Fine-Grained Test Case Prioritization for Integration Testing of Delta-Oriented Software Product Lines

Remo Lachmann[1] Sascha Lity[1] Mustafa Al-Hajjaji[2] Franz Fürchtegott[1] Ina Schaefer[1]

[1]Technische Universität Braunschweig, [2]University of Magdeburg, Germany

{r.lachmann, s.lity, f.fuerchtegott, i.schaefer}@tu-bs.de, m.alhajjaji@iti.cs.uni-magdeburg.de

Abstract

Software product line (SPL) testing is a challenging task, due to the huge number of variants sharing common functionalities to be taken into account for efficient testing. By adopting the concept of regression testing, incremental SPL testing strategies exploit the reuse potential of test artifacts between subsequent variants under test. In previous work, we proposed delta-oriented test case prioritization for incremental SPL integration testing, where differences between architecture test model variants allow for reasoning about the execution order of reusable test cases. However, the prioritization left two issues open: (1) changes to component behavior are ignored, influencing component interactions and, (2) the weighting and ordering of similar test cases result in an unintended clustering of test cases. In this paper, we extend the test case prioritization technique by (1) incorporating changes to component behavior allowing for a more fine-grained analysis and (2) defining a dissimilarity measure to avoid clustered test case orders. We prototyped our test case prioritization technique and evaluated its applicability and effectiveness by means of a case study from the automotive domain showing positive results.

Categories and Subject Descriptors D.2.5 [*Software Engineering*]: Testing and Debugging

Keywords Delta-Oriented Software Product Lines, Test Case Prioritization, Model-Based Integration Testing

1. Introduction

Software product lines (SPL) facilitate to capture individual customer demands for products by introducing variability to software development in large scales [26]. An SPL comprises a family of similar software systems sharing common and variable *features*, i.e., customer visible system functionality. Due to valid feature combinations, the number of potential variants increases exponentially [26] making their efficient testing a very challenging task [10]. Hence, SPL testing techniques require to exploit the commonality between variants to reduce the inherent testing redundancy.

As testing is limited by budget and time, test effort has to be reduced. Testing product lines from product to product, i.e., incrementally, reduces testing effort and adopts concepts of *regression testing* [35] for SPL testing. The commonality and variability between variants is exploited to reduce redundant testing [11]. In previous work, we proposed an incremental SPL integration testing strategy for *test case prioritization* (TCP). It is based on delta modeling [6], allowing for the reuse and incremental adaptation of test sets. Deltas describe transformations between product variants, allowing for a focus of newly introduced changes in the current product variant under test. Based on changes (i.e., deltas) between variant-specific architecture test models, we computed priorities for the ranking of reusable test cases. In context of this and previous work, test cases describe communications between components based on exchanged signals.

However, the existing TCP left two crucial issues open. First, it ignores changes to component behavior. Components may not change between variants on the architecture level, but on their behavioral level and may influence the inter-component communication. To this end, we extend our approach to take information about changes on component behavior level into account. Second, the previous technique does not consider the similarity between test cases. Accordingly, always selecting the next highest weighted test case in the ordering may lead to an unintended clustering of test cases based on similar weights for similar test cases. To this end, we introduce a dissimilarity-based TCP, which is combinable with the delta-oriented TCP.

We contribute the following:

- A fine-grained analysis of changed components based on behavioral changes and their impact. We incorporate changes to component behavior, e.g., obtained from the already applied component testing [19, 20], to allow for a fine-grained test case prioritization.

FOSD'16, October 30, 2016, Amsterdam, Netherlands

http://dx.doi.org/10.1145/3001867.3001868

- A dissimilarity measure to avoid a clustering of the prioritized test cases for SPL integration testing. The smaller the number of identical signals exchanged in test cases, the more dissimilar they are.
- An evaluation of our technique in terms of TCP quality. We compare it to a random ordering and our previous TCP technique [16].

2. Foundations

Delta-Oriented Software Product Lines. Delta modeling [6] is a transformational variability modeling technique for model-driven SPL development. For all variants $P_{SPL} = \{p_{core}, p_1, \ldots, p_n\}$ of an SPL, their variant-specific model m_{p_i} is defined by its differences to a designated *core model* $m_{p_{core}}$ of a *core variant* p_{core}. Those differences specified by means of transformations, called *deltas*, are additions/ removals of model elements used to transform the core model into the particular variant-specific model. Each delta δ captures either an addition or a removal of an element e. For every variant $p_i \in P_{SPL}$, a predefined set of deltas $\Delta^{\mathcal{M}}_{p_i} \subseteq \Delta^{\mathcal{M}}_{SPL}$ exists for an automated generation of the corresponding model by applying each $\delta \in \Delta^{\mathcal{M}}_{p_i}$ consecutively in a predefined order. By $\Delta^{\mathcal{M}}_{SPL}$, we refer to the set of all valid deltas of the current SPL and model domain $\mathcal{M}$. The concept of delta modeling allows for the derivation of differences between arbitrary variants, e.g., architectures, encapsulated in *model regression deltas* $\Delta^{\mathcal{M}}_{p_i,p_j} \subseteq \Delta^{\mathcal{M}}_{SPL}$ by taking their delta sets $\Delta^{\mathcal{M}}_{p_i}$ and $\Delta^{\mathcal{M}}_{p_j}$ into account. We refer to prior work, for the derivation of model regression deltas [19, 20].

We adapted delta modeling in prior work to architecture models [20] as well as behavioral models, i.e., state machines [19]. An architecture defines the structure of a system by specifying the computational entities, i.e., *components*, and their explicit communication dependencies by means of *connectors*. Thus, an *architecture model* $arc = (C, Con, \Pi)$ comprises a finite set of components $C = \{c_1, \ldots, c_m\}$, a finite set $Con = \{con_1, \ldots, con_l\}$ of connectors, and a finite set $\Pi = \{\pi_1, \ldots, \pi_k\}$ of *signals* transmitted via connectors, allowing for component interaction. A connector $con = \pi_{c \to c'}$ specifies an unidirectional interaction between its source component c sending the signal π and its target component c' receiving π. Thus, for each component $c \in C$, we are able to derive the set of *incoming connectors* I_c and the set of *outgoing connectors* O_c comprising all connectors, where c is the target or source component. The sets of incoming and outgoing connectors define a components *interface*. For the specification of variable architecture models, we require a core architecture $arc_{p_{core}}$ and a set of *architecture deltas* $\Delta^{\mathcal{ARC}}_{SPL}$ to allow for the generation of variant-specific architecture models arc_{p_i}. A corresponding architecture delta $\delta \in \Delta^{\mathcal{ARC}}_{SPL}$ captures either an addition or a removal of components/connectors.

EXAMPLE 1. *Consider the sample architecture model arc_{p_0} in Fig. 1 used as core. By applying the set of architecture deltas $\Delta^{\mathcal{ARC}}_{p_1}$, arc_{p_0} is transformed into arc_{p_1}. To step from arc_{p_1} to arc_{p_2}, we derive and apply the architecture model regression delta $\Delta^{\mathcal{ARC}}_{p_1,p_2}$.*

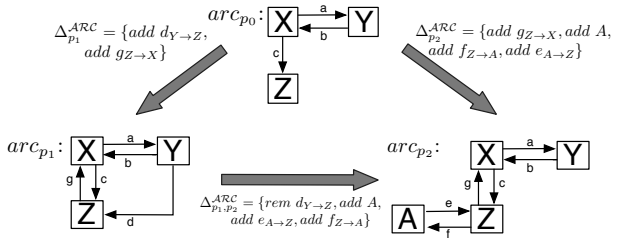

Figure 1. Concept of Delta-Oriented Architectures

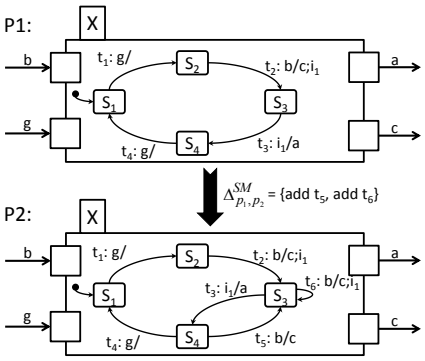

Figure 2. Concept of Delta-Oriented State Machines

In contrast to architecture models, behavioral models, such as state machines, are used to specify the behavior of components, i.e., defining the reaction on incoming signals with corresponding outgoing signals. A *state machine* $sm = (S, T, E)$ comprises a finite set of states $S = \{s_1, \ldots, s_o\}$ representing execution states of the component and a finite set of transitions $T = \{t_1, \ldots, t_u\}$ defining the transfer between states based on *events* from the finite set of events $E = \{e_1, \ldots, e_v\}$. The set of events $E = E_I \cup E_O \cup E_\tau$ is divided into distinct sets of *input events* E_I, of *output events* E_O and *internal events* E_τ. We assume identical names of signals and events to specify the mapping between input/output signals and events. Internal events are solely used and visible within a state machine to allow for behavior. To enable the transfer between states, transitions are triggered based on input, whereas each transition may generate outputs as reaction. The syntax for transition labels is $t : e_i/\{e_o, \ldots\}$, where $e_i \in E_I \cup E_\tau$ describes the triggering event of transition t in addition to a set of output events e_o sent by the transition. For the specification of variable state machines, we require a core state machine $sm_{p_{core}}$ and a set of *state machine deltas* $\Delta^{\mathcal{SM}}_{SPL}$ to allow for the generation of variant-specific state machines sm_{p_i}. A corresponding state machine delta $\delta \in \Delta^{\mathcal{SM}}_{SPL}$ captures either an addition or a removal of states or transitions.

EXAMPLE 2. *In Fig. 2, the effect of the transformation of* arc_{p_1} *to* arc_{p_2} *for component X from Ex. 1 is shown. On the architecture model level, X has no interface changes, but on its behavioral level, we transform its state machine by applying the state machine regression delta* $\Delta^{\mathcal{SM}}_{p_1,p_2}$.

Incremental SPL Integration Testing. State machines and architecture models can be used as *test model specifications* for *model-based component* [32] and *integration testing* [5]. For SPLs, we applied their delta-oriented versions for *incremental* component and integration testing in prior work [19, 20]. In this paper, we solely focus on SPL integration testing. However, we incorporate the information about delta transformations on component state machines.

For incremental SPL integration testing [20], we use delta-oriented architecture models to specify variant-specific test models arc_{p_i} for each $p_i \in P_{SPL}$. To guide the test process of variants, we apply structural coverage criteria, e.g., all-component or all-connector coverage, to derive a set of *test requirements* to be covered by a set of variant-specific *test cases* $TC_{p_i} = \{tc_1, \ldots, tc_n\} \subseteq \mathcal{TC}$ called test set. By $\mathcal{TC}$, we refer to the set of all test cases for an SPL under test. A test case tc defines an interaction scenario between components of a corresponding architecture test model. Similar to prior work [16, 20], we use *Message Sequence Charts* (MSC) [13] as test cases. An MSC captures a communication scenario within the architecture. Each MSC comprises a set of components $C_{tc} \subseteq C$ and connectors $Con_{tc} \subseteq Con$ for the specification of component interactions, where $s(c_i, c_j) \subseteq \Pi_{tc} \subseteq \Pi$ defines the set of signals exchanged between components $c_i, c_j \in C_{tc}$.

The incremental testing workflow is defined as follows [20]: We test p_{core} first by applying standard model-based integration testing [5]. The remaining variants $p_i \in P_{SPL}$ are tested based on their predecessors, where we exploit the reuse potential of test artifacts. We adopt the concept of delta modeling [6] to define regression deltas for the set of test artifacts, e.g., architectural test models. By stepping from variant p_{i-1} to the subsequent p_i under test, we use the derived regression deltas to adapt the variant-specific test artifacts. A crucial part of the adaptation is the decision whether previously executed test cases $tc_j \in TC_{p_{i-1}}$ can be reused for p_i and if they have to be re-executed to validate that changes have no unintended influences on already tested behavior. Therefore, we categorize test sets TC_{p_i} similar to regression testing [35] into sets of *new*, *reusable*, and *obsolete* test cases. New test cases are defined for a variant to test its new untested functionality. Reusable test cases have been executed on previous variants and are also valid for current variant p_i. Obsolete test cases are not valid and removed from the test set, but are stored for subsequent testing steps. To identify the category of a given test case, the components and connectors required by the test case are analyzed. If at least one component or connector is no longer present in the test case, it is obsolete for the current variant. Otherwise, the test case is reusable. From the set of reusable test cases, we can either select certain test cases to be re-executed [20] or prioritize test cases for re-execution [16]. Testing steps are repeated until all selected variants are tested.

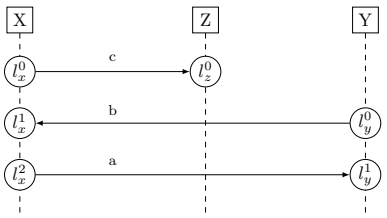

Figure 3. Sample Integration MSC Test Case

EXAMPLE 3. *Consider core* arc_{p_0} *in Fig. 1. We apply all-connector coverage, which requires the coverage of the three contained connectors. Test case* tc_1 *shown in Fig. 3 covers all three connectors and is reusable for testing* p_1 *and* p_2.

SPL Integration Test Case Prioritization. In regression testing, various techniques exist to reduce the overall testing effort by applying prioritization, selection and/or minimization to test cases [35]. While the selection and minimization of test cases aim to reduce the size of test sets to be executed by deriving a representative subset of all test cases, prioritization allocates a priority value to each test case. This value allows to order test cases such that the most important test cases w.r.t. given criteria, e.g., potential fault detection capability, are executed first. Based on TCP, the testing process can stop at any time according to available resources, ensuring the most important test cases have been executed.

In prior work [16], we proposed TCP for incremental integration testing of delta-oriented SPLs. The technique takes the commonalities and differences between subsequent variants under test based on their architecture regression deltas into account. A test case gets a higher priority the more its respective interaction scenario covers changed elements. For priority computation, we determine the changes applied for the very first time when stepping to subsequent variants. Focusing on never before tested changes reduces the redundancy between testing different product variants. Those changes are captured in the set of *changed incoming connectors* $IC_c \subseteq I_c$ and the set of *changed outgoing connectors* $OC_c \subseteq O_c$ for a component $c \in C_p$ of variant p and represent changes to the component interface derivable from applied deltas. We use both sets for the computation of component weights as follows.

$$w(c) = \alpha \cdot \frac{|IC_c|}{|I_c|} + \beta \cdot \frac{|OC_c|}{|O_c|} + \gamma \cdot \frac{|MPD_c|}{|P_c|}$$

We use α, β, γ as weighting factors to control the impact on each part of the function, where $\alpha + \beta + \gamma = 1$ holds. These factors can be adjusted by the tester. We normalize the

changed incoming and outgoing connectors by the size of all incoming (I_c) and outgoing connectors (O_c) of the component, respectively. By MPD_c, we refer to changes denoted by *multi product deltas* (MPD) [16]. These types of deltas occur when we compare different product variants at once. They describe changes, which are induced by the combination of deltas, which have already been tested in isolation, but never in their current combination. In particular, the interface of a component c has never been tested in its current configuration before, even though all related deltas have been covered by tests in previous product variants. We normalize MPD values by the number of product variants the component has occurred in all product variants under test thus far, denoted by P_c.

For TCP, we incorporate the component weights of a product variant in two different ways. First, we defined a *component-based prioritization*

$$prio(tc) = \frac{\sum_{j=1}^{n} w(c_j)}{n},$$

where $c_j \in C_{tc}$ and $n = |C_{tc}|$ holds, i.e., we sum all weights of components contained in a test case to be prioritized. The component-based prioritization focuses only on the components covered by a test case. Second, we defined a *signal-based prioritization*

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

where n is the number of signals a test case comprises, i.e., component weights are multiplied by the number of covered signals between components.

We identified two open issues for the previous TCP: Behavioral changes are not incorporated and similar test cases result in clusters in the ordering.

3. Extended Prioritization Concept

The fine-grained TCP for integration testing is based on the analysis of structural and behavioral deltas. We also introduce a dissimilarity-based approach to accelerate coverage of important system parts. To successfully apply the prioritization, test cases have to be defined for all product variants under test. Test case design or generation is not in the scope of this paper. For example, they could be manually defined or derived from the test models. To prioritize test cases for product variants, a set of variants has to be selected and ordered a priori. We do not focus on how to select product variants, but assume that they are available, e.g., using existing product selection techniques [15, 24].

The overall TCP process is shown in Fig. 4. It is defined for its application on delta-oriented SPLs. However, if an explicit delta information between variants is not available, we require to extract this information about differences from the SPL, e.g., by applying model differencing techniques [25].

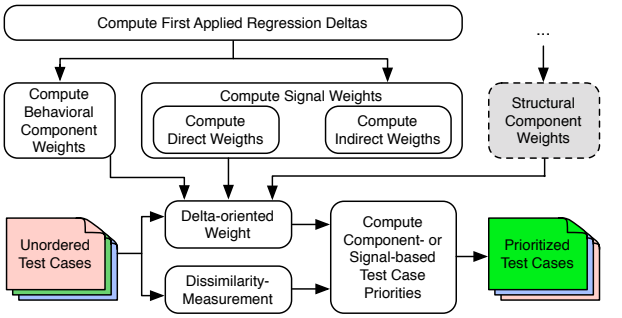

Figure 4. Fine-Grained Test Case Prioritization

3.1 Behavioral Component Weights

For integration testing, changes between components are essential to decide what to retest between variants. However, we did not incorporate internal changes on behavioral level in prior TCP techniques. In this work, we analyze the differences between state machines of an architecture model arc_{p_i} for product variant p_i to derive a *behavioral component weight*. A state machine represents the corresponding behavior of a component. Typically, the analysis begins with the first product variant when stepping from the core product to the next. The analysis is part of the incremental testing process for all product variants P_{SPL} under test.

Similar to the existing structure-based TCP (cf. Sec. 2), we focus on deltas $\Delta_{new} \subseteq \Delta_{SPL}^{\mathcal{SM}}$ never been applied before in prior tested variants $P_{tested} \subseteq P_{SPL}$ for the computation of a behavioral component weight. For state machines, we refine the occurring changes within a component c_i into *changed transitions* $CT_{c_i} \subseteq T_{c_i}$ and *changed states* $CS_{c_i} \subseteq S_{c_i}$ as these influence the behavior of c_i. In particular, a change is only considered for the component weight of the current product variant, if it occurs for the first time in the regression delta $\Delta_{P_{tested},p_j}^{\mathcal{SM}}$ of the current variant p_j to all prior tested variants P_{tested}, i.e., the corresponding delta operations have never been applied before [16]. These new regression deltas represent functionality that occurs for the first time in a product variant under test and, thus, is not covered in P_{tested}. The more of these new changes are detected for a component, the more it should be tested, as the behavior of the component and, thus, the communication with other components might have changed unwillingly.

These two types of changes, i.e., state changes CS_{c_i} and transition changes CT_{c_i} of a component c_i are computed to measure the degree of changes. The result is normalized by the components complexity, i.e., its number of states S_{c_i} and transitions T_{c_i}. This leads to the *behavioral component weight*, defined by the function $w_b : \mathcal{C} \rightarrow \mathbb{R}$. The higher the resulting weight, the more important is a test of this component. The behavioral component weight for $c_i \in C_{p_j}$ is computed as:

$$w_b(c_i) = \frac{|CT_{c_i}| + |CS_{c_i}|}{|T_{c_i}| + |S_{c_i}|}$$

The behavioral weight can be combined with the structural component weights $w(c)$ for a certain product variant to create a more fine-grained weight computation. This allows to capture both, changes on structural and behavioral level according to delta transformations. In particular, it enables us to detect changes that only occur within components and are invisible on architectural level. To further adjust the influences of the behavioral weight, we introduce a behavioral weighting factor ζ with $0 \leqslant \zeta \leqslant 1$, such that $\alpha + \beta + \gamma + \zeta = 1$. The other three factors are defined in $w_{comb}(c_i)$ (cf. Def. 1). The *combined component weight* is computed as follows:

$$w_{comb}(c_i) = w(c_i) + \zeta \cdot w_b(c_i)$$

3.2 Signal Weights

While each component receives a combined weight $w_{comb}(c)$ according to the applied structural and behavioral deltas, the signals exchanged between components might also have been influenced by delta operations. This is due to the fact, that *events* within the state machines of a component can be identified with *signals* on architectural level. Thus, changes to state machine events might influence signals, which might affect the communication with other components. To this end, we introduce delta-oriented *signal weights*, which influence the test case priority based on the *impact* of deltas on behavioral level. Basically, signal weights represent how much an internal change of a component and its signals influences the communication by this component on architectural level. For example, a newly introduced transition in a state machine representing the internal behavior of a component might be triggered by an incoming event from the components interface. We assume that this change to the signal has an influence on the behavior and might change the output of reusable test cases compared to previous tested product variants P_{tested}. Hence, we compute the number of deltas that directly modify an event, which corresponds to a signal, as *direct signal weight*.

In case of a behavioral change, it is of interest to identify any outgoing events, which are influenced by a state machine adaption of an input event. Our technique uses a slicing inspired technique [2] to analyze the impact between incoming and outgoing signals as *indirect signal weights*. Both analysis techniques and the resulting weight computations are described in the following.

Direct Signal Weights. A first applied delta $\delta \subseteq \Delta_{new}$ influencing a state machine might also influence a signal. That is, a transition might be added using a signal as incoming or outgoing event, or a transition has been modified or removed. For each signal $\pi \in \Pi$, which is received or send by a component $c \in C_p$ for product p, we count the number of transition-related deltas $\Delta_t \subseteq \Delta_{new}$ that contain an event e mapped to the signal on product level. The reason for a product-level analysis is that the incoming signal of one component is the outgoing signal of another component and signals can be reused by different components. We do not compute weights for internal events $e \in E_\tau$ as they are only observable within the component. As our TCP approach focuses on integration testing, only signals used for communication are visible within test cases. Hence, internal events do not have an impact for the TCP, as they are not visible on test case level. To measure the *direct signal weight*, we count the occurrences of a signal $\pi \in \Pi$ in all transition-related deltas using the function $count : \Pi \times \Delta \rightarrow \mathbb{N}$, i.e., we analyze deltas which add or modify (e.g., by adding or removing incoming or outgoing events) transitions in the current product variant. In addition, we only focus on deltas which have never been applied before in the previous product variants under test, i.e., $\Delta_t \subseteq \Delta_{new}$. To normalize these values, we compute the number of all signals received or transferred by component c_i, denoted as $\pi(c_i)$. These signals are derived from the components interface. Based on these values we compute a *direct signal weight* for each signal $\pi \in \Pi$ for a component $c_i \in C_{p_j}$. The direct signal weight is computed globally for one product variant by the function $w_{direct} : \Pi \times P_{SPL} \rightarrow \mathbb{R}$ as follows:

$$w_{direct}(\pi_k, p_j) = \sum_{i=1}^{|C_{p_j}|} \frac{count(\pi_k, \Delta_t)}{|\pi(c_i)|}, c_i \in C_{p_j}, \pi_k \in \Pi_{p_j}$$

Indirect Signal Weights. Alongside the *direct* signal weights, a change within a state machine may also indirectly influence other signals, which depend on the changed parts. On architectural level, this could lead to problems in communication, in case outgoing signals are indirectly influenced. Hence, we compute *indirect signal weights* to cope with these potential pitfalls by analyzing the state machine.

To measure the *indirect influences* for a certain component in a product variant, we have to analyze different paths within the state machine, which start with an influenced element, e.g., a changed transition trigger. First, we identify transitions t which have been changed by a delta $\delta \in \Delta_{new}$. Next, the incoming event e_i of this transition is checked. If the event is an input signal of the component, we start to create possible paths beginning from this transition. The reason for this is, that in integration testing, we are interested in input signals influencing output signals on architectural level. Hence, we want to find those pairs of signals which are contained in the relation $influence_c \subseteq I_c \times O_c$. In other words, we are interesting if the influence of an incoming connector is indirectly related to an outgoing connector, which is caused by an influences path within the state machine connecting both corresponding interface parts.

To find these indirect influence paths, an event e_{sent} sent by the same transition is stored in a set $E_{visited}$. Within the state machine, a sent event is visible in the next computation step of the system. Therefore, we start to examine for all states which are reachable via the first transition, what further transitions could be taken using e_{sent} as incoming event. If such a transition is found, the process is repeated,

meaning that all new outgoing events are stored in $E_{visited}$ and used as incoming events in the following steps.

As usual for such slicing related techniques, a stop criterion is necessary [2]. We stop if already traversed elements are reached again, or if the current transition has a trigger event $e_{current}$ which matches to an incoming signal on architectural level, i.e., $e_{current} \in E_I$. In other words, if an external event is detected in the current path, it breaks the data flow which has been started by the original external event e_i. Once no more elements can be traversed we compute the impact of a signal by using the function $impact : \Pi \times \mathcal{P}(Con \times Con) \rightarrow \mathbb{N}$. It counts all occurrences of events e_{π_i} related to the signal π_i in the *influence* relation. Hence, all events in $E_{visited}$, which have been detected on the traversed transitions, are inspected if they are part of the outgoing signals of the component, i.e., if there exists an $\pi_k \in O_c$ for $e_k \in E_{visited}$. We normalize these results over the number of influences. To this end, we introduce the indirect signal weight function $w_{indirect} : \mathcal{C} \times \Pi \rightarrow \mathbb{R}$ as:

$$w_{indirect}(c_i, \pi_j) = \begin{cases} \frac{impact(\pi_j, influence_{c_i})}{|influence_{c_i}|} & if\ influence_{c_i} \neq \varnothing \\ 0 & otherwise \end{cases}$$

Based on both, direct and indirect signal weights, we are able to compute a final weight for each signal of a product variant. As the two different weights effect the priority differently, we define two factors λ and μ to adjust the ratio of influence. We define the behavior-based signal weight function $w_s : \mathcal{C} \times \Pi \rightarrow \mathbb{R}$ for a signal π_j considering component c_i and two factors $\lambda + \mu = 1$ as follows:

$$w_s(c_i, \pi_j) = \lambda \cdot w_{direct}(c_i, \pi_j) + \mu \cdot w_{indirect}(c_i, \pi_j)$$

EXAMPLE 4. *Consider the sample state machine in Fig. 2. For variant p_2, the direct signal weight of* b *is* 0.5*, as it is used as event by transition t_5 and t_6 normalized over a total of four signals used in the component. In addition, although signal* a *has no direct weight, we are able to detect an indirect weight for* a*, as there is a new path $t_6 \rightarrow t_3$ based on delta application, i.e., t_6 is added to the state machine indirectly influencing* a *sent by t_3.*

3.3 Test Case Dissimilarity

In prior work [16] we noticed that a weight-based prioritization leads to potential redundancy between different test cases with similar weights. Currently, the potential redundancy is ignored by the weight-based TCP. This results in *clusters* of very similar test cases in the prioritization order as these test cases have very similar weights. A potential result is a decrease of the fault detection rate as testing might focus too much on the same parts of the system. Hemmati et al. [14] report that a dissimilar test case selection for single system model-based testing detect more faults than a selection of similar ones. Thus, we argue that the similarity of test cases should be considered as well for TCP.

We introduce a dissimilarity measurement for test cases represented as MSCs in terms of their shared signals. We analyze if a signal occurs in two compared test cases, disregarding multiple occurrences to to avoid that the sole repetition of the same signal has a negative impact on the similarity. Two signals are only identical, if they are exchanged by the same components in both test cases. We refer to the unique signals of a test case tc as Π_{tc}. We measure the appearances of the same signals in the set Π_{tc} of each test case compared to the total number of unique signals used by both. The dissimilarity of two test cases is defined as Jaccard distance on a scale between 0 and 1, where 0 indicates that both test cases use exactly the same signals between the same components. This leads to a dissimilarity measurement function $dissim : \mathcal{TC} \times \mathcal{TC} \rightarrow [0, 1]$ between two test cases, defined as follows:

$$dissim(tc_i, tc_j) = 1 - \frac{|\Pi_{tc_i} \bigcap \Pi_{tc_j}|}{|\Pi_{tc_i} \bigcup \Pi_{tc_j}|}, tc_i, tc_j \in TC_p, i \neq j$$

To compare a potentially large set of test cases, we have to extend the dissimilarity computation. While performing the TCP, more and more test cases will be prioritized. Hence, suitable next test cases have to be compared to all already prioritized ones. By convenience, the function $dissim : \mathcal{TC} \times \mathcal{P}(\mathcal{TC}) \rightarrow [0, 1]$ computes the average value of pairwise dissimilarity between one test case and a set of test cases TC_p for a product variant as follows:

$$dissim(tc_i, TC_p) = \frac{\sum_{n=1}^{|TC|} dissim(tc_i, tc_n)}{|TC_p|}, tc_i \notin TC_p$$

3.4 Prioritization Formulas

We are able to combine the component-based (CB), signal-based (SB) and dissimilarity-based (DB) prioritization, which makes the TCP approach very flexible. In general, the test case with the highest priority is added to the set of ordered test cases and removed from the set of unordered test cases.

As described in previous work, a basic prioritization is based on the weights of components [16]. We refer to this as *component-based regression priority*. Compared to the previously introduced component weights, we are now able to use the behavioral weight of a component $w_b(c_j)$ as explained in Sect. 3.1. This leads to the prioritization function $prio_{sig} : \mathcal{TC} \times P_{SPL} \rightarrow \mathbb{R}$, which uses the component-weights of components covered by a test cases and measures how often they are used by the incorporated test case signals. It is defined as follows:

$$prio_{comp}(tc_i) = \frac{\sum_{j=1}^{n} w_b(c_j)}{n}, n = |C_{tc_i}|$$

In contrast to the component-based prioritization, we introduce a more sophisticated prioritization technique based on both, component and signal weights. We argue, that the

signals influence the weight of a test case as well, as a test case might comprise a lot of components, but only few signals are exchanged between important components. In addition, component weights do not consider any form of indirect influences, which is why signal weights have been introduced. We introduce the signal-based prioritization as function $prio_{sig} : \mathcal{TC} \times P_{SPL} \rightarrow \mathbb{R}$, defined as:

$$prio_{sig}(tc_i, p_j) = \frac{\sum_{m=1}^{n}\sum_{k=1}^{n} s(c_m, c_k)(w(c_m) + w(c_k)) + w_s(s(c_m, c_k))}{\sum_{m=1}^{n}\sum_{k=1}^{n} s(c_m, c_k)},$$

where *n* is the number of signals of a test case.

The prioritization functions are combinable with the introduced dissimilarity-measure (cf. Sec. 3.3). We are able to compute a final delta priority value for a test case tc_i for a certain variant p_j and the set of already ordered test cases $TC^{p_j}_{ordered}$ based on the component and dissimilarity weight.

The final priority based on both, regression test priority and dissimilarity-based priority, is described by the function $priority : \mathcal{TC} \times P_{SPL} \times \mathcal{P}(\mathcal{TC})$. We define the function as:

$$priority(tc_i, p_j, TC_{ordered}) = dissim(tc_i, TC_{ordered}) + prio_{sig}(tc_i, p_j)$$

4. Evaluation

To show the effectiveness of our fine-grained TCP technique, we formulate two research questions. We evaluate our approach based on an automotive case study. We prototyped and measured the results of our technique compared to the previous prioritization approach and random testing.

Research Questions. To evaluate the contributions of this paper, we formulate the following research questions:

RQ1: How do *behavioral changes impact* the TCP in terms of change coverage?

RQ2: Regarding the *dissimilarity* of test cases, a) how does a *combination* with delta-oriented techniques influence the change coverage and b), how does a dissimilarity-based technique perform in *isolation*?

Subject System. To evaluate our TCP technique, we use the Body Comfort System (BCS) case study [17]. The BCS describes an automotive SPL, including delta-oriented architectures, delta-oriented state machines and 92 test cases in form of MSCs. BCS describes a body comfort system of car, comprising $11,616$ product variants. To reduce the testing complexity, this number has been reduced in previous work to a total of 17 product variants, using the *MoSo-PoLiTe* sampling testing technique [23]. In addition, a core product (called *P0*) has been defined as basis for delta modeling of the SPL [17]. Hence, we perform our evaluation on the solution space artifacts for these 18 product variants. We do not focus on a certain order of the variants, but begin with the core and incrementally test product variants as they occur in the order generated by the sampling technique.

Implementation. We prototyped our technique as plug-ins for Eclipse using EMF and XText. This allows for an automated derivation of product variants based on feature configurations. Our tool categorizes test cases into new, invalid, reusable and retest for each variant, based on their architectures.It automatically computes component and signal weights and prioritizes test cases based on these values.

Methodology. We performed the evaluation using the 18 different product variants described in previous work [17]. 92 test cases are prioritized for product variants $P1$ to $P17$, whereas $P0$ is left out as it is the core variant. To assess the prioritization quality, we use the *average percentage of changes covered* (APCC) metric introduced in previous work [16]. It is based on the *average percentage of faults detected* (APFD) metric [30], which measures the failure detection rate of a TCP. In contrast, APCC is applicable when no failure information is available as it measures the coverage of component interfaces for components which have been changed compared to all previous product variants, i.e., if a component has a priority > 0 its complete interface has to be retested.

APCC is defined for *n* test cases and *m* changed interface connectors, with the $i-th$ connector being covered by the test case at position T_{change_i} as follows [16]:

$$1 - \frac{\sum_{i=0}^{m} T_{change_i}}{nm} + \frac{1}{2n}$$

We compare different combinations of our approach, i.e., component-based vs. signal-based prioritization, with dissimilarity testing and without as well as the previously introduced structural component and signal-based approaches. In addition, we compare our techniques to an unordered and randomized approach. The unordered approach takes test cases according to their name, the random approach shuffles the (reusable) test cases arbitrarily. In particular, we compute 100 random orderings and normalized the results.

Results. We computed a TCP for each product variant and computed the respective APCC. The results are shown as bar chart in Fig. 5. Each bar represents results of one technique in a certain product. The diagram is missing two product variants, $P0$ and $P17$. This is due to the fact, that we do not prioritize test cases for the core, as everything has to be tested. For $P17$, we would prioritize test cases, but our technique did not detect any test cases with priority value greater than 0, as all deltas have been applied previously. In addition, we also provide the average APCC values for each technique over all variants in Tab. 1.

RQ1: As Fig. 5 shows, the addition of the behavioral component weights has only a slight impact. Compared to the previously introduced component-based technique (*Old CB* with $\alpha = 0.5, \beta = 0.25, \gamma = 0.25$), we only see an improvement of the average APCC of 0.03. We tried different weightings in terms of influences on the overall priority computation, but the differences in APCC results

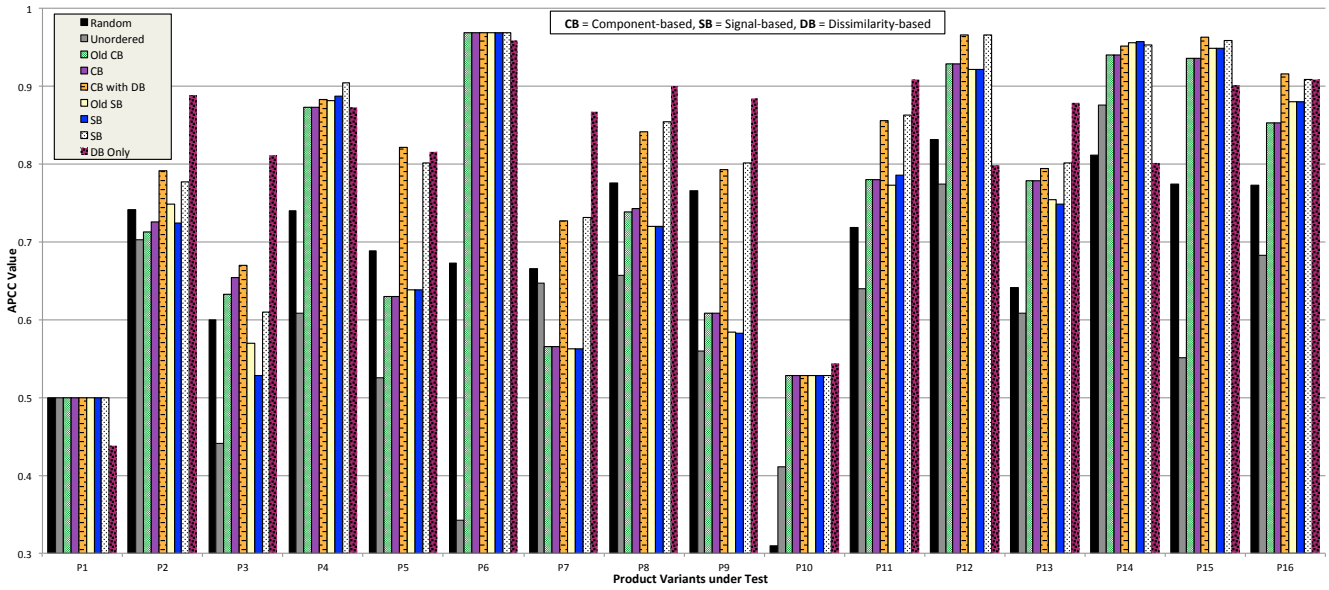

Figure 5. APCC Results of the Different Techniques for all Product Variants

Table 1. Overview of average APCC results

Technique	*Random*	*Unordered*	*Old CB*	*CB*	*CB with DB.*	*Old SB*	*SB*	*SB with DB.*	*DB Only*
Average APCC	0.688	0.595	0.748	0.751	0.81	0.746	0.743	0.808	0.823

are negligible. Hence, we only show the combinations of $\zeta = 0.2$ for both, component and signal-based prioritization. In both cases, the configuration of the structural TCP is similar to previous work, where we argue that input changes should be more important than output changes and, thus, we use a configuration of $\alpha = 0.4, \beta = 0.2$ and $\gamma = 0.2$.

A similar observation is made when looking at the addition of *signal weights* to the signal-based prioritization (*SB*). The figure shows that the original value is similar to the new value. We measure a decrease of APCC by 0.03. As this is a very small change, the addition of new signal weights has no measurable influence on the quality for BCS.

Summarizing, the behavioral weight does influence the results only slightly. The reason for this lies within the artifacts given for BCS, which do not favor a detailed behavioral analysis. For a more complex case study, we argue that the fine-grained analysis still will reveal certain situations of interest, i.e., it will outperform the structural analysis when interfaces are unchanged, but only internal changes occur.

RQ2: Besides measuring the structural and behavioral prioritization, we examined the influences of the dissimilarity-based approach (cf. Sec. 3.3) a) in combination with the delta-oriented prioritization and b) in isolation. For the combination, we used the same influence factors for the delta-oriented prioritization as described in RQ1 and combined them with the dissimilarity based technique. We employed an equal weighting of both, priority given by delta-oriented prioritization and dissimilarity-based prioritization. Compared to behavioral weights, the addition of dissimilarity-based prioritization changes the results of both, component-based and signal-based prioritization, to the better. In fact, it does increase the average APCC results by about 0.05 for both techniques, as shown in Tab. 1. The results in Fig. 5 show that the dissimilarity approach increases the differences compared to the original techniques. The overall APCC value of 0.8 for both combinations shows the potential of this technique. As for the reasons behind this increase, the dissimilarity approach tries to force a fast coverage of the system. When combined with delta-oriented techniques, it operates on the test cases classified as to be retested, i.e., with a priority value > 0. Hence, the focus lies on the most important parts while increasing the coverage compared to the original technique. We state that a combination of delta-oriented testing and dissimilarity based testing achieves better results than our previous technique.

We also measured the results of a solely dissimilarity-based prioritization. One important factor is that we did employ the dissimilarity-based approach on all *reusable* test cases for a product variant, i.e., we did not select the *retest* test cases as a dissimilarity-based approach has no information about changes between products. The APCC results for each product variant are shown in Fig. 5, denoted by *DB Only*. It becomes evident, that the results are mixed compared to previous techniques. In general, the results are slightly better (cf. Tab. 1) with an average APCC for BCS of 0.82. While this is a very good result, the dissimilarity-

based approach took all reusable test cases into account, i.e., the test set is larger than for our technique, where we preselect test cases of importance due to changes. This discrepancy becomes evident for certain product variants, e.g., $P1$. Here, our techniques only detect one test case as important, leading to an APCC of 0.5. The dissimilarity-based approach does not have this information and achieves a worse APCC for $P1$ as it examines a larger test set. In case many test cases are to be retested compared to all reusable test cases, the dissimilarity-based technique achieves very good results.

Summarizing, the fine-grained TCP technique is able to execute less test cases while achieving a similar APCC as the dissimilarity-based TCP. In certain situations a combined approach outperforms the dissimilarity-based approach significantly (e.g., in $P12$ or $P14$). Overall, we are able to improve test effectiveness compared to a random prioritization.

Threats to Validity. The set of test cases defined for the BCS SPL is a potential threat as they comprise partially redundant interaction scenarios. The rather small number of test cases reduces the amount of different orders. However, the corresponding problem of designing test cases exits in general for model-based testing techniques [5, 32]. We designed our test cases such that we reduced unnecessary redundancy and further avoided to specify super test cases, i.e., test cases which contain more than three components and a lot of interactions between them. We used the APCC metric to compare our novel technique with our prior work [16]. For a more realistic scenario, fault-based metrics are desired. However, we make the valid assumption that even small changes lead to faults [9] and, thus, a high APCC is desirable. APCC measures how fast changes are covered. That means, while our technique is feasible to achieve good APCC values it is not necessarily the best technique as test case clusters might reduce change covering speed. This rejects the assumption that the metric fits the technique by design. While the pure dissimilarity-based approach achieves a higher APCC than the delta-oriented or combined techniques, it does not take changes into account, thus, all reusable test cases are prioritized, whereas our approach first select test cases and then prioritize them. Finally, future work comprises further evaluations with realistic SPLs to consolidate and generalize our positive results.

5. Related Work

SPL Regression testing techniques are mainly applied in the industrial context [8, 10, 31], for SPL architectures [7, 22], for sample-based testing [27, 28], and to facilitate incremental SPL testing [3, 4, 18–20, 33, 34]. Uzuncaova et al. [33] define an incremental refinement of variant-specific test suites when stepping to the next variant. Baller et al. [3, 4] propose amulti-objective test suite optimization for efficient SPL testing by taking profit constraints for test artifacts into account. Varshosaz et al. [34] define delta-oriented test case generation by exploiting the incremental structure of delta-oriented test models. Lity et al. [18] present a technique for retest test case selection based on the application of incremental slicing for change impact analysis when stepping to subsequent variants under test. In contrast to our work, where test cases are prioritized for retest, the related techniques consider the creation and optimization of test suites or the selection of test cases to be retested for an SPL.

In the context of SPL integration testing, Muccini and van der Hoek [22] discuss challenges and opportunities for variability-aware integration testing based on the comparison to existing single-system testing techniques. Neto et al. [7] present a framework for regression testing of SPL architectures, where retest decision are made based on the similarity of architecture variants and code. Reis et al. [29] propose a technique for integration test case generation and reuse on the basis of variability-aware UML activity diagrams specifying interaction scenarios to be tested. Those techniques introduce test case selection, regression frameworks or test case generation, but do not perform TCP.

Techniques for SPL TCP are prevalent in the context of feature configurations [1, 12, 21]. Ensan et al. [12] describe a feature-based prioritization, where features and important goals specified by stakeholders are combined to select and prioritize configurations. Lopez-Herrejon et al. [21] propose a technique for prioritizing pairwise feature configurations by applying evolutionary algorithms. Al-Hajjaji et al. [1] present a similarity-based product prioritization, where the minimal similarity of feature configurations between tested, and untested variants are taken into account to select the next variant to be tested. These prioritization approaches are applied on feature configuration level, whereas we prioritize test cases represented as MSCs for each variant under test.

6. Conclusion

In this paper, we proposed two extensions for TCP for incremental SPL integration testing to enhance the testing effectiveness. We defined a fine-grained impact analysis of component interfaces by incorporating changes on the input/output behavior specified in component state machines in addition to the existing architecture analysis. We also presented a test case dissimilarity measure between message sequence charts. We evaluated our fine-grained TCP by means of a case study from the automotive domain showing a gain in testing effectiveness.

For future work, we plan to investigate the fault detection capabilities of our approach using different case studies using the well-known APFD metric [30]. This will allow to generalize our findings. We are investigating how to integrate risk-based testing into SPL test case prioritization based on architecture changes and their impact. We will design a TCP framework to make the approach easily adaptable and extensible. In the long run, the presented weight and prioritization functions could be used as features for a machine learning or search-based TCP techniques.

Acknowledgments

This work was partially funded by the German Research Foundation under Priority Program SPP 1593: Design For Future – Managed Software Evolution.

References

[1] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake. Similarity-based prioritization in software product-line testing. In *SPLC*, pages 197–206, 2014.

[2] K. Androutsopoulos, D. Clark, M. Harman, J. Krinke, and L. Tratt. State-based Model Slicing: A Survey. *CSUR*, 45 (4):53:1–53:36, 2013.

[3] H. Baller and M. Lochau. Towards Incremental Test Suite Optimization for Software Product Lines. In *FOSD*, pages 30–36. ACM, 2014.

[4] H. Baller, S. Lity, M. Lochau, and I. Schaefer. Multi-objective test suite optimization for incremental product family testing. In *ICST*, pages 303–312, 2014.

[5] A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti. An approach to integration testing based on architectural descriptions. In *ICECCS*, pages 77–84, 1997.

[6] D. Clarke, M. Helvensteijn, and I. Schaefer. Abstract delta modeling. In *GPCE*, pages 13–22, 2010.

[7] P. Da Mota Silveira Neto, I. do Carmo Machado, Y. Cavalcanti, E. de Almeida, V. Garcia, and S. de Lemos Meira. A Regression Testing Approach for Software Product Lines Architectures. In *SBCARS*, pages 41–50, 2010.

[8] M. Dukaczewski, I. Schaefer, R. Lachmann, and M. Lochau. Requirements-based delta-oriented spl testing. In *PLEASE*, pages 49–52, 2013.

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

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

[11] E. Engström and P. Runeson. Software product line testing - a systematic mapping study. *JIST*, 53:2–13, 2011.

[12] A. Ensan, E. Bagheri, M. Asadi, D. Gasevic, and Y. Biletskiy. Goal-oriented test case selection and prioritization for product line feature models. In *ITNG*, pages 291–298, 2011.

[13] D. Harel and P. S. Thiagarajan. Message sequence charts. In *In UML for Real: Design of Embedded Real-Time Systems*, pages 77–105, 2003.

[14] H. Hemmati, A. Arcuri, and L. Briand. Achieving scalable model-based testing through test case diversity. *TOSEM*, 22 (1):1–42, 2013.

[15] M. F. Johansen, Ø. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *SPLC*, pages 46–55, 2012.

[16] 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 *SPLC*, pages 81–90, 2015.

[17] 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.

[18] S. Lity, T. Morbach, T. Thüm, and I. Schaefer. Applying Incremental Model Slicing to Product-Line Regression Testing. In *ICSR*, 2016.

[19] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity. Incremental Model-based Testing of Delta-oriented Software Product Lines. In *TAP*, pages 67–82, 2012.

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

[21] R. E. Lopez-Herrejon, J. Javier Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In *GECCO*, pages 1255–1262, 2014.

[22] H. Muccini and A. van der Hoek. Towards Testing Product Line Architectures. *ENTCS*, 82(6):99 – 109, 2003.

[23] S. Oster, M. Zink, M. Lochau, and M. Grechanik. Pairwise feature-interaction testing for spls: Potentials and limitations. In *SPLC*, pages 6:1–6:8, 2011.

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

[25] C. Pietsch, T. Kehrer, U. Kelter, D. Reuling, and M. Ohrndorf. Sipl a delta-based modeling framework for software product line engineering. In *ASE*, pages 852–857, 2015.

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

[27] X. Qu, M. Cohen, and K. Woolf. Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization. In *ICSM*, pages 255–264, 2007.

[28] X. Qu, M. B. Cohen, and G. Rothermel. Configuration-aware Regression Testing: An Empirical Study of Sampling and Prioritization. In *ISSTA*, pages 75–86, 2008.

[29] S. Reis, A. Metzger, and K. Pohl. Integration Testing in Software product Line Engineering: A Model-Based Technique. In *FASE*, pages 321–335. Springer-Verlag, 2007.

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

[31] P. Runeson and E. Engström. Software product line testing - a 3d regression testing problem. In *ICST*, pages 742–746, 2012.

[32] M. Utting and B. Legeard. *Practical Model-based Testing*. Morgan Kaufmann, 2007.

[33] E. Uzuncaova, S. Khurshid, and D. Batory. Incremental Test Generation for Software Product Lines. *TSE*, 36(3):309–322, 2010.

[34] M. Varshosaz, H. Beohar, and M. R. Mousavi. Delta-Oriented FSM-Based Testing. In *ICFEM*, volume 9407 of *LNCS*, pages 366–381. Springer, 2015.

[35] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *JSTVR*, 22(2):67–120, 2007.