Risk-Based Integration Testing of Software Product Lines

Remo Lachmann[1], Simon Beddig[1], Sascha Lity[1], Sandro Schulze[2], Ina Schaefer[1]

[1]Technische Universität Braunschweig
{r.lachmann, simon.beddig, s.lity, i.schaefer}@tu-bs.de

[2]Otto-von-Guericke Universität Magdeburg
sanschul@iti.cs.uni-magdeburg.de

ABSTRACT

Software product lines (SPL) capture commonalities and variabilities of product families and, thus, enable mass customization of product variants according to customers desired configurations. However, they introduce new challenges to software testing due to a potentially large number of variants. While each variant should be tested, testing resources are limited and, thus, a retest of all, partially redundant, test cases for each variant is not feasible in SPL testing. Coping with these issues has been a major research focus in recent years, leading to different testing approaches. However, risk-based testing has not gained much attention in the SPL domain while being a successful approach for single software systems. In this paper, we propose a novel risk-based testing approach for SPL integration testing. We incrementally test SPLs by stepping from one variant to the next. For each variant, we automatically compute failure probabilities and failure impacts for its architectural components. To avoid a computational overhead of generating and analyzing each variant, we exploit the variability between variants defined as deltas to focus on important changes. We evaluate our approach using an automotive case study, showing that the risk-based technique leads to positive results compared to random and delta-oriented testing.

CCS Concepts

•Software and its engineering → Software product lines; Software testing and debugging;

Keywords

Software Product Lines, Risk-based Testing, Test Case Prioritization, Model-based Testing

1. INTRODUCTION

Software-intensive systems gain more and more importance due to the fact that almost all systems of our daily life are influenced by software. While a high quality is an important aspect of every software-intense system, it becomes even harder to achieve if variability-rich systems, e.g., *software product lines (SPLs)* [21], are developed. SPLs introduce variability, but also commonalities between a potential large set of variants. Testing is challenging for SPLs, as the number of product variants might be very large and commonalities between variants lead to a high degree of redundancy. Regression testing techniques [26], e.g., test case selection or prioritization, are sufficient to cope with potential redundancy and complexity in testing. Different SPL testing techniques have been introduced in the past [6]. In previous work, we introduced a test case prioritization technique for SPLs based on delta-oriented architecture models [17]. This technique focuses on changed parts between product variants by analyzing *deltas* between them. A delta describes modifications between variants, in this case between architecture models used for integration testing. Based on this delta information, we developed a test case prioritization technique, which ranks test cases according to coverage of architecture changes. We extended this approach using behavioral knowledge within components to a achieve a more fine-grained prioritization [16].

However, especially in safety-critical systems it is of importance to focus on system parts with a high *risk potential* to compensate limited resources [15]. Risk-based testing has been widely adopted for single-software systems to focus on test cases related to entities that exhibit the greatest risk to the system [1, 10]. In context of this paper, risk is defined in two dimensions: *failure impact*, i.e., the damage that a failure of a particular system part causes and the *failure probability*, i.e., the chance that a failure occurs [1]. However, despite the benefits of risk-based testing, only the work by Hartmann et al. [11] introduces a first idea of risk-based testing for SPLs. They apply risk-based testing in the domain engineering [21] phase, i.e., they annotate features with failure impact and probability values. In comparison to their basic approach, we introduce a risk-based testing concept which computes failure probabilities automatically and derives more complex impact values based on changes between product variants for each component in the variant. This allows for a risk-based approach that is far more efficient than manual risk assessment approaches as existing techniques do not scale with a growing number of product variants. Thus, our approach can be applied to a large number of variants. Similar to previous work [17, 16], we apply incremental testing based on delta knowledge and prioritize test cases based on their coverage of highly risky system parts. We evaluate

DOI:

our technique showing that our risk-based approach is more effective than random testing and our previous technique as it finds important failures earlier. In addition, it is far more efficient than manual risk-based testing.

In summary, we make the following contributions:

1. We introduce a novel semi-automatic risk-based testing approach based on delta-oriented architecture models, enabling product-wise risk computation.
2. We describe how to derive *failure impact values* for components automatically based on manually assigned feature impact values in the design phase.
3. We introduce a new technique to automatically derive *component failure probabilities* in an incremental fashion, based on previously tested product variants and delta knowledge. This reduces the testing effort significantly compared to traditional, manual approaches.
4. We evaluate our technique, leading to promising failure finding rates compared to other techniques.

This paper is structured as follows: We describe the necessary background in Sec. 2. Our novel risk-based test case prioritization is explained in Sec. 3. In Sec. 4, the results of our evaluation are presented. Related work is discussed in Sec. 5. Sec. 6 concludes the paper and shows future work.

2. BACKGROUND

To provide the necessary foundation for our concept, we will explain software product lines, delta modeling and delta-oriented test case prioritization in this section.

Software Product Lines. In many domains, the demand for customized products is increasing [21]. Based on a set of *features* $\mathcal{F} = \{f_1, \ldots, f_m\}$, i.e., customer-visible functionality, customers are capable to configure *products* $P = \{p_1, \cdots, p_n\}$ of an SPL according to their preferences. The corresponding configuration options are defined by a *feature model fm*. In feature models [14], features are hierarchically structured to capture their relationship, where features are further typed as *mandatory*, *optional*, *alternative*, and *or*. In addition, feature dependencies can be specified by *require*, i.e., a feature requires the selection of other features, and *exclude* relations, i.e., a feature excludes the selection of other features. Hence, for each product $p_i \in P$ a certain feature configuration $F_{p_i} \subseteq \mathcal{F}$ exists comprising all features of the current product and also satisfying the feature model such that $F_{p_i} \models fm$ holds.

EXAMPLE 1. *An example for a feature model* fm *is shown in Fig. 1. Here,* fm *contains five different features, i.e., the mandatory features* root, X *and* Y *and the optional features* Z *and* W. *Using* fm, *we could derive the minimal configuration with* root, X *and* Y.

Delta Modeling. Besides feature modeling [14] used for the specification of the set of configurable products P, several variability modeling techniques [24] exist for the development of reusable software artifacts. In this paper, we focus on delta modeling due to previous work on incremental SPL integration testing [19, 17]. *Delta modeling* [4] is a modular and flexible transformational modeling formalism applicable to various types of software artifacts such as source code [23] or finite state machines [19]. Based on a designated *core model* corresponding to a *core product* $p_{core} \in P$, a set of deltas $\Delta_{SPL} = \{\delta_1, \cdots, \delta_q\}$ is specified to transform the core into another product $p_i \in P$. A *delta* $\delta_j = (Op_j, \varphi_j)$ is defined by a finite set of *change operations* $Op_j = \{op_1, \ldots, op_r\} \subseteq \mathcal{OP}$ and an *application condition* $\varphi_j \in \mathbb{B}(\mathcal{F}_{SPL})$. A change operation represents an addition/removal of a model element or model relation. By $\mathcal{OP}$, we refer to the set of all change operations derivable for the current SPL under consideration. The application condition φ is a Boolean expression over features to specify when a delta has to be applied on the core, where $\mathbb{B}(\mathcal{F}_{SPL})$ denotes the set of all valid application conditions. For a given feature configuration F_{p_i} of a product p_i, all delta application conditions are evaluated. If an evaluation $\varphi_j \models F_{p_i}$ results in `true`, the corresponding delta δ_j is captured in the variant-specific delta set $\Delta_{p_i} \subseteq \Delta_{SPL}$ to transform the core into the respective model of p_i by applying each delta and, therefore, each captured change operation subsequently. By Δ_{SPL}, we refer to the set of valid deltas for the current SPL.

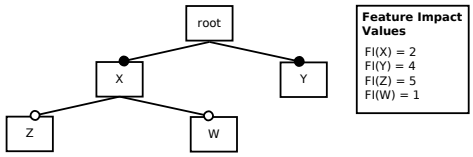

Figure 1: Sample Feature Model

In addition to the specification of the commonality and differences between the core p_{core} and any other product $p_i \in P$ by means of deltas, we are able to derive *regression deltas* Δ_{p_i,p_l} encapsulating the differences between arbitrary products p_i and p_l. Thus, a regression delta defines the change operations to transform the current product p_i into another one p_l. For the construction of regression deltas, we refer the reader to previous work [19].

In this paper, we focus on delta-oriented architectural models as already defined in prior work [19]. An *architectural model* $arc_{p_i} = (C_{p_i}, CON_{p_i}, \Pi_{p_i})$ for a product p_i describes the overall structure of a system and consists of a finite set of *components* $C_{p_i} \subseteq \mathcal{C}_{SPL}$ that represent atomic functional units, a finite set of *connectors* $CON_{p_i} \subseteq \mathcal{CON}_{SPL}$. Each connector connects two components each. Overall, the connectors transfer a finite set of *signals* $\Pi_{p_i} \subseteq \Pi_{SPL}$ between components to enable their communication. By $\mathcal{C}_{SPL}$, $\mathcal{CON}_{SPL}$, and Π_{SPL}, we refer to the set of all components, the set of all connectors, and the set of all signals of the SPL, respectively. A connector is always connected to an *output port* of a component, which sends a signal, and an *input port* of a receiving component. Thus, the sets of input and output ports define the *interface* of the component. For delta-oriented architectures, deltas are able to add/remove components, connectors and signals to a core architecture.

EXAMPLE 2. *Based on the core architecture* arc_{core} *shown in Fig. 2, there are two deltas* δ_{p_1} *and* δ_{p_2} *defined to create the architecture for variant* arc_{p_1} *and* arc_{p_2}. *Each delta contains an application condition* φ_{p_1} *and* φ_{p_2} *and a finite set of change operations, e.g.,* δ_{p_1} *adds a component* D *and two connectors* d *and* e *and removes connector* c. *The regression delta* δ_{p_1,p_2} *describes the differences between* p_1 *and* p_2.

Delta-Oriented Test Case Prioritization. In prior work, we applied delta-oriented architecture models for incremen-

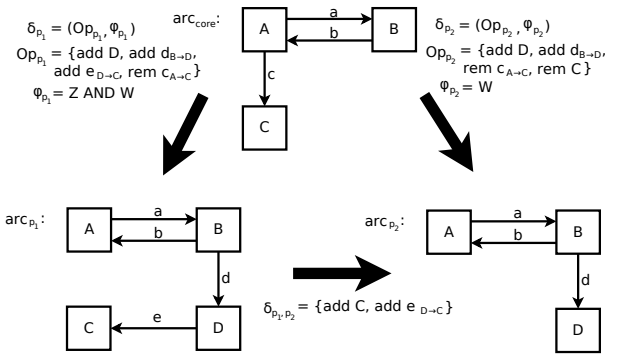

Figure 2: Example for Delta-Oriented Architecture Modeling

tal SPL integration testing [19], where for a *product variant under test* (PUT), we further prioritize test cases $\mathcal{TC} = \{tc_1, \ldots, tc_n\}$, i.e., message sequence charts (MSC) specifying interaction scenarios between components, based on delta application information [17]. Thus, we focused on the coverage of the changes in the next variant as early as possible as changes may introduce new faults to the system [5].

By stepping from a product p_i to its subsequent product p_{i+1} under test, we apply the following steps [19, 17]:

1. Compute the regression delta $\Delta_{p_i,p_{i+1}}$
2. Apply $\Delta_{p_i,p_{i+1}}$ on p_i to adapt its architecture model and its corresponding test artifacts, i.e., the set of test cases is adapted such that only new and reusable test cases are contained
3. Compute component weights
   (a) Determine first executed deltas
   (b) Determine multi product deltas
4. Prioritize test cases based on weights

In order to compute the component weights for each $c \in C_{p_{i+1}}$ in Step 3, we have to determine the *first executed deltas*. Therefore, we compute the regression deltas between the current product p_{i+1} under test and all already tested products $p' \in P_{tested} \subseteq P$. Based on those regression deltas, we are able to compare the applied change operations and, thus, are able to identify the deltas which are applied for the first time. This way, we expose architectural elements that have not been covered before via testing, as they occur for the first time in the current PUT. If the comparison results in the empty set, no new delta change operations are applied. Hence, all architectural elements have already been tested up to now in the incremental testing process.

There is a special case regarding the computation of first applied deltas. Although a component was already tested in prior products under test and there is no new behavior on first sight, i.e., the set of new deltas is empty, the component may have never appeared in its current interface configuration. In other word, the current delta change operations for a component have never been applied before in this particular combination. Hence, the product might contain new behavior never been covered by test cases before. We refer to these types of deltas as *multi product deltas* (MPD) as they only are detected when investigating more than one product under test at once.

Based on the first applied as well as multi product deltas, we are able to compute the *component weight* $w(c) : \mathcal{C} \to \mathbb{R}$ for each component $c \in C_{p_{i+1}}$ of the current product p_{i+1} under test. The weight represents the number of changes the component has undergone compared to all previously tested variants P_{tested}. Therefore, we consider for a component c the changes of incoming connectors IC compared to all incoming connectors I, the changed outgoing connectors OC in proportion to all outgoing connectors O and the MPDs in ratio to all product variants P_c in which the component c occurs as follows [17]:

$$w(c) = \alpha \cdot \frac{|IC(c)|}{|I(c)|} + \beta \cdot \frac{|OC(c)|}{|O(c)|} + \gamma \cdot \frac{|MPD(c)|}{|P_c|}$$

We use α, β, and γ as factors to individually adjust the different weight influences in the formula, where further $\alpha + \beta + \gamma = 1$ holds. In the end, the changed-based component weights are used for prioritizing the reusable test cases.

In Step 4, we incorporated the component weights into a prioritization function $prio_{sig} : \mathcal{TC} \times P \to \mathbb{R}$ [17]. The *signal-based prioritization* also takes the signals transmitted between components in a test case tc into consideration. Thus, for each signal the sending and receiving component are identified and their weights are used to compute the prioritization value of the test case as follows:

$$prio_{sig}(tc, p) = \frac{\sum_{j=1}^{n} \sum_{k=1}^{n} s(c_j, c_k) \cdot (w(c_j) + w(c_k))}{\sum_{j=1}^{n} \sum_{k=1}^{n} s(c_j, c_k)},$$

where n is the number of signals tc comprises and $s(c_j, c_k)$ returns the set of signals transmitted between c_j and c_k.

Risk-based Testing. As full testing often exceeds the available testing resources, techniques to reduce the testing effort are required. One popular approach to prioritize parts of the system is *risk-based testing* [1, 8]. *Risk* describes the chance of damage or loss of some system entity, based on two parameters. First, a *failure impact* has to be defined for a set of artifacts, e.g., requirements or functions. It represents the entities importance to the stakeholder or the system. Second, a *failure probability* is assigned for each artifact, i.e., the likelihood of malfunction. The resulting risk values indicate the importance of certain entities of the system. The higher the likelihood of a failure and its impact, the higher is the importance of the specified element. Based on this information, test cases are selected which cover the elements of highest risk to the system or project.

The impact and failure probability values are usually defined by experts based on their opinion, insight knowledge and different metrics [9]. This is a very tiresome process as each functional block or requirement has to be considered individually for single-software systems. This makes the risk-assessment of SPLs a very difficult problem due to the potential very large quantity of complex variants.

3. RISK-BASED SPL TESTING CONCEPT

In this paper, we propose a novel risk-based test case prioritization based on delta-oriented architectures for SPLs. The risk-based approach introduces a fully automatic computation of failure probabilities and a semi-automatic computation of failure impact values, which are used to prioritize test cases. Fig. 3 shows the four phases of our approach.

The first step is the manual assignment of *feature impact values*. Based on these values and the delta-oriented architectures, we compute failure probability and failure impact

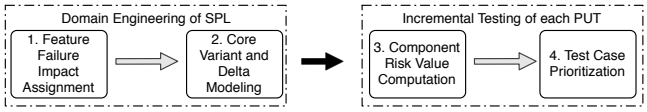

Figure 3: Overview of our Risk-based Approach

values for *each component* in the current PUT. Second, the risk-values are used to compute the priority of test cases.

3.1 Feature Failure Impact Assignment

As we perform risk-based integration testing for SPLs, we have to define both, the failure impact and failure probability for components. To this end, we require a manual assignment of failure impact values for features (FI) in the *Domain Engineering* [21] of the SPL. According to Hartmann et al. [11], feature models can be associated with impact values to introduce risk-based testing into SPL development. To this end, we use a form of *quantified feature models* (QFM) [3], to assign impact values to all features of the SPL. These FI values $FI : \mathcal{F} \rightarrow [1, 5]$ are defined on a fixed scale for each feature $f_i \in \mathcal{F}$. A value of 1 is the lowest impact value and 5 indicates a very important feature with high impact to the system. We require that each feature is assigned with an impact value manually by an expert, which is similar to currently applied risk-based approaches [11, 9]. We define the sum of feature values $FI_{\mathcal{F}}$ for a set of features $F \subseteq \mathcal{F}$ as function $FI_{\mathcal{F}} : \mathcal{P}(\mathcal{F}) \rightarrow \mathbb{N}$ such that $FI_{\mathcal{F}}(f) = \sum_{f_i \in F} FI(f_i)$ holds.

The result of this step is a QFM, which contains the SPL's features $\mathcal{F}$ in addition with impact values $FI(f_i)$ (cf. Fig. 1).

3.2 Core Variant and Delta Modeling

As delta-oriented testing is based on a *core variant* [4], we define a core architecture for integration testing. We assume that the core is tested extensively, as all parts of this product variant have not been tested yet [19]. Hence, all test cases $TC_{p_core} \subseteq \mathcal{TC}$ are executed for product variant p_{core}.

For our novel risk-based testing approach it is of importance to measure both, the failure impact and probability of each component in the system to compute the risk value of each component and to assess the priority of corresponding test cases. Thus, we require a manual mapping for each component $c_i \in C_{p_{core}}$ and each connector $con_j \in CON_{p_{core}}$ in the core variant to one (or more) corresponding features $f_i \in \mathcal{F}$ of the feature model to derive the feature impact values for later use in the risk computation. The mapping can be defined in the *Domain Engineering* phase of the SPL development [11] and is formally defined as function $F : (\mathcal{C}_{SPL} \cup \mathcal{CON}_{SPL}) \times P \rightarrow \mathcal{P}(\mathcal{F})$. Even though we do not prioritize test cases for the core, the mapping is important for testing of the following PUTs.

Example 3. *The architecture arc_{core} for product variant p_{core} is shown in Fig. 2. It is the minimal product configuration (cf. Fig. 1), comprising the mandatory features* root, X *and* Y. *In this example, the mappings of components and connectors shall be as follows:*

$$F(A) = F(C) = F(c) = X$$
$$F(B) = F(a) = F(b) = Y$$

Once the core has been defined, the next step is to define a set of deltas Δ_{SPL}, which are used to model a SPL in a delta-oriented fashion (cf. Ex. 2). Hence, the set of deltas has to be defined by an expert before actual product variants are generated and tested, i.e., we require a delta-oriented SPL. It is important that the application conditions φ_j for each delta δ_j are specified as boolean formula over the features $\mathcal{F}$ of the SPL. We will use the features and deltas to create an automated mapping from components and connectors influenced by the delta to the corresponding features. To create the architecture models for each variant p_j to be tested, the set of applicable deltas Δ_{p_j} is selected and applied to the core. Before testing of the PUT can commence, we need the mapping from features to components and connectors. Thus, we first define a function $operations : (\mathcal{C}_{SPL} \cup \mathcal{CON}_{SPL}) \times P \rightarrow \mathcal{OP}$ to determine the operations in $op_i \in \delta_i \in \Delta_{p_j}$ that affect a connector or component.

Next, we define the function $conditions : \mathbb{B}(\mathcal{F}) \rightarrow \mathcal{P}(\mathcal{F})$ which retrieves the set of features contained in δ_i's application condition φ_i. It only retrieves features that are present in the current variant (i.e., $f \in F_{p_j}$). Consequently, we are able to compute which elements are affected by an operation and the set of features which are linked to the operation. Now, we can define the *mapping function* $fmapping : (\mathcal{C}_{SPL} \cup \mathcal{CON}_{SPL}) \times \Delta_{SPL} \rightarrow \mathcal{P}(\mathcal{F})$ that returns the mapped features for an element $e \in \{\mathcal{C}_{SPL} \cup \mathcal{CON}_{SPL}\}$ as:

$$fmapping(e_k, \Delta_{p_j}) = \bigcup_{i=1}^{|\Delta_{p_j}|} conditions(\varphi_i)$$

where $\varphi_i \in \delta_i | operations(e_k, p_j) \neq \emptyset$ holds. Feature mappings can change for different product variants, depending on the deltas applied to the current PUT.

Example 4. *Applying the* mapping function *to Δ_{p_2} on the data from Ex. 2 and 3, we get the following mappings:*

$$F(A) = X$$
$$F(B) = F(a) = F(b) = Y$$
$$F(D) = F(d) = W$$

The mapping for all components and connectors which are still in the actual product, but are not affected by the delta operations remain the same as before.

3.3 Component Risk Value Computation

After a PUT p_i has been generated and the mapping has been modified by applying a set of deltas Δ_{p_i}, we are able to perform the risk computation for each component. We first have to automatically compute each component's failure impact and failure probability as a manual assessment for each component does not scale for SPLs.

Component Failure Impact. As described earlier, each feature f_i of the SPL has been assigned with a feature impact value $FI(f_i) = [1, 5]$. We will use these feature impact values and the mapping from features to architectural elements to compute the failure impact of each component $c \in C_{p_m}$ for the current PUT. As we focus on integration testing, the goal is to assess components with a high impact value if they are related to important features or are interacting with components of high importance.

As we perform incremental testing, the failure impact value changes over the course of tested product variants P_{tested}. Due to the existing mapping, we are able to use the feature impact values to compute *component failure impact* values. For the final impact value of each component,

it is also of interest, with which components the component c_j interacts (i.e., its *neighbors*), indicated by either an incoming connector $con_{in} = \pi_{c_n \to c_j}$ or an outgoing connector $con_{out} = \pi_{c_j \to c_n}$ to a neighboring component c_n. We define a *component neighbor function* $NC : \mathcal{C}_{SPL} \times P \to \mathcal{P}(C)$ which returns all neighboring components of a component for a particular product variant p_m as:

$$NC(c_i, p_m) = \{c_j \in C_{p_m} | \exists con \in CON_{p_m} : con = \pi_{c_j \to c_i} \vee con = \pi_{c_i \to c_j}\}$$

As we focus on integration testing, we are interested in connectors which are connected to the component c_i as they enable the interaction with other components. Hence, we define the *interface* as function $Int : \mathcal{C}_{SPL} \times P \to \mathcal{P}(\mathcal{CON}_{SPL})$. The interface of a component c_i comprises all incoming and outgoing connectors to its neighbors $NC(c_i, p_j)$ in product variant p_j. We will use the interface information later to compute the component's failure impact value.

$$Int(c_i, p_j) = \{con' \in Con_{p_j} | \forall c_j \in NC(c_i, p_j), \pi \in \Pi : con' = \pi_{c_j \to c_i} \vee con' = \pi_{c_i \to c_j}\}$$

The final failure impact of a component c_i is based on the feature impact values for three sets of features. First, it depends on the feature impact value of the features influencing component c_i itself. This is computed using the already defined functions $FI_{\mathcal{F}}(F(c_i))$. In addition, we use the feature impact values of the features linked with the neighbor components $FI_{\mathcal{F}}(F(NC(c_i, p_j)))$ in the current PUT. The last set of features are those, which influence the interface connectors $FI_{\mathcal{F}}(F(Int(c_i, p_j)))$ of component c_i.

Based on these feature sets, we define a function *impact* : $\mathcal{C}_{SPL} \times P \to \mathbb{R}$ to compute an impact value for a component $c_i \in \mathcal{C}_{SPL}$. We normalize the result with the maximum feature impact value to obtain a result between 0 and 1.

$$impact(c_i, p_j) = \frac{FI_{\mathcal{F}}(\bigcup F(e, p_j))}{|\bigcup F(e, p_j)| \cdot max(\bigcup_{k=1}^{|\mathcal{F}|} FI_{\mathcal{F}}(f_k))}$$

where $c_i \in C_{p_j}, e \in (c_i \cup Int(c_i) \cup NC(c_i))$ holds.

Example 5. *Based on Fig. 1, Ex. 2 and Ex. 4, we compute the failure impact values for $B \in p_2$ as follows:*

$$impact(B, p_2) = \frac{FI_{\mathcal{F}}(\{W, X, Y\})}{|\{W, X, Y\}| \cdot FI_{\mathcal{F}}(Z)} = \frac{7}{3 \cdot 5} \approx 0.47$$

Component Failure Probability. In addition to the failure impact values $impact(c, p)$, we require a failure probability for each component $c_i \in C_{p_j}$ to compute the risk. We compute this value fully automatically, which improves the scalability of our approach compared to traditional manual risk-assessment. Thus, we have to identify the set of delta operations which influence a component c. We define the set of delta operations for component $c_i \in C_{p_j}$ as $Op_{c_i, p_j} = \{op_1, \ldots, op_k\}$, such that each *op* adds or removes connectors in the components interface $Int(c_i, p_j)$.

To compute a component's failure probability, we compare the occurrences of deltas in the current PUT with all previously tested variants P_{tested}. In particular, we compute the average of differences in the applied delta operations compared to the other tested variants. This allows to measure how the component differs compared to all previously tested variants. We argue that a component with a small average of differences has undergone a more profound test than a component with a high average, i.e., the component has not occurred in the current configuration in previous PUT. We define the failure probability of a component $c_i \in C_{p_j}$ as $fprob : \mathcal{C}_{SPL} \times P \times \mathcal{P}(P) \to [0, 1]$. It is defined using the average number of operations from all symmetric differences ($\oplus$) between the set of component's current delta operations Op_{c_i} for p_j containing c_i and the set of delta operations from all tested product variants $P_{tested} \subseteq P$ in which c_i was also present. We define the failure probability of a component $c_i \in C$ based on the set of product variants $P_{c_i} \subseteq P_{tested}$ in which c_i occurs as:

$$fprob(c_i, p_j, P_{c_i}) = \frac{\sum_{k=0}^{|P_{c_i}|} |Op_{c_i, p_j} \oplus Op_{c_i, p_k}|}{|P_{c_i}| \cdot max(\bigcup_{l=0}^{|P_{c_i}|} |Op_{c_i, p_j} \oplus Op_{c_i, p_l}|)}$$

where $P_{c_i} \neq \emptyset$ holds. If there is no tested product which contains c_i before, the probability $fprob(c_o, p_j, P_{c_i})$ will be set to a fixed value of 1. We normalize the result with the maximum symmetric difference value that has been measured with the comparison to all previous PUT. In addition, we argue that often occurring components are less likely to fail and, thus, we include the number of the components occurrences in previously tested variants in the normalization. Consequently, failure probability values are between 0 and 1, with 1 being the highest probability value. In our context, probability values of 1 do not indicate that a failure will definitely occur, but a higher value indicates a higher likelihood that a failure occurs.

Example 6. *Based on Ex. 2, we compute the failure probability for $B \in C_{p_2}$.*

$$fprob(B, p_2, P_B) = \frac{1}{2 \cdot 1} = 0.5$$

We use the impact and failure probability of a component to compute a *component risk value*. We define a function $risk : \mathcal{C}_{SPL} \times P \times \mathcal{P}(P) \to \mathbb{R}$ which returns the risk value for a component $c_i \in C_{p_j}$ when $p_j \notin P_{tested}$ as:

$$risk(c_i, p_j, P_{c_i}) = impact(c_i, p_j) \cdot fprob(c_i, p_j, P_{c_i})$$

Example 7. *Based on Ex. 5 and Ex. 6, we compute the risk for $B \in P_B$ for product p_2.*

$$risk(B, p_2, p_B) = 0.47 \cdot 0.5 = 0.235$$

3.4 Test Case Prioritization

As described in Sec. 2, we prioritize test cases defined as MSCs [17]. In our incremental test case prioritization technique, we compute a priority value for each applicable test case $tc \in TC_{p_j}$ for a product variant p_j. This value is computed using the component's risk values described in the previous subsection. As described in Sec. 2, we define a formula $prio_{sig} : \mathcal{TC} \times P \to \mathbb{R}$ which prioritizes test cases for a certain product variant by analyzing every signal used in the MSC and the components, between which the interaction happens. Similar to this, we compute the risk-based priority of test cases TC_{p_j} for product variant p_j using the function $prio_{risk} : \mathcal{TC} \times P \to \mathbb{R}$ as follows:

$$\forall tc \in TC_{p_j} : prio_{risk}(tc, p_j) = \frac{\sum_{c_n} \sum_{c_m} s(c_n, c_m) \cdot (risk(c_n, p_j, P_t) + risk(c_m, p_j, P_t))}{|\sum_{c_n} \sum_{c_m} s(c_n, c_m)|},$$

where $c_n, c_m \in C_{tc}$ and $P_t = P_{tested}$.

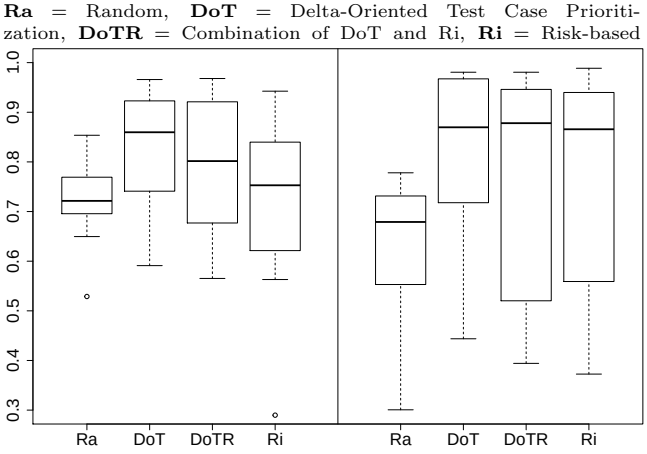

Figure 4: Boxplots for APFD for Arbitrary (left) and Important Faults (right) in Changed Interfaces

4. EVALUATION

We evaluated our approach to assess its effectiveness and efficiency. First, we will explain the research questions used as foundation for the evaluation. Next, the subject system to which we applied our technique is described. Afterwards, the methodology including the measured metric is explained. Finally, we present and discuss the results.

4.1 Research Questions

To investigate our technique, we formulate the following two research questions to be answered by our evaluation:

RQ1: *Is the risk-based prioritization able to outperform both, a random prioritization and an existing delta-oriented prioritization technique in terms of failure finding rate?* As we prioritize test cases, we want to assess the prioritization quality in terms of failure finding rate.

RQ2: *Are there benefits in terms of efficiency of the automatic computation of fault probabilities compared to existing techniques?* Traditional risk-based testing approaches require manual effort to define risk values. Thus, our technique should be more efficient.

4.2 Subject System

We use an automotive case study that describes a *Body Comfort System* (BCS) for evaluation. The BCS has been designed as SPL that comprises 11, 616 product variants [18]. We will focus on a subset of 18 variants in this work (including the core), which have been derived in previous work by a pairwise feature coverage approach [13, 20]. The product variants comprise different features, such as an *automatic power window*, a *finger protection* or a *remote control key*. BCS consists of an architecture model for the core product and a set of corresponding deltas. For communication, an average of 72 connectors are transferring 60 different signals between four and 19 components per product variant. Our approach is incrementally applied to the 17 derived variants. For testing, BCS provides a total of 92 MSCs as test cases.

4.3 Methodology

To assess the quality of our technique, we measure the *average percentage of faults detected* (APFD) metric defined by Rothermel et al. [22]. APFD measures how fast m faults are detected in a test set with n test cases. APFD considers the first position of a test case T_{fa_i} that reveals failure fa_i until all faults are covered and is defined as follows:

$$APFD = 1 - \frac{\sum_{i=1}^{m} T_{fa_i}}{nm} + \frac{1}{2n}$$

As BCS does not provide information about faults, we perform two different fault seeding experiments to assess the APFD of our technique [12]. Both experiments are designed with the assumption in mind that changed parts of the system have a higher likelihood to introduce faults [5].

Experiment 1. We randomly seed faults in connectors that correspond to *changed components*, i.e., a delta has been applied which influences the corresponding components in the current PUT. This resembles our assumption that changes introduce new faults. To reduce the number of faults per product variant to an average of 1.5 faults per variant, we compute the change ratio for each changed component, i.e., how large is the difference to previous occurrences of the component. Within the components, 10% of its interface connectors can be faulty. The chance that a connector is faulty depends on the change ratio, i.e., the more changes a component had, the higher the chance that a connector is faulty. In addition, we set an upper limit of 5 faults per component.

Experiment 2. Similar to Experiment 1, we seed faults in connectors that are part of a changed component's interface. In contrast to the previous experiment, we only allow for the fault seeding to influence components which are mapped to features with an impact value of 3 or higher. This resembles the idea that our technique shall be able to find *important faults* as early as possible. The distribution of faults is as described in Experiment 1.

To avoid statistical outliers, we repeat both experiments 100 times with random fault seeds for all product variants and compute the average APFD achieved over all 100 repetitions. As a baseline, we compare the APFD of our technique to the existing delta-oriented technique [17], a combination of the delta-oriented technique and the risk-based test case prioritization as well as a random approach. To combine the delta-oriented technique with our novel approach, we adapt the way that component weights are computed as follows:

$$w_{risk}(c_j, p_j) = w(c_j, p_j) + \lambda \cdot risk(c_j),$$

where λ is a weighting factor between 0 and 1. To avoid outliers in the random prioritization, we repeated it 100 times for each product variant and computed the average APFD.

4.4 Results

RQ1. We analyze how the risk-based test case prioritization performs in terms of APFD for seeded faults in BCS. Thus, for both experiments presented in the previous subsection, we measure how fast faults are covered. In total, we compare four different prioritization approaches: Random test case prioritization, delta-oriented test case prioritization, risk-based test case prioritization and a combination of both, delta-oriented and risk-based test case prioritization. In Fig. 4, we show on the left side the results for all 100 repetitions of experiment 1 (cf. Sec. 4.3).

As we can see, the risk-based technique achieves good APFD results with a median value of 0.75. However, the previously introduced delta-oriented test case prioritization [17]

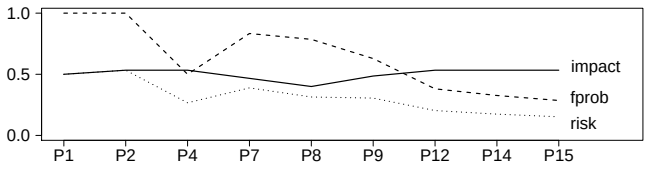

Figure 5: Failure Impact, Failure Probability and Risk Value for component AutoPW

labeled with *DoT* achieves better results with an APFD of 0.86. This is due to the fact, that our previous technique explicitly focuses on changes between components, while the risk-based technique focuses on overall important parts of the PUT. The combination of both techniques (labeled with *DoTR*) leads to results between both approaches used in separation, improving the results of the risk-based technique. The random technique labeled with *Ra* achieves good results, but falls short compared to the other techniques with a median of 0.72.

To evaluate, whether the risk-based technique is able to find important faults, we performed the second experiment with different fault seedings, focusing on important components in the set of changed components (cf. Sec. 4.3). The results are shown in Fig. 4 on the right side. Similar to experiment 1, each boxplot comprises all 100 repetitions of our analysis. While the random technique still is able to perform surprisingly well with an APFD median of 0.67, we notice that the risk-based technique has improved drastically with a median APFD of 0.86. While we expected an improvement to some extent, the results show that our technique is able to automatically identify test cases that correspond to important parts of the system, with a high chance to reveal these important faults first. The previous delta-oriented prioritization still performs well, leading to a median APFD of 0.84. However, when combining the risk-based and delta-oriented test case prioritization, we receive the best result with a median of 0.87. This is due to our focus on changes and, within these, the most important ones.

Summarizing, we argue that our risk-based testing approach outperforms a random prioritization in both experiments and improves the APFD compared to the previous, delta-oriented approach when looking at important faults in the system, which is the main goal of risk-based testing.

RQ2. The answers to RQ1 show that our technique is able to detect faults that are important and can be combined to the existing delta-oriented test case prioritization. Besides the fault finding capabilities, the technique's efficiency is also of importance. As we describe earlier, one main contribution of our approach is the automatic derivation of failure impact and failure probabilities for each component in each product variant. While the failure impact values are based on feature impact values defined manually in the design phase, the failure probabilities are computed fully automatic. In Fig. 5, the change of the failure impact, failure probability and risk value for the automatic power window component *AutoPW* in the corresponding 9 product variants from BCS is shown. The results indicate that impact values are similar for the different PUTs. The failure probability and the risk value decreases in case that product variants, containing the component, were tested.

Using traditional approaches [9], this would resolve in a very high manual effort which is infeasible for large quantities of product variants under test. In contrast, our technique has shown that is able to compute individual risk values for each component in each product variant based on previously tested variants in a very efficient manner. Within our case study, the manual effort contained the assignment of 27 feature impact values to features and 32 mappings from features to components and connectors for the core product. On the contrary, our automated computation performs 1627 mappings from features to components and connectors and 181 failure impact and 181 failure probability computations in 17 product variants (excluding the core) for BCS. Consequently, we argue that we are able to elevate the concept of risk-based testing successfully to SPLs while avoiding the immense manual effort that the application of traditional risk-based techniques would introduce for SPLs.

4.5 Threats to Validity

As BCS does not provide realistic failures we have to use seeded failures for APFD measurement. This reduces the generalizability of our findings. However, as we repeat two different experiments and compare different techniques, we are able to present first indications that our approach works sufficiently well and, thus, an investigation using realistic case studies should be the next step. Furthermore, the number of test cases and product variants evaluated is rather small. This might influence the results. However, we argue that for larger test set sizes the difference between random and our technique will improve further as the chance to randomly find a failure revealing test case decreases.

5. RELATED WORK

Test case prioritization has been explored in the past by several authors as popular regression testing technique next to selection and minimization [26].

Risk-based testing (RBT) has been explored in literature for single software systems. According to a recent systematic review of risk-based testing approaches [10], several specific risk-based selection and prioritization techniques have been reported by different authors. Yoon and Choi [27] propose a RBT test case prioritization technique on the basis of risk exposure values, which are estimated by experts and correlated to test cases. Bai et al. [2] are able to extract failure probabilities and importances from Web services based on semantic information, formalized in an ontology to prioritize and select tests for Web service evaluation. Kloos et al. [15] construct test models represented as state machines. They perform a fault tree analysis on these models such that test cases can be derived, selected and prioritized according to the severity of the identified risks and the basic events that cause it. The focus of their approach are safety-critical systems. Felderer et al. [7] integrate an automated model-based risk-assessment based on metrics with manual risk assessment procedures. Stallbaum et al. [25] present an automated test case generation and prioritization based on risk-based testing using activity diagrams. All of these techniques focus on single software systems without using delta-knowledge to avoid redundancy between variants.

Hartmann et al. present a risk-based testing approach for SPLs [11]. They introduce the concept of assigned risk values and failure probabilities to features in a feature model. In contrast to our work, there are no sophisticated computations involved as the approach is manually performed.

6. CONCLUSION AND FUTURE WORK

Concept. In this paper, we presented a novel approach for test case prioritization based on risk-based testing. While providing feature impact values manually, our technique is able to automatically compute *component failure impact* and *component failure probabilities* for each product variant under test automatically. Thus, our technique enables risk-based testing of SPLs with justified effort compared to previous manual techniques as the manual estimation of risk values for each component of an arbitrary number of product variants is infeasible. Our evaluation shows that our technique is able to effectively find important failures. APFD shows desirable results, improving the failure finding rate compared to other techniques.

Future Work. Risk-based testing for SPLs is still an unexplored area which requires further investigations. For future work, we want to investigate how the technique performs on large-scale real-life case studies. We also want to investigate on how other information can be integrated into our approach to improve the risk computation even more.

7. REFERENCES

[1] S. Amland. Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *J. Sys. and Soft.*, 53:287–295, 2000.

[2] X. Bai, R. S. Kenett, and W. Yu. Risk assessment and adaptive group testing of semantic web services. *Int. J. of Softw. Eng. and Knowledge Eng.*, 22(05):595–620, 2012.

[3] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. *Automated Reasoning on Feature Models*, pages 491–503. Springer Berlin Heidelberg, 2005.

[4] D. Clarke, M. Helvensteijn, and I. Schaefer. Abstract Delta Modelling. 25(3):482–527, 2015.

[5] J. A. Duraes and H. S. Madeira. Emulation of software faults: A field data study and a practical approach. *IEEE Trans. Soft. Eng.*, 32(11):849–867, 2006.

[6] E. Engström. *Exploring Regression testing and software product line testing - research and state of practice*. Lic dissertation, Lund University, May 2010.

[7] M. Felderer, C. Haisjackl, R. Breu, and J. Motz. Integrating manual and automatic risk assessment for risk-based testing. *Software Quality Process Autom. in Softw. Develop.*, pages 159–180, 2012.

[8] M. Felderer, C. Haisjackl, V. Pekar, and R. Breu. A risk assessment framework for software testing. In *Proc. Int'l Symposium Leveraging App. of Formal Meth., Verific. and Validation*, pages 292–308. 2014.

[9] M. Felderer, C. Haisjackl, V. Pekar, and R. Breu. An exploratory study on risk estimation in risk-based testing approaches. In *Software Quality Days (SWQD)*, pages 32–43. 2015.

[10] M. Felderer and I. Schieferdecker. A taxonomy of risk-based testing. *Int. J. Softw. Tools Technol. Transf.*, 16(5):559–568, 2014.

[11] H. Hartmann, F. van der Linden, and J. Bosch. Risk based testing for software product line engineering. In *Proc. Int'l Software Product Line Conference*, pages 227–231, 2014.

[12] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Trans. Soft. Eng.*, 37(5):649–678, Sept 2011.

[13] M. F. Johansen, Ø. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *Proc. Int'l Software Product Line Conference*, pages 46–55, 2012.

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, S.E. Institute - Carnegie Mellon University, 1990.

[15] J. Kloos, T. Hussain, and R. Eschbach. Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In *Proc. Int'l Conf. Softw. Testing, Verific. and Valid. Workshops*, pages 26–33, 2011.

[16] R. Lachmann, S. Lity, M. Al-Hajjaji, F. E. Fürchtegott, and I. Schaefer. Fine-grained test case prioritization for integration testing of delta-oriented software product lines. In *Proc. Int'l Workshop Feature-Oriented Software Development*, 2016.

[17] R. Lachmann, S. Lity, S. Lischke, S. Beddig, S. Schulze, and I. Schaefer. Delta-oriented test case prioritization for integration testing of software product lines. In *Proc. Int'l Software Product Line Conference*, pages 81–90, 2015.

[18] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer. Delta-oriented software product line test models - the body comfort system case study. Technical report, TU Braunschweig, 2013.

[19] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz. Delta-oriented model-based integration testing of large-scale systems. *J. Sys. and Soft.*, 91:63–84, 2014.

[20] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. le Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Jour.*, 20(3-4):605–643, 2012.

[21] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering*. Springer, 2005.

[22] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Soft. Eng.*, Vol.27 No.10:929–948, 2001.

[23] I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella. Delta-oriented programming of software product lines. In *Proc. Int'l Software Product Line Conference*, pages 77–91, 2010.

[24] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela. Software Diversity: State of the Art and Perspectives. *Int. J. Softw. Tools Technol. Transf.*, 14(5):477–495, 2012.

[25] H. Stallbaum, A. Metzger, and K. Pohl. An automated technique for risk-based test case generation and prioritization. In *Proc. Int'l Workshop Automation of Softw. Test*, pages 67–70, 2008.

[26] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120, 2007.

[27] H. Yoon and B. Choi. A test case prioritization based on degree of risk exposure and its empirical study. *Int. J. of Softw. Eng. and Knowledge Eng.*, 21(02):191–209, 2011.