Applying Incremental Model Slicing to Product-Line Regression Testing

Sascha Lity[1(✉)], Thomas Morbach[1], Thomas Thüm[2], and Ina Schaefer[2]

[1] Institute for Programming and Reactive Systems, TU Braunschweig, Braunschweig, Germany
{s.lity,t.morbach}@tu-braunschweig.de

[2] Institute of Software Engineering and Automotive Informatics, TU Braunschweig, Braunschweig, Germany
{t.thuem,i.schaefer}@tu-braunschweig.de

Abstract. One crucial activity in software product line (SPL) testing is the detection of erroneous artifact interactions when combined for a variant. This detection is similar to the retest test-case selection problem in regression testing, where change impact analysis is applied to reason about changed dependencies to be retested. In this paper, we propose automated change impact analysis based on incremental model slicing for incremental SPL testing. Incremental slicing allows for a slice computation by adapting a previous slice with explicit derivation of their differences by taking model changes into account. We apply incremental slicing to determine the impact of applied model changes and to reason about their potential retest. Based on our novel retest coverage criterion, each slice change specifies a retest test goal to be covered by existing test cases selected for retesting. We prototypically implemented our approach and evaluated its applicability and effectiveness by means of four SPLs.

Keywords: Software product line · Model-based testing · Regression testing · Model slicing

1 Introduction

Testing is a necessary and challenging activity during software development [15]. Testing a *software product line* (SPL) [31], i.e., a family of similar software systems sharing a common platform and reusable artifacts, is even more challenging. Due to the huge number of potential variants, testing each variant individually is often infeasible. Besides the standard testing problem finding failures in (variable) software artifacts, SPL testing also focuses on the detection of erroneous interactions of variable artifacts [27]. Existing SPL testing techniques [13,23,28,30] reduce the overall testing effort, e.g., by testing only a subset of variants [19]. However, each variant is still tested individually without

This work was partially supported by the German Research Foundation under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

taking the commonality and obtained test results into account as common in regression testing [42].

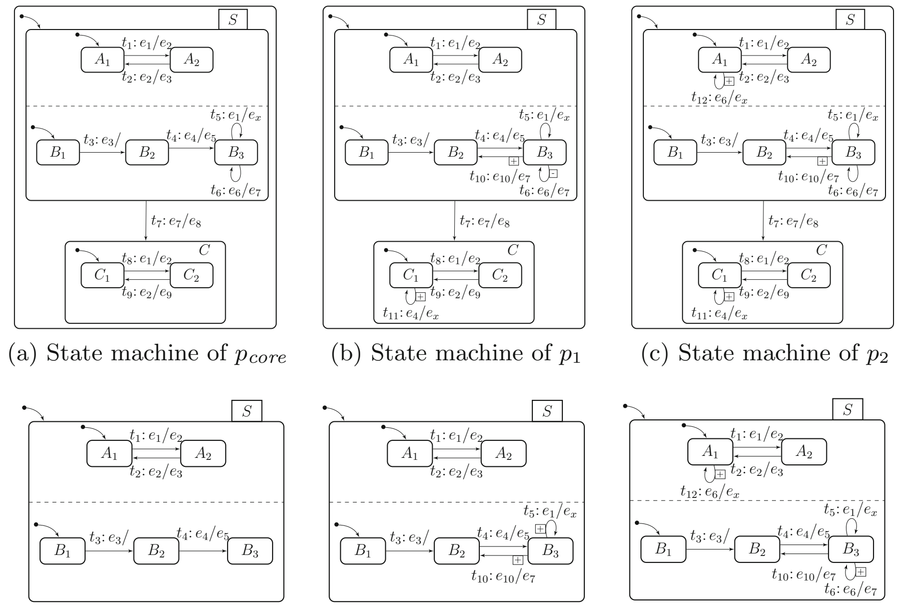
In prior work [25,26], we proposed incremental SPL testing combining the concepts of model-based [37] and regression testing [42] including the reuse of test artifacts and test results. Based on delta-oriented modeling [9] specifying the commonality and variability of variants by means of change operations called deltas, variant-specific test artifacts, such as the test suite, are incrementally adapted. Starting with a core variant, each subsequent variant is tested based on its tested predecessor. Here, a crucial task is the decision whether a reusable test case should be retested to validate that already tested behavior is not unintentionally affected by changes. In our prior work, this retest decision is to be performed manually. To cope with this well-known *retest test-case selection problem* [42] in regression testing, change impact analysis techniques are required.

Slicing [4,41] is a promising analysis technique to automate this retest decision [8]. Existing approaches apply slicing in a white-box regression testing setup [1,7,14]. In line with our prior work [25,26], we focus on model-based regression testing, where we have no access to source code. Therefore, state-based model slicing techniques [4] allow for model-based change impact analysis and, hence, to reason about the retest of test cases [21,36]. Based on a given model element used as slicing criterion, a behavioral specification, such as a finite state machine, is reduced comprising solely elements affecting the criterion. Thus, a modified slice indicates changed dependencies regarding the criterion. In prior work [24], we proposed incremental model slicing for delta-oriented SPLs. When stepping from a variant to its subsequent one, we reuse the applied model changes specified by deltas to incrementally adapt already existing slices for same slicing criteria. As further result, we capture the differences between two subsequent slices as slice changes useable to reason about the impact of model changes.

In this paper, we adapt and apply our incremental slicing as change impact analysis technique *to automate retest decisions in incremental SPL testing*. Hence, we investigate the impact of model changes to common behavior between subsequent variants. We capture the obtained slice differences representing new/changed dependencies in respective slice changes, e.g., potential artifact interactions to be retested. The slice changes are used *to define a new coverage criterion facilitating the retest decision*. Therefore, we select reusable test cases for re-execution and further use test-case generation for generating new retest test cases to ensure coverage. We prototypically *implemented and evaluated our approach by means of four SPLs* to validate its applicability and effectiveness.

2 Foundations

Delta-Oriented Software Product Lines. Delta modeling [9] is a modular and flexible variability modeling approach already applied to various types of SPL development artifacts, e.g., finite state machines [26]. Differences between variants are explicitly specified by means of transformations called *deltas*. Based on a given *core model* M_{core} modeling a *core variant* $p_{core} \in P_{SPL}$, deltas $\delta \in \Delta_{\mathcal{M}}^{SPL}$ are



(a) State machine of p_{core} (b) State machine of p_1 (c) State machine of p_2

(d) Core slice for t_4 (e) Slice for t_4 of p_1 (f) Slice for t_4 of p_2

Fig. 1. Example of delta modeling [9] and incremental model slicing [24]

defined transforming M_{core} into the model M_i of variant $p_i \in P_{SPL}$. By P_{SPL}, we refer to the set of all valid variants of an SPL, whereas $\Delta^{SPL}_{\mathcal{M}}$ denotes the set of all deltas for P_{SPL} and model domain $\mathcal{M}$, here, *finite state machines* (FSM). A delta δ specifies change operations, such as additions (*add e*) or removals (*rem e*) of elements. For each $p_i \in P_{SPL}$, a predefined set of deltas $\Delta_{p_i} \subseteq \Delta^{SPL}_{\mathcal{M}}$ exists subsequently applied in a predefined order to automatically generate M_i. For the specification of changes between arbitrary variants, we are able to derive *model regression deltas* $\Delta^{\mathcal{M}}_{p_{i-1},p_i} \subseteq \Delta^{SPL}_{\mathcal{M}}$ to encapsulate the differences between two variants by incorporating their delta sets $\Delta_{p_{i-1}}$ and Δ_{p_i}. We refer the reader to prior work [25,26] for the construction of model regression deltas.

Example 1. Consider the sample core state machine shown in Fig. 1a. Based on the delta set $\Delta_{p_1} = \{\delta_1 = (\text{add}\, t_{10}), \delta_2 = (\text{add}\, t_{11}), \delta_3 = (\text{rem}\, t_6)\}$ denoted by either $+$ (add) or $-$ (rem), the core is transformed into the state machine of p_1 depicted in Fig. 1b. For p_2, we apply the delta set $\Delta_{p_2} = \{\delta_1, \delta_2, \delta_4 = (\text{add}\, t_{12})\}$. To step from p_1 to p_2, we derive the model regression delta $\Delta^{FSM}_{p_1,p_2} = \{(\text{add}\, t_{12}), (\text{add}\, t_6)\}$ by incorporating the deltas captured in Δ_{p_1} and Δ_{p_2}.

Incremental Model-Based SPL Testing. Model-based testing [37] defines processes for automatic test-case generation based on test models. A *test model tm*, e.g., a finite state machine, specifies the behavior of a *system under test* in terms of *controllable inputs* and *observable outputs*. A system behaves as expected if it reacts to inputs with expected outputs. To validate the conformance, test cases are derivable from the test model. A *test case tc* corresponds to a test model path defining a sequence of controlled inputs and expected outputs. For guiding the test-case generation, *adequacy criteria*, e.g., structural coverage criteria like *all-transitions* [37], where each transition of the test model must be traversed by a test case, are used to define a set of *test goals* $TG = \{tg_1, \ldots, tg_n\}$. For each test goal, at least one test case must be generated to be collected in a *test suite* $TS = \{tc_1, \ldots, tc_m\}$. For SPLs, these *test artifacts* exist for each $p_i \in P_{SPL}$ captured in variant-specific test artifact sets $TA_{p_i} = \{tm_{p_i}, TG_{p_i}, TS_{p_i}\}$.

In prior work [25,26], we proposed incremental SPL testing, where variants are subsequently tested. To exploit the reuse potential of test artifacts during testing, we adopt the concept of delta modeling to define regression deltas for the set of test artifacts. Hence, we are able to adapt the variant-specific test artifacts when stepping from p_{i-1} to p_i under test. Furthermore, we categorize the test suite TS_{p_i} similar to regression testing [42] in sets of new $TS_N^{p_i}$, reusable $TS_{Re}^{p_i}$, and obsolete test cases $TS_O^{p_i}$. *New test cases* are generated for yet uncovered test goals of p_i. *Reusable test cases* were generated in prior testing steps and are also valid for p_i. *Obsolete test cases* are not valid and removed from TS_{p_i} of p_i. Obsolete test cases are potentially reusable for a subsequent p_j under test and, thus, remain recorded in a shared artifact repository. From $TS_{Re}^{p_i}$, we manually select test cases $TS_R^{p_i} \subseteq TS_{Re}^{p_i}$ for retesting that already tested behavior is not influenced as intended also known as retest test-case selection problem [42].

The testing workflow is defined as follows. We start by testing p_{core}, i.e., we apply standard model-based testing [37]. The obtained test suite and test results are stored in a shared test artifact repository reused and updated by subsequent testing steps. By stepping from p_{i-1} to p_i, we apply the following:

1. Compute test artifact regression deltas based on delta sets $\Delta_{p_{i-1}}$ and Δ_{p_i}.
2. Apply regression deltas to adapt the reusable test artifacts.
3. Categorize test cases to obtain test suites $TS_{Re}^{p_i}$ and $TS_O^{p_i}$.
4. Apply test-case generation for uncovered test goals to obtain $TS_N^{p_i}$.
5. *Analyze change impact of applied model changes.*
6. *Select reusable test cases to be retested captured in* $TS_R^{p_i}$.
7. (Re-)Test variant p_i and record results.

Afterwards, we select the next variant p_{i+1} to be tested until no variants remain. In Sect. 3, we present how Steps 5 and 6 are automated by applying change impact analysis techniques, i.e., incremental model slicing.

Example 2. We use the core from Example 1 as test model for p_{core} and focus on all-transition coverage. A test case tc_1 is generated for transition t_6 also covering t_1, t_2, t_3, and t_4. To step to p_1, we adapt the core test artifacts to TA_{p_1} based on Δ_{p_1} also representing the regression delta $\Delta^{FSM}_{p_{core},p_1}$. The test model is transformed as shown in Fig. 1b. $TG_{p_{core}}$ is adapted by removing t_6 and adding t_{10} and t_{11}. The test suite is adapted such that tc_1 is obsolete and removed as well as new test cases for t_{10} and t_{11} are added. To step to p_2, the test artifacts are adapted similarly, whereas tc_1 becomes reusable again and is added to $TS_{Re}^{p_2}$.

Incremental Model Slicing. Slicing was first proposed by Weiser [41] for static analysis of procedural programs. Meanwhile, slicing is also adapted for the analysis of state-based models, e.g., finite state machines. Such *model slicing* [4,18,20] allows for a model reduction by abstracting from model elements not influencing a selected element, e.g., a transition, used as slicing criterion c. A reduced model is called *slice* and preserves the execution semantics compared to the original model w.r.t. the criterion c. We focus on *backward slicing*, where a slice comprises elements influencing the criterion c determined based on (1) *control* and *data dependencies* between elements, and (2) to ensure model well-formedness, e.g., element reachability and model connectedness. Backward slicing investigates which elements have potential influences on a slicing criterion and, thus, indicates retest potentials when changed influences are determined.

In prior work [24], we combined model slicing and delta modeling to analyze delta-oriented SPLs. A slice $Slice^{c}_{p_i}$ for a criterion $c \in C_{p_i}$ of variant p_i is incrementally computed by exploiting the model changes specified in the model regression delta $\Delta^{FSM}_{p_{i-1},p_i}$. By C_{p_i}, we refer to the set of all slicing criteria of p_i. The incremental computation is defined by $incr_{Slice} : C_{SPL} \times \Delta^{SPL}_{FSM} \times SLICE_{C_{SPL}} \rightarrow SLICE_{C_{SPL}}$ where $C_{SPL} = \bigcup_{p_i \in P_{SPL}} C_{p_i}$ and $SLICE_{C_{SPL}}$ denotes the set of all slices of the current SPL. Starting with the prior slice $Slice^{c}_{p_j}$ for criterion c, the regression delta is applied to change the slice completed by standard model slicing. If there is no prior slice, we apply standard model slicing to compute a new slice reusable for subsequent variants. In addition, we directly determine the differences, i.e., *slice changes*, between the prior and current slice captured in a *slice regression delta* $\Delta^{p_j,p_i}_{Slice_c}$. The slice changes indicate the impact of applied model changes to the slicing criterion referring to behavior to be retested and, therefore, facilitate automated retest decisions as proposed in the next section.

Example 3. Consider the core slice for t_4 capturing all its influencing elements shown in Fig. 1d. Based on Δ_{p_1} from Example 1, the core is transformed into model p_1. As those model changes have influences on t_4, we recompute its slice and derive the slice regression delta $\Delta^{p_{core},p_1}_{Slice_{t_4}} = \{(add\, t_5), (add\, t_{10})\}$ shown in Fig. 1e. By stepping to p_2, the model changes also results in slice changes for $Slice^{t_4}_{p_2}$.

3 Automated Retest Decisions Based on Slice Changes

Change Impact Analysis. By stepping from p_{i-1} to p_i under test, the test model regression delta $\Delta^{FSM}_{p_{i-1},p_i}$ captures all changes between the test models. Each change has an impact on already tested behavior common in both variants. To investigate whether these impacts have unintended side effects, e.g., based on unintended artifact interactions, the potentially affected behavior should be retested by re-executing reusable test cases $tc_j \in TS^{p_i}_{Re}$ of p_i. The identification of affected behavior and the selection of reusable test cases is performed manually in our incremental SPL testing strategy [25,26]. For automation, we apply our incremental slicing technique [24] as change impact analysis. Based on obtained slice changes indicating changed dependencies and their side effects to already tested behavior, we automate the retest decision.

For the successful application of incremental model slicing as change impact analysis technique, we require (1) a set of slicing criteria suitable to investigate the change impact, (2) a set of meaningful dependencies, and (3) model regression deltas specifying the test model changes between subsequent variants under test. Similar to test-case generation, where test goals are used to guide the generation process, we use test goals as slicing criteria for (1), as changes may influence their corresponding behavior. We focus on all-transition coverage, i.e., a well-known structural coverage criteria used for test-case generation [37], such that each transition serves as a slicing criterion. As in prior work [24], we focus on the control dependencies proposed by Kamischke et al. [20] for (2). To fulfill (3), the required model regression deltas are provided by the incremental testing strategy.

Furthermore, the order in which variants are tested influence the result [3,16] as the derived model regression deltas and, thus, the result of the change impact analysis may differ. We omit the investigation of the best testing order, i.e., an order of variants allowing for maximal test artifact reuse and minimal retest decisions, and leave it for future work. We assume that a certain testing order is given starting with the core variant. However, to incorporate the testing order for change impact analysis, we identify three potential application scenarios for the derivation of slice changes as follows, where each scenario specifies which prior slice for a given slicing criterion is to be selected for adaptation to reason about change impact. For a slicing criterion contained in a test model for the first time, we start with a new slice.

1. **Least Slice Difference** – By stepping to the next variant, we select one of prior computed slices for the same slicing criterion, i.e., a transition contained in both variants, such that the resulting slice changes are minimized. To select a suitable slice, all prior computed slices must be stored and an additional analysis is required. The determination of a suitable slice, e.g., by analyzing the core and all deltas defined for already tested variants, is left open for future work. Due to a potential change minimization, a reduction of retest decisions and a reduction of test cases to be retested is achievable.
2. **Last Variant** – For a slicing criterion, we check whether a slice to be adapted exists in the prior variant p_{i-1}. If a prior slice exists, we obtain slice changes for the slicing criterion. Otherwise, a new slice has to be computed and no retest decision can be made. In such a case, either all reusable test cases covering the slicing criterion should be retested or none of the test cases are re-executed leading to possible missed failures. A new slice may be used as basis for incremental slicing in the subsequent variant p_{i+1} under test.
3. **Previous Slice** – Similar to the prior scenario, we check for a slicing criterion if a previous slice to be adapted exists. In contrast, we select the last computed slice for the corresponding criterion of an already tested variant p_j under test, where the respective test model element was contained. Thus, we ensure the

determination of slice changes if existing, resulting in retest decisions to be made for the current variant p_i under test.

We do not choose the *Least Slice Difference* scenario as it requires an additional analysis effort, i.e., a crucial factor for change impact analysis, and the slice selection to achieve minimized slice changes is an open question left for future work. We further do not choose the *Last Variant* scenario as we are interested in retest decisions to be made in every variant. The scenario does not ensure retest decisions as for some slicing criteria the last variant under test does not contain a required slice to be adaptable. In this paper, we focus on the *Previous Slice* scenario following the incremental idea by providing retest decisions based on slice changes for each variant under test. Depending on a given testing order, the scenario, however, may result in redundant decisions for some slicing criteria in subsequent variants under test. To select already computed slices, we store each new/adapted slice in a shared slice repository.

Based on the *Previous Slice* scenario, we apply incremental slicing for change impact analysis as follows. The incremental testing strategy starts with the test of p_{core}. As p_{core} is the first variant under test, no retest potentials arise and we solely compute for all test goals used as slicing criteria the first slices such that $\forall tg \in TG_{p_{core}} : incr_{Slice}(tg, \emptyset, \epsilon) = Slice^{tg}_{p_{core}}$ holds. By the second parameter $\emptyset$, we refer to the empty model regression delta not existing for the core, whereas the third parameter ϵ represents that no prior slices exist for the slicing criterion tg. These slices build the basis of the change impact analysis for the next variants. For the remaining variants to be tested, we apply our analysis after the test-case generation step for covering not yet covered test goals such that

$$\forall tg \in TG_{p_i} : \begin{cases} incr_{Slice}(tg, \emptyset, \epsilon) = Slice^{tg}_{p_i} & \text{if } \neg\exists Slice^{tg}_{p_j}, j < i \quad (1) \\ incr_{Slice}(tg, \Delta^{FSM}_{p_{i-1},p_i}, Slice^{tg}_{p_j}) = Slice^{tg}_{p_i} & \text{if } \exists Slice^{tg}_{p_j}, j < i \quad (2) \end{cases}$$

holds. For each test goal of a variant p_i under test, one of the two cases is valid. In case (1), the test goal and, thus, the respective model element is contained in a test model for the first time. That is, no prior slice exists in the shared slice repository to be adaptable for analyzing the change impact. For the test goal, a new slice is computed and stored for subsequent testing steps. In case (2), we have access to a prior slice of the test goal to be adaptable based on incremental model slicing. In addition to the adapted slice, we obtain the slice regression delta $\Delta^{p_j,p_i}_{Slice_{tg}}$ as result facilitating retest decision to be made.

A slice regression delta captures additions/removals of test model elements of the updated slice. Both, additions and removals represent changed behavioral influences on the respective slicing criterion, i.e., test goal, caused by the applied model changes. Those changed influences may indicate potential sources of errors to be (re-)tested, e.g., due to unintended artifact interactions. In contrast, an empty slice regression delta implies that no changed influences exist and, thus, no retest potentials arise for the corresponding test goal.

To reason about the retest of test cases, we require a scale by means of an expressive criterion based on these slice changes. Coverage criteria are promising scales as they are already applied in various scenarios, e.g., all-transition coverage for test-case generation [37]. Hence, we define a new coverage criterion to control the retest decision as described in the next paragraph.

Example 4. Consider Examples 2 and 3 again. For p_{core}, we compute for each test goal a respective slice, e.g., the slice for t_4 depicted in Fig. 1d. By stepping to p_1, all test artifacts are adapted and for each test goal, a slice is either created from scratch or updated by applying the incremental slicing, in which we use the slices of the core as basis. For instance, compared to the base slice of t_4, its updated slice comprises two new elements as shown in Fig. 1e indicating new dependencies for t_4 to be tested. By stepping to p_2, the slice of t_4 is again updated. Furthermore, the slice of t_6 computed for p_{core} is now updated to investigate the applied model changes as the element was not contained in p_1.

Retest Coverage Criterion. Similar to the definition of model-based coverage criteria, where test model elements, e.g., transitions are used as test goals [37], we combine slicing criteria with their slice changes to define a novel *retest coverage criterion.* A *retest test goal* $rtg_l = (e_k, tg) \in TG_R^{p_i}$ is defined as a pair of test model elements such that tg represents a slicing criterion of p_i for which a slice regression delta $\Delta_{Slice_{tg}}^{p_{i-1},p_i}$ exists, and e_k corresponds to one of the following cases:

1. A state/transition added to the slice $Slice_{p_i}^{tg}$ via $\Delta_{Slice_{tg}}^{p_{i-1},p_i}$.
2. A source/target state of a transition removed with $\Delta_{Slice_{tg}}^{p_{i-1},p_i}$ still contained in the current test model.

Retest test goals are defined for Case 1 to (re-)validate newly introduced dependencies between element e_k and slicing criterion tg. For Case 2, retest test goals are specified to validate that removed behavior represented by a removed transition is not remained in the variant implementation under test, and that new dependencies build due to the removal are implemented as expected. With $TG_R^{p_i}$, we refer to the retest test goal set of the variant p_i under test.

To cover a retest test goal $rtg_l \in TG_R^{p_i}$ by a test case tc, both, e_k and tg must be traversed by the test model path of tc. Test case tc specifies a representative execution of the current variant validating that no unexpected behavior is implemented based on the new dependencies between both elements. Similar to model-based coverage criteria, for each retest test goal $rtg_l \in TG_R^{p_i}$, at least one test case must exist for its coverage. An empty retest test goal set indicates that no retest decisions are to be made.

Example 5. We use the slice changes for t_4 obtained in Example 4 by stepping from p_{core} to p_1 as basis to define respective retest test goals. The set $TG_R^{p_1}$ comprises two retest test goals $rtg_1 = (t_5, t_4)$ and $rtg_2 = (t_{10}, t_4)$ to be covered by (reusable) test cases. In p_2, we similarly derive for t_4 the retest test goal $rtg_3 = (t_6, t_4)$.

Retest Test-Case Selection and Generation. To validate that changes do not unintentionally influence already tested behavior when stepping to a subsequent variant under test, we require the selection of reusable test cases to be retested. Based on a set of retest test goals $TG_R^{p_i}$ for a variant p_i, we are able to make retest decisions. Therefore, each retest test goal $rtg_l \in TG_R^{p_i}$ must be covered by at least one (reusable) test case. We identify three coverage scenarios as follows:

1. *New Test Cases* – By adapting the test model based on a model regression delta, new model elements e_k may be added for the first time, i.e., all prior tested variants do not comprise e_k. Those elements will also be contained in slices for the first time resulting in retest goals $rtg_l = (e_k, tg)$ not coverable by reusable test cases. For this type of retest test goals, we check whether newly added test cases $tc \in TS_N^{p_i}$ are selectable to cover retest test goals.
2. *Reusable Test Cases* – By stepping to a subsequent variant, some elements e_k may again be added to the current model via the model regression delta, i.e., e_k was already contained in a prior test model. Those elements will again be contained in slices resulting in retest goals $rtg_l = (e_k, tg)$ covered by reusable test cases to be selected for retest. For this type of retest test goals, we check whether reusable test cases $tc \in TS_{Re}^{p_i}$ are selectable for retest.
3. *New Retest Test Cases* – Some retest test goals may not be covered by the current test suite. All test cases are defined by a certain test model path which must traverse the element combination specified by a retest test goal for its coverage. We apply test-case generation to derive new *retest test cases* covering the remaining retest test goals. The newly generated retest test cases are added to the variant-specific test suite also reusable for subsequent variants under test. For this type of retest test goals, we generate new test cases $tc \in TS_{R,N}^{p_i}$ for covering the remaining uncovered retest test goals.

The selected reusable test cases $TS_R^{p_i}$ and the generated retest test cases $TS_{R,N}^{p_i}$ are captured in a *retest suite* $TS_R^{p_i} = TS_R^{p_i} \cup TS_{R,N}^{p_i}$ for variant p_i under test.

The resulting test suite $TS_{p_i} = TS_N^{p_i} \cup TS_R^{p_i}$ for p_i comprising all new and retest test cases is executed to (re)test p_i. The test suite may contain redundant test cases, e.g., by means of test goal coverage, and, thus, be still optimizable to further reduce the testing effort. To minimize the set of executed test cases, test suite minimization [42] is applicable, which is, however, out of our scope. After test case execution, all new and updated test artifacts, e.g., slices and test cases, are stored in the shared test artifact and slice repository. The incremental testing process is either finished or selects the next variant to be tested.

Example 6. To cover the retest test goals from Example 5, we select a reusable test case, i.e., generated for p_{core}, covering t_5 and the retest test goal rtg_1. In contrast, as t_{10} is a new transition not contained in a prior variant, we generate a new test case for t_{10} to cover rtg_2. For covering rtg_3 in p_2, we again select a reusable test case, i.e., the test case for t_6 defined in Example 2. To cover the new retest test goal $rtg_4 = (t_{12}, t_4)$ in p_2, we generate a new retest test case.

4 Evaluation

Prototype. Our approach is realized as Eclipse[1] plug-ins to facilitate its extendability in future work. We use the CAISE-tool eMoflon[2] for the model-driven development based on respective meta models defined in the Eclipse modeling framework.[3] We applied FeatureIDE[4] for feature modeling and to derive the set of valid feature configurations generated in default order, i.e., by incrementing the number of features contained in a configuration after covering all potential combinations of the current number of features. Furthermore, our approach requires the generation of test cases to cover (retest) test goals during testing all variants. For test-case generation, we apply CPA/Tiger[5] an extension of the symbolic model-checker CPA/Checker[6] for the language C. Therefore, we realized a transformation from our event-based models to C-Code by encoding the event handling based on respective variables and their restricted access.

Subject Product Lines. For evaluation, we apply our approach to our example and to three existing SPLs already served as benchmarks in the literature [10]. These systems are (1) a *Wiper* SPL comprising variable qualities of rain sensors and wipers, (2) a *Vending Machine* SPL with optional selection of various beverages, and (3) a *Mine Pump* SPL describing a pump control system with variable water level handling and an optional methane detection facility. We chose these systems as each has unique characteristics, e.g., the mine pump SPL is specified by parallel regions mainly synchronized by internal variables and events, whereas the wiper subsystems communicate via internal events. Based on the set of dependencies containing only control and not data dependencies, the slicing and the change impact analysis is affected by the different system characteristics.

Results. For the evaluation of our approach and its prototypical implementation, we derive two research questions to be answered:

RQ1. Is incremental model slicing *applicable* as change impact analysis to reason about retesting of test cases in incremental SPL testing?
RQ2. Do we achieve a gain in *effectiveness* from the slicing-based retest test-case selection compared to retest-all [42]?

As already explained in Sect. 3, for the evaluation set-up, (1) we focus on all-transition coverage [37], (2) transition test goals are used as slicing criteria, (3) we choose the set of control dependencies proposed by Kamischke et al. [20], and (4) we select the *Previous Slice* scenario for the derivation of slice changes.

[1] https://eclipse.org/.
[2] http://www.emoflon.org/emoflon/.
[3] https://eclipse.org/modeling/emf/.
[4] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/.
[5] http://forsyte.at/software/cpatiger/.
[6] http://cpachecker.sosy-lab.org/.

Table 1. Results of change impact analysis and retest test-case selection

SPL	$\|P_{SPL}\|$	⌀ Test Goals (Transition/Retest)	⌀ Test Cases (New/Reuse/Obsolete)	⌀ Retest (Select/New)	⌀ Retest All
Example	3	24.0 (14.3/9.6)	17.0 (8.0/7.6/1.3)	4.6 (1.6/3.0)	7.6
Wiper	8	63.6 (17.5/46.1)	71.1 (14.1/33.5/23.5)	32.5 (21.0/11.5)	33.5
Mine pump	16	106.2 (33.5/72.7)	51.7 (4.6/26.3/20.8)	16.3 (13.5/2.8)	26.3
Vending machine	28	108.8 (15.4/93.4)	92.2 (5.2/16.0/70.9)	18.2 (14.6/3.6)	16.0

In Table 1, our results of the change impact analysis and the retest test-case selection are summarized denoted by average values of test goals and test cases. For change impact analysis, we consider the number of retest test goals referring to the impact of applied model changes between subsequently tested variants. As we can see, the three existing SPLs have a high number of retest test goals compared to their approximate model size represented by the number of transition test goals. Thus, the applied model changes have a large impact on common behavior detected based on our incremental slicing. For instance, the vending machine SPL has 93.4 retest test goals compared to 15.4 transition test goals on average, where we have a maximum of 208 retest test goals for variant p_{11} and a minimum of 27 for p_2. This difference is mainly caused based on the exchange of the offered beverages representing the main behavior of a vending machine variant, whereas the exchange of the payment currency has a rather small impact. Another crucial factor for the high number of retest test goals is the testing order. In the given default order, the exchange of beverages is alternating and, therefore, results in lots of model changes with a large impact on common behavior between variants. A different order may reduce these changes and also their impact to be retested. Similarly, for the wiper and the mine pump SPL, the testing order as well as the behavior of some features alternating added or removed from the variants result in high number of retest test goals.

By comparing the number of retest test cases generated and selected with our approach to retest-all, we see at first sight that our approach and retest-all executes similar numbers of test cases for the vending machine and the wiper SPL. Solely for the mine pump SPL and the running example, we achieve a reduction of test cases to be retested (16.3/4.6) against retest-all (26.3/7.6). However, based on a more precise consideration of the vending machine and wiper results, we see that this average values are of course influenced by the number of retest test goals. In those variants, where we have a high number of retest goals, e.g., for variant p_{11} (208) of the vending machine SPL, we require more covering test cases (34) with our approach in contrast to retest-all (20) as retest-all does not ensure our retest coverage criterion. But, for variants with a small number of retest goals, e.g., for variant p_4 (28) of the wiper SPL, we require less test case (26) than retest-all (51). Hence, our retest decisions depend also on the testing order as well as the added/removed behavior of some features as described above. Again, another testing order may improve also the results of the vending machine and wiper SPL for the retest test-case selection.

Summarizing, our results show the applicability of our approach (**RQ1**). Based on incremental slicing, we are able to detect the impact of applied model changes when stepping to subsequent variants captured by retest test goals. The determined retest test goals are then used to select test cases to be retested, where we obtain positive results in the reduction of test suite size to be enhanced by investigating testing orders in future work.

To answer **RQ2**, we evaluate the fault detection capability of our approach compared to retest-all [42] by means of a set of faults to be detected. Unfortunately, for the SPLs under consideration, we do not have such a set of real faults. As no faults exist, we utilize simulated ones derived by incorporating the changes applied to a test model when stepping to a subsequent variant. Therefore, we take changes captured in the model regression deltas into account, where added transitions as well as source and target states of removed transitions which are still contained in the test model are used for fault generation. We combine those elements with other transitions from the test model to define faults representing erroneous artifact interactions caused by changes and their impact on common behavior. For the evaluation, we generate for each variant a set of simulated faults. We randomly select 10 % of the faults from this set to obtain a random data set on which test cases are executed to validate the fault detection. Depending on the size of the original variant-specific fault set, we derive a maximum of different random data sets to investigate the fault detection capability.

In Table 2, our results for the fault detection capability are summarized denoted by average values of undetected (alive) and detected (dead) faults. As we can see, our approach has for all four SPLs a good fault detection rate and performs better compared to retest-all. We obtained the best results for the vending machine SPL, where almost all faults are detected (0.08 alive) by our determined retest suites. But, this result must be relativized when considering the results shown in Table 1. On average, we select more test cases as retest-all allowing for a better chance to detect faults. However, for those variants, where we select less test cases than retest-all, we still have a better detection rate. For the wiper SPL, we select slightly less test cases compared to retest-all, where only three of the eight variants under test increase the average values. Thus, our approach allows for a good detection rate with a reduced set of test cases to be retested. This is supported based on the results of the mine pump SPL. Here,

Table 2. Results of the approach effectiveness compared to retest-all

SPL	∅ Faults	# Fault Sets	Size Fault Sets	Approach ∅ Alive/∅ Dead	Retest-all ∅ Alive/∅ Dead
Example	17	22	7	0.11/6.89	2.84/4.16
Wiper	69	22	7	1.05/5.95	3.05/3.95
Mine pump	136	54	17	3.79/13.21	5.11/11.89
Vending machine	73	30	10	0.08/9.92	0.96/9.04

we obtained a good detection rate with reduced variant-specific retest suites (cf. Table 1).

Summarizing, our approach shows a better fault detection capability compared to retest-all (**RQ2**). For all SPLs, we detect more faults than retest-all, where in the worst cases our approach detect and miss at least the same numbers of faults. The undetected faults mainly belong to artifact interactions seeded in model parts, where changes had no impact and are ignored by our approach.

Threats to Validity. For our approach and its evaluation, the following threats to validity arise. Due to varying interpretations of a systems' behavior, test modeling may result in different models to be used for model-based testing processes. This problem exists in general for model-based testing [37] and is not a specific threat for the delta-oriented test models of the four SPLs. To cope with this threat, we compared our re-engineered models with the original documented models [10] to ensure that both instances specify the same behaviors.

The non-existence of faults is a potential threat. For evaluating the effectiveness, we require the existence of faults detectable during test-case execution. Therefore, we derived simulated faults representing potential erroneous artifact interactions. We further select several randomly chosen fault sets to obtain varying data sets for each variant under test. In future work, we want to use a model-based mutation testing framework [2] for fault generation and apply the approach on real SPLs with an existing fault history.

We evaluated our approach by means of four SPLs with different system characteristics. Based on the obtained results showing a gain in effectiveness, we propose the assumption that our obtained results, up to a certain extent, are generalizable to other SPLs as well. However, we must investigate this assumption by performing more experiments with more complex systems. In addition, all four SPLs are modular event-based systems, i.e., the behavior is specified based on several subsystems, which are synchronized and controlled via events. To find the barrier for which systems our approach does not fit, we must include other types of systems in future experiments.

The choice of the coverage criterion used for test-case generation is a relevant factor influencing the obtained test suites. We focused on all-transition coverage, whereas more complex criteria exists to be considered in future experiments.

In this paper, we apply only control dependencies for the slicing computation due to our prior work [24]. As next step, we integrate also data dependencies and more complex control dependencies [4] allowing for a more fine-grained change impact analysis and, therefore, to ensure a more conclusive retest decision.

The testing order is also a threat. We applied the default order obtained from FeatureIDE, but other testing orders may influence our results. An investigation of the best testing order optimized for incremental SPL testing is required to be used for future evaluations to consolidate the achieved positive results.

The choice of the test-case generator may represent a threat. Depending on the applied generator, the obtained test suites may differ and, thus, the selection of test cases to be retested. However, our approach is independent of a specific test-case generator only required for providing test cases.

The neglection of factors such as time required for change impact analysis or test-case execution cost may be threats. The analysis time is a crucial factor for a successful retest test-case selection. Likewise, testing costs are important as there are solely limited testing budgets/resources available. We will perform a comprehensive evaluation in future work, where the analysis time is measured as well as limited testing resources are taken into account.

5 Related Work

We discuss related work regarding SPL regression testing as well as the application of slicing for change impact analysis. For a discussion concerning related (1) variability-aware slicing approaches, e.g., conditioned model slicing for annotated state machines [20], and (2) incremental slicing techniques, e.g., to support software verification [40], we refer to our prior work on incremental slicing [24].

SPL Regression Testing. Existing techniques realize SPL regression testing in the industrial context [13,33,34], for product-line architectures [11,22,29], for sample-based testing [32], as well as to allow for incremental testing [5,6,12, 25,26,38,39]. Uzuncaova et al. [38] propose one of the first incremental strategies for SPL testing, where test suites are incrementally refined for a variant under test. Baller et al. [5,6] present multi-objective test suite optimization for incremental SPL testing by incorporating costs and profits of test artifacts. Varshosaz et al. [39] present delta-oriented test-case generation, where delta-oriented test models facilitate test-case generation by exploiting their incremental structure. Compared to our approach, where test cases are selected for retest, those techniques focus on the determination and optimization of SPL test suites. Dukaczewski et al. [12] propose an adaption of our previous work [25,26] for incremental requirements-based SPL testing. In contrast, we focus on finite state machines facilitating a more fine-grained reasoning about behavioral change.

Slicing for Change Impact Analysis. Mainly white-box regression testing approaches exist applying slicing for change impact analysis [1,7,8,14,17,35]. Agrawal et al. [1] propose three types of slices for a test case comprising its executed program statements, where a modification of at least one of these statements indicates a retest. Jeffrey and Gupta [17] adopt this technique and define a prioritization of test cases to be re-executed. Bates and Horwitz [7] present slicing on program dependence graphs and reason about retest based on slice isomorphism. Gupta et al. [14] propose program slicing to identify affected def-use pairs to be retested by selected test cases. Tao et al. [35] present object-oriented program slicing for change impact analysis by incorporating the logical hierarchy of object-oriented software. We refer to Binkley [8] for an overview of program slicing techniques applied for change impact analysis. In contrast to those techniques, our approach is applied in model-based testing and it does not cope with evolution of software, but rather with incremental testing of SPLs. In the context of model-based regression testing, Korel et al. [21], and Ural and Yenigün [36] propose retesting of test cases based on dependence analysis. Both techniques are similar to our approach, also starting their analysis based on model differences to reason about change impact. In contrast to our approach, where slices are defined for model elements, their dependence analysis is applied on the model path of a test case resulting in a different retest test-case selection.

6 Conclusion

In this paper, we proposed the application of incremental slicing as change impact analysis technique for automated retest decisions in SPL regression testing. By stepping to a subsequent variant under test, we capture differences between slices as slice changes indicating the impact of model changes, i.e., changed dependencies between model elements. Based on a novel retest coverage criterion, each slice change represents a retest test goal to be covered by reusable or newly generated retest test cases. We prototypically implemented and evaluated our approach concerning its applicability and effectiveness by means of four SPLs.

We obtained positive results to be enhanced in future work by (1) considering more elaborate control and data dependencies [4], and (2) also applying forward slicing facilitating a more fine-grained analysis and more comprehensive retest decisions. Furthermore, we plan a case study by means of a real SPL from the medical domain provided by our industrial partner, in which we also want to investigate how we can apply our approach in practice. In addition, we want to adapt our analysis technique to cope with SPL evolution. In this context, an investigation of the first application scenario for change impact analysis (cf. Sect. 3) is aspired as this scenario seems to be more preferable when SPL evolution arises.

References

1. Agrawal, H., Horgan, J.R., Krauser, E.W., London, S.: Incremental regression testing. In: ICSM 1993, pp. 348–357. IEEE Computer Society (1993)
2. Aichernig, B.K., Brandl, H., Jöbstl, E., Krenn, W., Schlick, R., Tiran, S.: Killing strategies for model-based mutation testing. Softw. Test. Verif. Reliab. **25**(8), 716–748 (2014)
3. Al-Hajjaji, M., Thüm, T., Meinicke, J., Lochau, M., Saake, G.: Similarity-based prioritization in software product-line testing. In: SPLC 2014, pp. 197–206 (2014)
4. Androutsopoulos, K., Clark, D., Harman, M., Krinke, J., Tratt, L.: State-based model slicing: a survey. ACM Comput. Surv. **45**(4), 53:1–53:36 (2013)
5. Baller, H., Lity, S., Lochau, M., Schaefer, I.: Multi-objective test suite optimization for incremental product family testing. In: ICST 2014, pp. 303–312 (2014)
6. Baller, H., Lochau, M.: Towards incremental test suite optimization for software product lines. In: FOSD 2014, pp. 30–36. ACM (2014)
7. Bates, S., Horwitz, S.: Incremental program testing using program dependence graphs. In: POPL 1993, pp. 384–396. ACM (1993)

8. Binkley, D.: The application of program slicing to regression testing. Inf. Softw. Technol. **40**(1112), 583–594 (1998)
9. Clarke, D., Helvensteijn, M., Schaefer, I.: Abstract delta modeling. In: GPCE 2010, pp. 13–22 (2010)
10. Classen, A.: Modelling with FTS: a collection of illustrative examples. Technical report P-CS-TR SPLMC-00000001, PReCISE Research Center, University of Namur (2010)
11. Da Mota Silveira Neto, P., do Carmo Machado, I., Cavalcanti, Y., de Almeida, E., Garcia, V., de Lemos Meira, S.: A regression testing approach forsoftware product lines architectures. In: SBCARS 2010, pp. 41–50 (2010)
12. Dukaczewski, M., Schaefer, I., Lachmann, R., Lochau, M.: Requirements-based delta-oriented SPL testing. In: PLEASE 2013, pp. 49–52 (2013)
13. Engström, E.: Exploring regression testing and software product line testing - research and state of practice. LIC dissertation, Lund University (2010)
14. Gupta, R., Harrold, M.J., Soffa, L.: Program slicing-based regression testing techniques. Softw. Test. Verif. Reliab. **6**, 83–112 (1996)
15. Harrold, M.J.: Testing: a roadmap. In: ICSE 2000, pp. 61–72. ACM (2000)
16. Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., Traon, Y.L.: Bypassing the combinatorial explosion: using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans. Softw. Eng. **40**(7), 650–670 (2014)
17. Jeffrey, D., Gupta, R.: Test case prioritization using relevant slices. In: COMPSAC 2006, vol. 1, pp. 411–420 (2006)
18. Wang, J., Dong, W., Qi, Z.-C.: Slicing hierarchical automata for model checking UML statecharts. In: George, C.W., Miao, H. (eds.) ICFEM 2002. LNCS, vol. 2495, pp. 435–446. Springer, Heidelberg (2002)
19. Johansen, M.F., Haugen, O., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. In: SPLC 2012, pp. 46–55. ACM (2012)
20. Kamischke, J., Lochau, M., Baller, H.: Conditioned model slicing of feature-annotated state machines. In: FOSD 2012, pp. 9–16 (2012)
21. Korel, B., Tahat, L., Vaysburg, B.: Model-based regression test reduction using dependence analysis. In: ICSM 2002, pp. 214–223 (2002)
22. Lachmann, R., Lity, S., Lischke, S., Beddig, S., Schulze, S., Schaefer, I.: Delta-oriented test case prioritization for integration testing of software product lines. In: SPLC 2015, pp. 81–90 (2015)
23. Lee, J., Kang, S., Lee, D.: A survey on software product line testing. In: SPLC 2012, pp. 31–40. ACM (2012)
24. Lity, S., Baller, H., Schaefer, I.: Towards incremental model slicing for delta-oriented software product lines. In: SANER 2015, pp. 530–534 (2015)
25. Lochau, M., Lity, S., Lachmann, R., Schaefer, I., Goltz, U.: Delta-oriented model-based Integration testing of large-scale systems. J. Syst. Softw. **91**, 63–84 (2014)
26. Lochau, M., Schaefer, I., Kamischke, J., Lity, S.: Incremental model-based testing of delta-oriented software product lines. In: Brucker, A.D., Julliand, J. (eds.) TAP 2012. LNCS, vol. 7305, pp. 67–82. Springer, Heidelberg (2012)
27. McGregor, J.D.: Testing a software product line. Technical report CMU/SEI-2001-TR-022, Carnegie Mellon University (2001)
28. da Mota Silveira Neto, P.A., Carmo Machado, I.D., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R.: A systematic mapping study of software product lines testing. Inf. Softw. Technol. **53**, 407–423 (2011)
29. Muccini, H., van der Hoek, A.: Towards testing product line architectures. Electron. Notes Theor. Comput. Sci. **82**(6), 99–109 (2003). TACoS 2003

30. Oster, S., Wübbeke, A., Engels, G., Schürr, A.: A survey of model-based software product lines testing. In: Zander, J., Schieferdecker, I., Mosterman, P.J. (eds.) Model-Based Testing for Embedded Systems, pp. 338–381. CRC Press, Boca Raton (2011)
31. Pohl, K., Böckle, G., Linden, F.J.V.D.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heldelberg (2005)
32. Qu, X., Cohen, M.B., Rothermel, G.: Configuration-aware regression testing: an empirical study of sampling and prioritization. In: ISSTA 2008, pp. 75–86 (2008)
33. Runeson, P., Engström, E.: Chapter 7-regression testing in software product line engineering. Adv. Comput. **86**, 223–263 (2012). Elsevier
34. Runeson, P., Engström, E.: Software product line testing - a 3D regression testing problem. In: ICST 2012, pp. 742–746. IEEE (2012)
35. Tao, C., Li, B., Sun, X., Zhang, C.: An approach to regression test selection based on hierarchical slicing technique. In: COMPSACW 2010, pp. 347–352 (2010)
36. Ural, H., Yenigün, H.: Regression test suite selection using dependence analysis. J. Softw.: Evol. Process **25**(7), 681–709 (2013)
37. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., Burlington (2006)
38. Uzuncaova, E., Khurshid, S., Batory, D.: Incremental test generation for software product lines. IEEE Trans. Softw. Eng. **36**(3), 309–322 (2010)
39. Varshosaz, M., Beohar, H., Mousavi, M.R.: Delta-oriented FSM-based testing. In: Butler, N., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 366–381. Springer, Switzerland (2015). doi:10.1007/978-3-319-25423-4_24
40. Wehrheim, H.: Incremental slicing. In: Liu, Z., Kleinberg, R.D. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 514–528. Springer, Heidelberg (2006)
41. Weiser, M.: Program slicing. In: ICSE 1981, pp. 439–449 (1981)
42. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Softw. Test. Verif. Reliab. **22**(2), 67–120 (2007)