed by a Software Product Line (SPL) [14]. Therefore, the software systems are split into dedicated *features* that represent specific functionalities and are mapped to software artifacts that implement them. Individual features can then be selected to create a *configuration* that is used to compose the involved features into a *software variant*. Allowing arbitrary configurations potentially results in unintended or erroneous variants which is why the creation of configurations needs to be limited. To describe and model those limitations, a *feature model* [7] can be used.

Feature models (Figure 1) are based on a tree structure of features to describe their hierarchical relations, which can be refined by different modifiers. In general, there are two types of modifiers that are either valid for a single feature (*mandatory*, *optional*) or groups of features (*alternative*, *or*). While *mandatory* features need to be selected, *optional* features do not. Groups of features that are marked as *alternative* imply that exactly one of those features needs to be selected while an *or*-group requires that at least one of the involved features is selected. It needs to be considered that all of those modifiers only apply, if the parent of the respective feature is selected.

Figure 1 shows a feature model that describes the variability of a car and serves as the running example for this paper. It uses the previously outlined modifiers to express how a car can be configured. The features *Carbody* and *Gearbox* are mandatory, as they are needed in every car. The gearbox requires further configuration, as it has two child features being *Manual* or *Automatic*. Those features are modeled as alternatives, whereby only one selection is permitted. The *Radio* is modeled as an optional feature, indicating that not every car needs to have one. If selected, the *Radio* offers further configuration like a *Bluetooth* interface or *Navigation* functionality. Besides features, their relations and modifiers, feature models

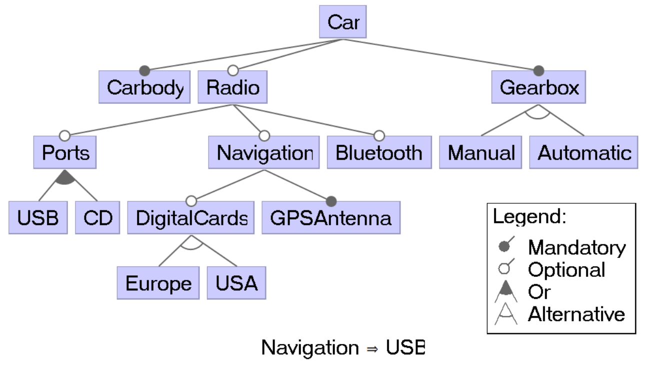

Figure 1: Example: Simplified Feature Model in the Automotive Domain

further support *cross-tree-constraints* (CTC). Those constraints are used to express feature relations that cannot be modeled by the tree structure itself. In Figure 1, the constraint $Navigation \implies USB$ forces the selection of the *USB* port if the *Navigation* feature is selected.

2.2 Feature Model Defect Analysis

A feature model describes the relations of individual features and thereby models all configurable variants of its corresponding platform. However, it can contain anomalies, called *defects*, that violate the intended variability, like for example features that can never be selected. Defects emerge through interactions between different feature model parts such as constraints, feature relations and their modifiers. To reveal contained defects, automated analyses can be applied. Therefore, a feature model can be translated into an equivalent boolean formula FM [12] that creates the basis for most of the computations performed during the analysis. Although there are multiple different types of defects, we only consider *dead features* in this paper.

Dead Features. The feature model limits the configurable variants by defining a semantic of how the individual features can be combined. However, it is possible to create feature models that contain features which can never be selected. The affected features are called *dead features* and cannot be part of any configuration and thereby are not present in any variant. In research, dead features are handled as an error, as they add complexity to the model and its maintainability without offering any benefit, similar to dead code in programming. It is commonly recommended to resolve those defects or to delete the respective features.

In the example shown in Figure 2 there are four dead features of which one is the feature *Manual*. It is marked as dead due to the constraint $Carbody \implies Automatic$. This is because *Carbody* is a mandatory feature and therefore contained in every configuration. To satisfy the constraint, *Automatic* also needs to be contained in every configuration. Because *Manual* is modeled as an alternative to *Automatic*, it can never be selected, as an alternative group only allows one selection.

Dead features, like most of defects, can be identified by utilizing SAT solvers. A SAT solver is able to determine, whether a boolean

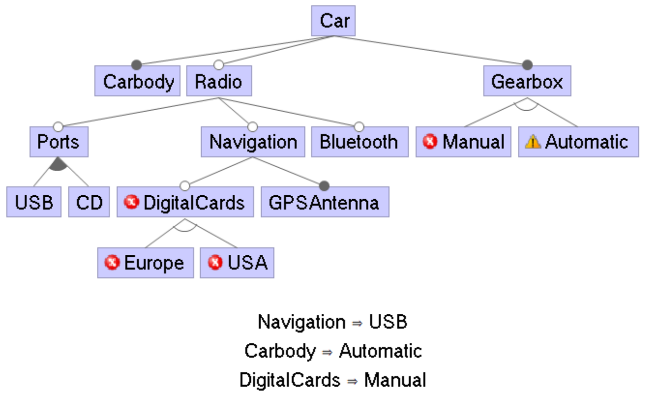

Figure 2: Example Feature Model Containing Dead Features

formula has any assignment that evaluates to *true*. To evaluate if a feature f is dead, it is determined whether the formula $d_f = FM \wedge f$ is satisfiable. If it is not, the feature f is dead.

2.3 Explanations of Defects

The cause for a feature model defect can get complex and may include many different feature model parts. To support the developer when aiming to resolve a specific defect, it is highly beneficial to be able to derive a precise explanation that identifies those parts. To derive an explanation for dead features, it is initially determined if the target feature f is actually dead by performing the satisfiability check based on the previously introduced formula $d_f = FM \wedge f$. If the feature is dead, the explanation is generated based on the *Minimal Unsatisfiable Subset* [11] of that very formula.

Definition 2.1 (Minimal Unsatisfiable Subset). Given a set $\mathbb{M}$ of constraints, a subset $\mathbb{N} \subseteq \mathbb{M}$ is a minimal unsatisfiable subset (MUS) of $\mathbb{M}$, if and only if $\mathbb{N}$ is unsatisfiable and for all $n \in \mathbb{N}$ the set $\mathbb{N} \setminus \{n\}$ is satisfiable. [3]

Deriving the MUS is suitable to identify the set of constraints which contribute to the unsatisfiabiliy. However, for a given set of constraints, many different minimal unsatisfiable subsets can be computed. To reduce the probability of missing any constraint that contributes to the unsatisfiability of the formula, a set of MUSes is derived. Next, for each constraint in d_f, it is determined how many of the derived MUSes contain that very constraint. This allows further conclusions about the likeliness of the constraint to be part of the cause for the defect. This likeliness is referred to as its *confidence* and is computed by determining the ratio between the number of MUSes that contain the constraint and the total number of derived MUSes. A constraint that is present in many MUSes is assigned with a high confidence while constraints that are only present in some MUSes are assigned with a low confidence. To encode the confidence, Kowal et al. [9] propose to color involved constraints with a gradient from red to black, where red-colored constraints show a high confidence and black-colored constraints show a low confidence. For our concept, we formalize this color gradient with a value range from 0 to 1, where the value 1 indicates the highest confidence and the value 0 indicates the lowest confidence.

Defect	Constraint	Confidence
d_M	Car	0.667
	$Car \implies Carbody$	1
	$\neg(Manual \wedge Automatic)$	1
	$Carbody \implies Automatic$	1
d_{DC}	Car	0.667
	$Car \implies Carbody$	1
	$\neg(Manual \wedge Automatic)$	1
	$Carbody \implies Automatic$	1
	$DigitalCards \implies Manual$	1
d_{EU}	Car	1
	$Car \implies Carbody$	1
	$\neg(Manual \wedge Automatic)$	1
	$Carbody \implies Automatic$	1
	$DigitalCards \implies Manual$	1
	$Europe \implies DigitalCards$	1
d_{USA}	Car	1
	$Car \implies Carbody$	1
	$\neg(Manual \wedge Automatic)$	1
	$Carbody \implies Automatic$	1
	$DigitalCards \implies Manual$	1
	$USA \implies DigitalCards$	1

Table 1: Defects and their Explanations

The final explanation consists of a the defect, i.e. the dead feature, and a set of constraints with their respective confidence. Table 1 shows the explanations of the four dead features contained in the running example.

3 DEALING WITH HIGH NUMBERS OF FEATURE MODEL DEFECTS

Although automatically generated explanations can be useful for developers when aiming to fix a specific defect, handling high numbers of defects remains challenging, as defects can be related and mutually dependent. Hence, it is difficult to decide which defect or which defect-creating constraint should be prioritized to achieve the best possible progress when aiming to resolve present defects. To mitigate this problem, a hyper explanation can be used that aggregates the information of explanations and provides the bases for deriving a prioritization.

3.1 Deriving a Hyper Explanation

Formally, a hyper explanation for a set of defects is a set of 3-tuples, one for each unique constraint that is contained in the explanation of any defect. Those 3-tuples consist of the constraint itself, a float value between 0 and 1 that indicates the constraints *severity* and a set of defects, which contains all defects in which the respective constraint is involved. Table 2 shows an example of the structure of those 3-tuples based on the constraint $Car \implies Carbody$.

Constraint	Severity	Defects
$Car \implies Carbody$	[0 - 1.0]	$d_M, d_{DC}, d_{EU}, d_{USA}$

Table 2: 3-Tuple of Constraint $Car \implies Carbody$

While the constraint and the set of defects can be directly derived from the given explanations, the severity is subject to further computation. The severity is an indicator for the relevance of the respective constraint and is used as central information when deriving the final prioritization for defect resolving, we consider different strategies for computing the severity.

Strategy 1: Absolute Values. The first strategy is based on the assumption that constraints that appear in multiple explanations should be assigned with a high severity. Therefore, this strategy computes the severity of each constraint by counting the explanations that contain this very constraint and divides the result by the total number of explanations to obtain a value between 0 and 1. Table 3 shows the resulting hyper explanation using this strategy based on the explanations shown in Table 1.

Constraint	Severity	Defects
Car	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$Car \implies Carbody$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$\neg(Manual \wedge Automatic)$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$Carbody \implies Automatic$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$DigitalCards \implies Manual$	0.75	d_{DC}, d_{EU}, d_{USA}
$Europe \implies DigitalCards$	0.25	d_{EU}
$USA \implies DigitalCards$	0.25	d_{USA}

Table 3: Hyper Explanation by Absolute Values

Strategy 2: Average Confidence. The second strategy considers further information that is provided by the given explanations. Here, the severity of each constraint is derived by computing the average confidence of the very constraint across all explanations where explanations that do not contain the target constraint are considered with a value of 0. Table 4 shows the resulting hyper explanation that is obtained when applying this strategy.

Constraint	Severity	Defects
Car	0.8335	$d_M, d_{DC}, d_{EU}, d_{USA}$
$Car \implies Carbody$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$\neg(Manual \wedge Automatic)$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$Carbody \implies Automatic$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$DigitalCards \implies Manual$	0.75	d_{DC}, d_{EU}, d_{USA}
$Europe \implies DigitalCards$	0.25	d_{EU}
$USA \implies DigitalCards$	0.25	d_{USA}

Table 4: Hyper Explanation by Average Confidence

Strategy 3: Maximum Confidence. The third strategy is based on the assumption, that only the highest confidence of a specific constraint across all explanations is crucial for the final severity value of that very constraint. Therefore, this strategy is based on determining the highest confidence for each individual constraint to use it as its severity value. The corresponding hyper explanation is shown in Table 5. Even though all three strategies are based on reasonable considerations, the results showed major differences that make them differently suitable for our goal of deriving a prioritization. While the first strategy is able to derive differentiable severity values, it lacks detail. Especially by ignoring the confidence of the processed constraints, the resulting severity values are not detailed enough to derive a reliable prioritization. This also applies to the third strategy. Here, the severity value is computed without considering the number of explanations that contain a specific constraint. In the example in Table 5, it can be seen that all constraints are assigned with the same severity value, despite major differences in terms of involved defects. Therefore, we decided to use the second strategy which computes the severity values based on the average confidence, because it provides detailed values that are computed by considering all available information that is contained in the given explanations.

Constraint	Severity	Defects
Car	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$Car \implies Carbody$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$\neg(Manual \wedge Automatic)$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$Carbody \implies Automatic$	1	$d_M, d_{DC}, d_{EU}, d_{USA}$
$DigitalCards \implies Manual$	1	d_{DC}, d_{EU}, d_{USA}
$Europe \implies DigitalCards$	1	d_{EU}
$USA \implies DigitalCards$	1	d_{USA}

Table 5: Hyper Explanation by Maximum Confidence

3.2 Determining a Prioritization of Defect-Creating Constraints

By deriving the hyper explanation, we create the basis for determining a prioritization of constraints. We aim to create a mechanism that prioritizes constraints that are involved in defects that are mutually dependent or generally related to other defects, because resolving those constraints potentially also resolves other defects completely or partially. Because the severity value already encodes this information for every constraint contained in the hyper explanation, their final prioritization can be directly derived by ordering them by their assigned severity. Table 6 shows the resulting prioritization of defect-creating constraints contained in the running example. In case of two constraints showing the exact same severity we assign them the same prioritization, as we cannot state which constraint is to be preferred.

Constraint	Severity	Prioritization
$Car \implies Carbody$	1	1
$\neg(Manual \wedge Automatic)$	1	1
$Carbody \implies Automatic$	1	1
Car	0.8335	2
$DigitalCards \implies Manual$	0.75	3
$Europe \implies DigitalCards$	0.25	4
$USA \implies DigitalCards$	0.25	4

Table 6: Final Prioritization of Defect-Creating Constraints

3.3 Determining a Prioritization of Defects

Besides the prioritization of constraints, it can further be beneficial to be able to derive a prioritization of the defects themselves, because the process of handling erroneous feature models is commonly oriented on resolving defects. While the prioritization of defect-creating constraints can be derived by ordering them by their severity value, determining the prioritization of defects requires further computation. Here, we argue that defects whose explanations contain high numbers of highly severe constraints should be prioritized. Further, we determine the severity of defects by computing the average severity of the constraints involved in the respective defect. Based on the determined severity value, we are able to derive the final prioritization.

Defect	Severity	Prioritization
d_M	0.96	1
d_{DC}	0.92	2
d_{EU}	0.81	3
d_{USA}	0.81	3

Table 7: Prioritization of Defects Based by Severity

4 EVALUATION

In this paper, we introduce the concept of hyper explanations to derive a prioritization of defects and defect-creating constraints. To evaluate this concept and its impact on the process of handling multiple defects, we defined two research questions that are answered to determine the benefit of hyper explanations.

RQ1: *How many defects do industrial feature models contain?*
With this research question, we aim to evaluate the presence and the number of defects contained in industrial feature models to capture the significance and relevance of those anomalies in industry.

RQ2: *Are hyper explanations suitable to derive a prioritization that benefits the process of handling multiple feature model defects?*
With this research question, we aim to determine the impact of hyper explanations on the process of identifying and handling multiple defects at once. Here, it is of special interest whether the derived prioritization is suitable to identify defects or constraints that resolve multiple other defects as well.

4.1 Experimental Setup

To perform the evaluation, we apply the concept of hyper explanations to industrial variability models. The target models consist of selectable configuration options along with constraints that define how those options are to be combined. To apply our concept, we convert the target variability models into standardized feature models by creating a feature for each configuration option and adapting the constraints accordingly. During the conversion, we had to consider the entry and cancellation dates which are assigned to all configuration options and constraints and define an interval where the respective model part is valid. Due to this temporal aspect, converting a single variability model results in a set of feature models, one for each interval. For our case study, we chose to select the feature model that contains the highest number of dead features to gain the best possible insight into the results obtained by hyper explanations. This feature model contains a total of **549** features of which **94** are marked as *dead*. The model contains **1109** constraints. The generated hyper explanation that is based on the explanations of the 94 dead features contains **178** constraints.

4.2 Results

Based on the hyper explanation that was derived for the 94 dead features contained in the selected feature model, we determine the prioritization of defect-creating constraint and the defects themselves and analyze its benefit for the defect resolving process.

Prioritizing Defect-Creating Constraints. The severity values of constraints contained in the derived hyper explanation show a wide spread reaching from **0.002** up to a maximum of **0.95**. The distribution of the severity values is shown in the chart in Figure 3. This chart shows the severity on the X-axes and the respective defect-creating constraints on the Y-axes. For readability reasons, we left out the data of 150 low severe constraints.

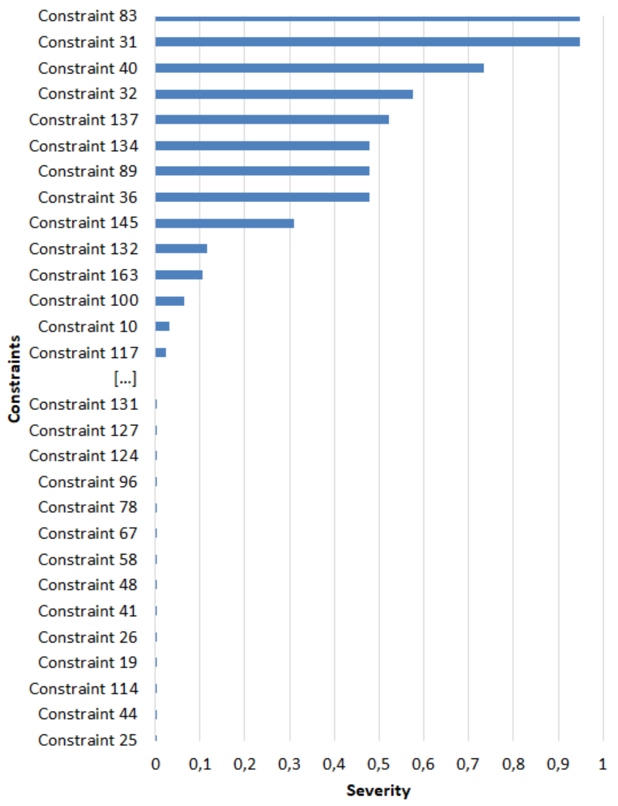

Figure 3: Defect-Creating Constraints and their Severity

To evaluate the usefulness of hyper explanations and the resulting prioritization of constraints on the process of addressing multiple defects, we analyze the effect when resolving the most severe constraint. As mentioned earlier, the analyzed feature model originally contains 94 dead features. Fixing only the most severe constraint (*Constraint 83*) already reduces this number by 69 to a total of 25 dead features. Continuing with fixing the second most severe constraint further reduces this number to a total of 16. Overall, fixing the three most severe constraints reduces the total number of dead features from 94 to 12. To ensure that this effect is achieved through the derived prioritization, we checked the effect when resolving the three least severe constraints, which did not reduce the amount of dead features at all.

Prioritizing Defects. After evaluating the prioritization of defect-creating constraints, we aim to consider the benefit of the prioritization on defect level. Therefore, we computed the severity of all dead features contained in the target feature model and derived the corresponding prioritization. The obtained results are visualized in the chart shown in Figure 4. To enhance the readability, we left out the data of 65 defects that showed a medium severity. It can be seen that even though the computed severity values show a wide spread from 0.02 to a maximum of 0.74, they cannot be categorized into different groups in contrast to the previously computed severity values of defect-creating constraints.
To evaluate the benefit of the resulting prioritization, we consider the effect when fixing high prioritized defects and compare it to the effect when fixing defects with a low prioritization. This evaluation brought similar results as on constraint level. Fixing only the defect with the highest prioritization (*Defect 47*) reduced the number of dead features by 69 to a total of 25. Fixing the next two highest prioritized defects further reduced this number to a total of 12 dead features. On the contrary, we evaluated the effect when resolving the three least prioritized defects. By doing so, the number of dead features was only reduced by exactly those three defects.

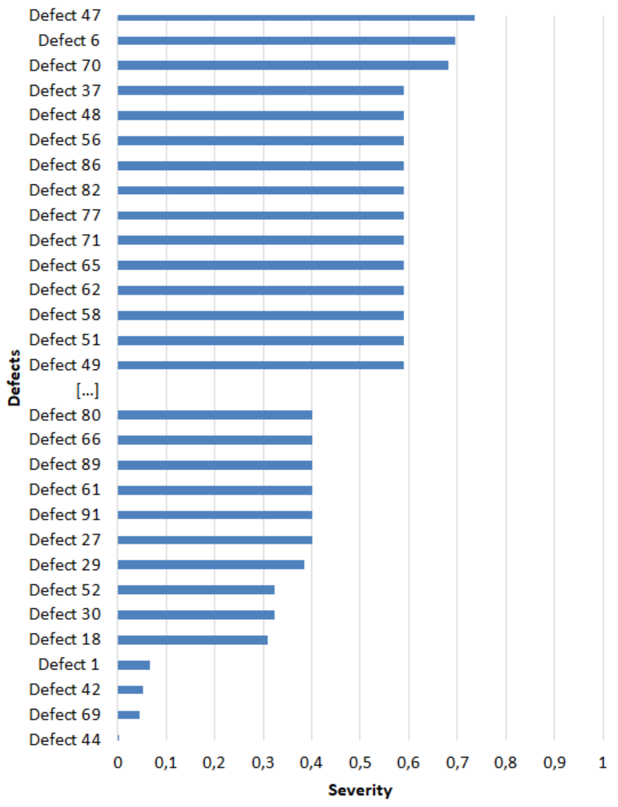

Figure 4: Defects and their Severity

4.3 Discussion

The severity values of defect-creating constraints in hyper explanations provide interesting insights. It can be seen that a few constraints stand out by showing a significantly higher severity than others. Moreover, the set of severity values can be subdivided into two groups: High severity and low severity as there are hardly any values in between. We used the derived prioritization of constraints to evaluate its benefit when handling high numbers of defects. Here, we were able to show that the prioritization is highly beneficial, as removing only the three constraints with the highest priority already reduced the amount of dead features by 74 %. This is surprising, as those three constraints only represent ca. 1.6 % of all defect-creating constraints. Moreover, we evaluated the benefit of the prioritization of defects and were able to confirm a similar, positive effect. Further, we were able to confirm the described effects for all variability models that we considered during our evaluation. Based on those results, we answer the previously defined research questions:

RQ1: *How many defects are contained in industrial feature models?*
We converted industrial variability models into standardized feature models to apply automated analyses and revealed that all of the resulting feature models contain multiple defects. In the worst case, the resulting feature model contained a total of 549 features of which 94 were marked as dead. This equals 17 % of all contained features, which is a surprisingly high number when considering that it is commonly recommended that those features should be deleted. While this feature model represents the peak in terms of present defects, the other feature models showed similar results.

RQ2: *Are hyper explanations suitable to derive a prioritization that benefits the process of handling multiple feature model defects?*
Based on the selected case study, we were able to show that the prioritization derived by hyper explanations is suitable to significantly improve the process of managing high numbers of feature model defects. We were able to show that modifying only a few, highly prioritized constraints or defects already reduce the number of defects to a large extent, which confirms our assumption that hyper explanations are suitable to identify constraints that show a high relation to other defects.

4.4 Threats to Validity

Although our evaluation showed the feasibility of hyper explanations and brought promising results, this concept is subject to different threats that impair its validity.
First, the evaluation was performed using exemplary converted industrial variability models. Even though the used models are large and contain high numbers of defects and constraints, they are not suitable to show a general applicability and correctness of our concept of hyper explanations and the resulting prioritization. To

mitigate this, we considered multiple variability models to increase the reliability of the obtained results.
Moreover, the severity values of defects and defect-creating constraints, which are used to determine their final prioritization, are computed based on a manually designed metric. To limit this threat, we considered different strategies to compute the severity and analyzed their results. However, a general validity of the resulting severity values and the respective prioritization cannot be stated for arbitrary feature models. Furthermore, hyper explanations and the according prioritization are based on generated explanations and their corresponding confidence values. As those values cannot be computed with an absolute precision due to the outlined characteristics of minimal unsatisfiable subsets, the lack of precision is transferred to our computations and the resulting prioritization.

5 DEAD FEATURES IN INDUSTRIAL FEATURE MODELS

After translating the industrial variability models into standardized feature models, we are able to apply automated analyses which expose high numbers of dead features. To understand the reasons for those defects, we conducted interviews with different domain experts who are responsible for modeling and maintaining the considered variability models. In this section, we summarize the interviews and discuss potential consequences.

5.1 Intentional Use of Dead Features

Due to the high number of dead features that were revealed in the converted variability model, we initially expected either an error in our translation or actual defects that were modeled by accident. However, the interviews brought two surprising insights. (1) The presence of dead features is known and (2) most of the dead features are not actual defects but intentionally created. According to the domain experts, those dead features are used to model temporarily disabled configuration options which are needed for various reasons, such as:

Configuration Option is Subject to Supply Shortages. If a specific configuration option cannot be selected due to supply shortages, it needs to be disabled to prevent customers from selecting it until it is available again.

Configuration Option is Planned for Later Release. If a specific configuration option is planned for later use, it needs to be added to the variability model before its actual release to be available in other pre-sales systems for, e.g., logistical applications. However, it needs to be ensured that customers are not able to select this particular option before its planned entry date.

Configuration Option has Issues. If a specific configuration option is involved in any issue, e.g. technical or logistical, it is disabled to avoid its selection until the issue is fixed.

Configuration Option is Potentially Hazardous. If a specific configuration option is potentially hazardous due to external influences, it needs to be disabled. Although this is a rather exceptional case, it was applied a few years ago during the nuclear accident in Fukushima in 2011, where the components of suppliers located near the power plant were contaminated by radiation.

5.2 Mechanisms for Modeling Dead Features

Based on the fact that dead features are intentionally created to model disabled configuration options, the experts further pointed out multiple different mechanisms to create dead features in the corresponding variability model. We were able to confirm those mechanisms in the data of configuration options and constraints of the previously converted variability model and classified those mechanisms into two main categories. In the following, d is used to denote the feature that is to be modeled as dead.

Constraints. The first mechanism uses constraints that explicitly prevent the selection of the target feature. This can either be done by directly preventing the selection of the target feature with the constraint $\neg d$, or by setting up a constraint that forces an invalid configuration if d is selected, like the deselection of a core feature c as follows: $d \implies \neg c$.

Temporal Validity. The entry and cancellation date of a specific configuration option can be used to prevent its selection. Therefore, the entry date can be set to the distant future making the feature invalid for the current time. A similar approach is to choose a cancellation date terminated before the entry date to prevent the validity of the option for any point in time.

Although disabling features by making them dead is an intentional and frequently used approach, none of the outlined mechanisms is a defined standard according to the experts. Due to the lack of a standardized mechanism for modeling disabled features, the defect analysis of variability models is further complicated, because disabled features can't be distinguished from actual dead features.

5.3 Discussion

The interviews with domain experts have shown that the revealed dead features are not actual defects, but intentionally modeled. The experts outlined different reasons for the need of modeling dead features, which all require the target feature to be disabled *temporarily*. Moreover, the experts explicitly stressed that modeling a disabled feature by temporarily deleting it is impractical as the affected data is processed by other IT-systems, like logistics or marketing, where the deletion would most likely result in a high amount of issues and side-effects. In research, a dead feature is treated as an error, implicitly stating that it needs to be resolved or deleted. However, our analysis and evaluation revealed that dead features are actively used in the industry to prevent their selection. Based on those insights, we state that there is a need for a defined mechanism to distinguish them from accidentally created dead features and actual defects. A possible solution for this requirement would be to use temporal feature models [13] to model the temporal validity of the target feature. However, dealing with the overhead that comes with this concept makes it unsuitable for this use case, which is why we propose to extend standard feature models with the option to mark features as *disabled*. Thereby, we not only create an explicit distinction in the terminology but are also able to define an individual handling of those features in terms of a visual highlighting in the feature model itself or a specific consideration during different automated analyses.

6 RELATED WORK

The field of automated feature model analysis has found a lot of attention in research. For our work, especially the automated detection of feature model defects is relevant, because it creates the basis for our concept of hyper explanations. Benavides et. al [2] outlined different automated analyses that create the foundation of hyper explanations, because they allow to reveal feature model defects such as dead features.

A further foundation of our concept of hyper explanations is the work of Kowal et. al [8], who introduced defect explanations to identify feature model parts whose interaction cause a specific defect. While the identification of model parts that are involved in a specific defect only supports the treatment of this very defect in isolation, hyper explanations aggregate the information about erroneous feature model parts and their corresponding confidence of all explanations. This allows to reason about the severity of defect and defect-creating constraints. In general, we argue that both concepts can be used complementary to enhance the process of resolving defects in feature models.

Another application of explanations used in the context of feature models is introduced in [10]. Here, explanations are not used to identify the cause of a defect, but to create a plain text explanation of the behavior and the variability of dynamic feature models. To derive the explanation, Kramer et. al assigned text attributes to features and their relations (mandatory, alternative, etc.) that represent text fragments which are composed to a complete explanation based on their path in the feature model tree.

Moreover, the defect analysis of large and proprietary variability models has also been applied to the Linux kernel. This variability model shows a similar complexity to our case study. Tartler et al. [15] designed a tool that is able to perform different integrity analyses between the kernels configuration files and the mapped preprocessor directives in the source code that are used to compose the configured kernel. Thereby, Tartler et al. were able to expose inconsistencies that result in dead code. In detail, over 300 of those anomalies were extracted.

7 CONCLUSION

Industrial variability models are subject to multiple reasons for defects. However, automated analyses of variability models are still not common in industry, which is why contained defects remain undetected. To obtain detailed information about contained defects, we converted industrial variability models into standardized feature models to apply automated analyses. We were able to show that those variability models indeed contain high numbers of defects. However, analyzing and addressing each defect in isolation turned out to be cumbersome, since the overall complexity of the corresponding feature models and individual defect explanations complicates this highly manual process.

To improve the management of multiple defects at once, we introduced the concept of hyper explanations that is suitable to automatically derive a prioritization of defects and defect-creating constraints. We evaluated our concept by applying it to the previously translated industrial variability model and showed that hyper explanations are highly beneficial when dealing with many defects. In particular, the computation of severity values for defects and defect-creating constraints is a major improvement for the defect resolution process, as the resulting prioritization allows to identify defects that, when resolved, also resolve other defects.

In addition, applying hyper explanations to real world industrial variability models brought further insights regarding dead features. In contrary to the common recommendation in research to resolve or delete dead features, we were able to show that dead features are used in practice to temporarily *disable* individual features. Based on this knowledge, we propose to extend feature models by a mechanism that allows to disable features. This mechanism helps to distinguish disabled features and dead features. The concept of disabled features is subject to future work, where especially the implementation of this mechanism is a central aspect. We propose to introduce a flag, similar to abstract or concrete features [17], that explicitly marks a feature as disabled. Moreover, it needs to be analyzed how those features can be handled semantically, especially in the context of automated analyses. Here, transitive dependencies to other features may be subject to further research.

8 DISCLAIMER

The results, opinions and conclusions expressed in this thesis are not necessarily those of Volkswagen Aktiengesellschaft.

REFERENCES

[1] David Benavides, Antonio Ruiz-Cortés, Pablo Trinidad, and Sergio Segura. 2006. A Survey on the Automated Analyses of Feture Models. *Jornadas de Ingeniera del Software y Bases de Datos* (2006), 367–376.
[2] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708.
[3] Jaroslav Bendík and Ivana Černá. 2020. MUST: Minimal unsatisfiable subsets enumeration tool. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 135–152.
[4] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. 7:1–7:8.
[5] Megha Bhushan, Arun Negi, Piyush Samant, Shivani Goel, and Ajay Kumar. 2020. A classification and systematic review of product line feature model defects. *Software Quality Journal* (2020), 1–44.
[6] José A. Galindo, David Benavides, Pablo Trinidad, Antonio-Manuel Gutiérrez-Fernández, and Antonio Ruiz-Cortés. 2019. Automated Analysis of Feature Models: Quo Vadis? *Computing* 101, 5 (2019), 387–433.
[7] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute.
[8] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. 2016. Explaining Anomalies in Feature Models. 132–143.
[9] Matthias Kowal, Sofia Ananieva, Thomas Thüm, and Ina Schaefer. 2017. Supporting the Development of Interdisciplinary Product Lines in the Manufacturing Domain. 50, 1 (2017), 4336–4341.
[10] Dean Kramer, Christian Sauer, and Thomas Roth-Berghofer. 2013. Towards explanation generation using feature models in software product lines. *Knowledge Engineering and Software Engineering (KESE)* (2013), 13.
[11] Mark H. Liffiton and Karem A. Sakallah. 2008. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. 40, 1 (2008), 1–33.
[12] Mike Mannion. 2002. Using First-Order Logic for Product Line Model Validation. 176–187.
[13] Michael Nieke, Christoph Seidl, and Thomas Thüm. 2018. Back to the Future: Avoiding Paradoxes in Feature-Model Evolution. 48–51.
[14] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.*
[15] Reinhard Tartler, Julio Sincero, Wolfgang Schröder-Preikschat, and Daniel Lohmann. 2009. Dead or Alive: Finding Zombie Features in the Linux Kernel. 81–86.
[16] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. 47, 1 (2014), 6:1–6:45.
[17] Thomas Thüm, Christian Kästner, Sebastian Erdweg, and Norbert Siegmund. 2011. Abstract Features in Feature Modeling. 191–200.