# A Survey on Software Product Line Testing

Jihyun Lee
College of Liberal Arts, Daejeon University,
62 Daehak-ro, Dong-gu, Daejeon, Korea
jihyun30@kaist.ac.kr

Sungwon Kang, and Danhyung Lee
Department of Computer Science, KAIST
291 Daehak-ro, Yuseong-gu, Daejeon, Korea
{sungwon.kang, danlee}@kaist.ac.kr

## ABSTRACT
Software product line (SPL) testing consists of two separate but closely related test engineering activities: domain testing and application testing. Various software product line testing approaches have been developed over the last decade, and surveys have been conducted on them. However, thus far none of them deeply addressed the questions of what researches have been conducted in order to overcome the challenges posed by the two separate testing activities and their relationships. Thus, this paper surveys the current software product line testing approaches by defining a reference SPL testing processes and identifying, based on them, key research perspectives that are important in SPL testing. Through this survey, we identify the researches that addressed the challenges and also derive open research opportunities from each perspective.

## Categories and Subject Descriptors
D.2.13 [Software Engineering]: Reusable Software – *domain engineering*.

## General Terms
Reliability, Verification.

## Keywords
Software product line testing; Software product line engineering; Software testing

## 1. INTRODUCTION
In software product line (SPL) engineering, domain engineering sets up a common product line platform by identifying commonality and variability while application engineering develops individual products based on the platform. Domain testing produces test assets that will be reused by products in the product line. Domain testing includes testing for common parts related to variable artifacts that may or may not be realized during domain engineering. Meanwhile, application testing has to achieve an efficient reuse of domain test assets while it tests application-specific parts.

During the past ten years, many approaches to SPL testing have been proposed. Several surveys on SPL testing have also been conducted [1, 2, 3, 4, 33, 34, 45]. However, while addressing questions such as what topics the past research has focused on, they missed other important questions such as the main research challenges in domain testing and application testing and how such challenges have been resolved in each approach. Nor did the studies [34, 45] that map out selected researches to pre-defined research issues answer such questions. This paper tries to compare and analyze in detail how the existing approaches tackle the research challenges. To compare and analyze the existing SPL testing approaches, this paper first defines a survey framework that consists of a reference SPL testing process and research perspectives. A reference SPL testing process clearly distinguishes SPL testing from testing in a single-system development environment. We analyze perspectives in SPL testing based on the reference SPL testing processes. For comparison and analysis, this paper selects from the published literature approaches with significant contribution to the defined perspectives. Then, based on the survey framework, with the contributions of the approaches research opportunities thus far unresolved by the current researches are identified from the gaps in between the survey frameworks and the contributions.

This paper is organized as follows: Section 2 presents our survey framework. Section 3 compares and analyzes the existing approaches based on the survey framework. Section 4 summarizes the open research questions, and finally, Section 5 concludes the paper.



*SPLC '12*, September 02 - 07 2012, Salvador, Brazil


## 2. THE SURVEY FRAMEWORK
In this section, we define a reference SPL test process that provides the basis for analyzing the differences between SPL testing and single product testing. Based on the differences a survey framework that consists of perspectives as major research challenges and observations is derived. Then we select and classify researches on SPL testing for survey. In this paper, the term 'testing' refers to the activities that include test case design (i.e. description of test objectives, test data, test actions, expected results, and execution preconditions for a test item) and test execution (i.e. running a test on the domain or application artifacts and producing its results).

### 2.1 The reference SPL testing process
SPL testing has a 'W'-shape lifecycle [5], called extended V-model in [46, 47] formed by two overlapping V-models as Figure 1 shows. The dotted arrows from left to right in Figure 1 indicate that domain test assets are used as inputs to application testing. Test assets such as test plans, test cases, and test scenarios must be produced in the relevant engineering phase. To realize such test assets, test engineers initiate system testing in the domain or application requirements engineering phase, integration testing in the domain or application architecture design, and unit testing in the domain or application realization phase.

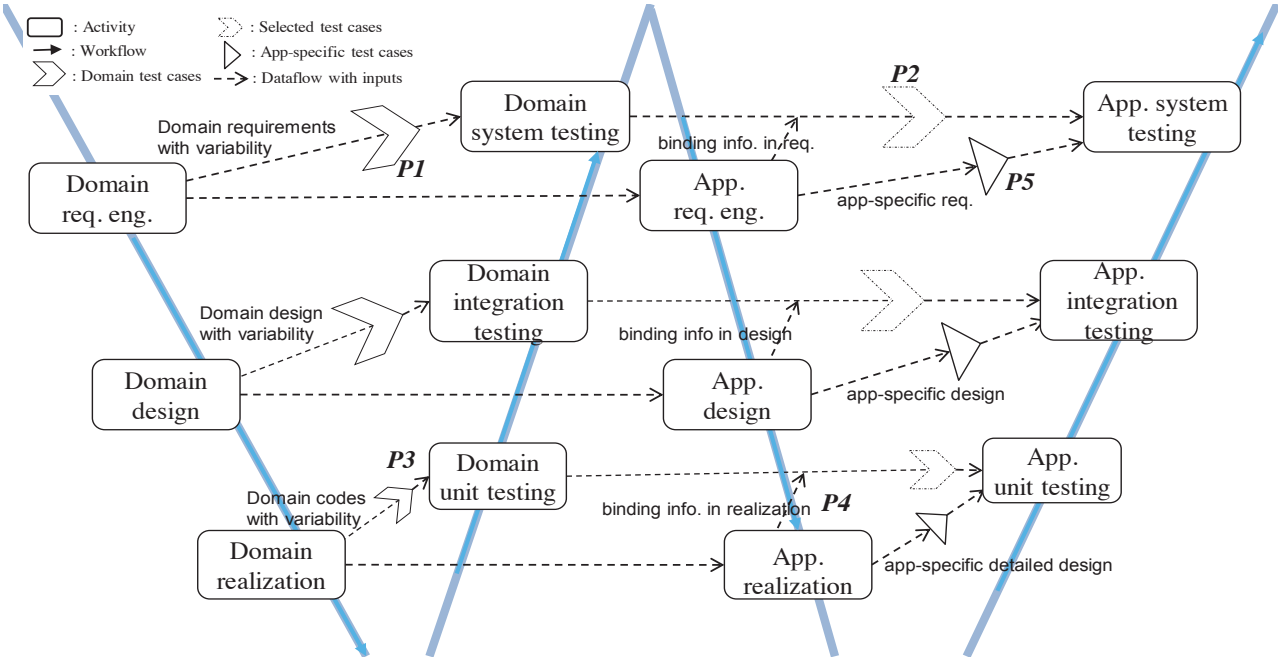


**Figure 1. A reference SPL testing process**

Test assets can be produced and executed either in domain testing or in application testing. However, it is desirable to test core assets in domain testing so that application testing can focus on the application specific parts not covered in domain testing. In the normal case, complete products are not obtained during domain engineering because domain engineering focuses on core asset development. Therefore, in most cases domain system testing can only be conducted in a limited way.

## 2.2 A survey framework and research selection

In this sub-section we present the survey framework used in this paper for comparing and analyzing existing approaches. The survey framework consists of eight perspectives, where five of them are derived from the reference SPL testing process and three of them are defined to assess maturity of approaches.

The major difference between SPL testing and single product testing comes from variability in domain artifacts. Handling variability is very challenging and the decisions on how to deal with it are the starting point in SPL testing. From this view, as the reference SPL testing process in Figure 1 shows there are two types of test cases in SPL testing, domain test cases (test cases produced during domain engineering) and application test cases (test cases produced during application engineering).

Domain test cases should have the forms that can be efficiently reused in application testing and address variability. In addition, test data for commonality and variability must be examined and determined for test cases [9]. Further, an approach should have a way for testing a member product of a product line that reuses domain test cases and minimizes retesting for the parts that are already tested in domain testing or by another member product [45]. For them, a reasonable test case selection method should be provided so as to select domain test cases to be executed. Therefore, test case creation (which, in this paper, includes test data, *P1: Test case creation in Figure 1*) and test case selection are research challenges for SPL testing (*P2: Test case selection in Figure 1*).

In addition, seldom all variants are implemented during domain engineering. So domain testing must consider tests for non-executable domain artifacts due to the undeveloped variants, i.e. absent variants [7]. Coping with absent variants is also challenging because absent variants can complicate integration testing and system testing, making domain system testing impossible in some cases. However, because the quality of domain artifacts affects the quality of all member products in a product line, test execution of parts linked to absent variants should be carefully handled during domain testing (*P3: Test execution for absent variants in Figure 1*).

Absent variants are bound during application engineering. The binding phases of absent variants can be widely apart [29], so application integration and system testing are much more complicated (*P4: Variability binding in testing in Figure 1*). If a product is tested after all variants have been bound, the defect correction cost becomes high. If testing is executed whenever binding occurs, much effort will be required for developing test stubs and drivers. Therefore, the cost tradeoff between defect correction and test code development should be considered.

Application testing tests application-specific artifacts and re-tests the domain artifacts that were already tested but need be re-examined as new functionalities are added or as the existing domain artifacts are adapted [46] for satisfying the application-specific requirements (*P5: Application-specific tests in Figure 1*).

For assessing maturity of approaches three perspectives are employed. Because SPL testing deals with multiple products, its complexity is very high and tool support is essential (*P6: SPL test tool support*). In many cases, the number of variations can be large, often hundreds or more, so the scalability is an important aspect (*P7: Scalability*), and the evidence on the feasibility of an approach is also important (*P8: Evidence*).

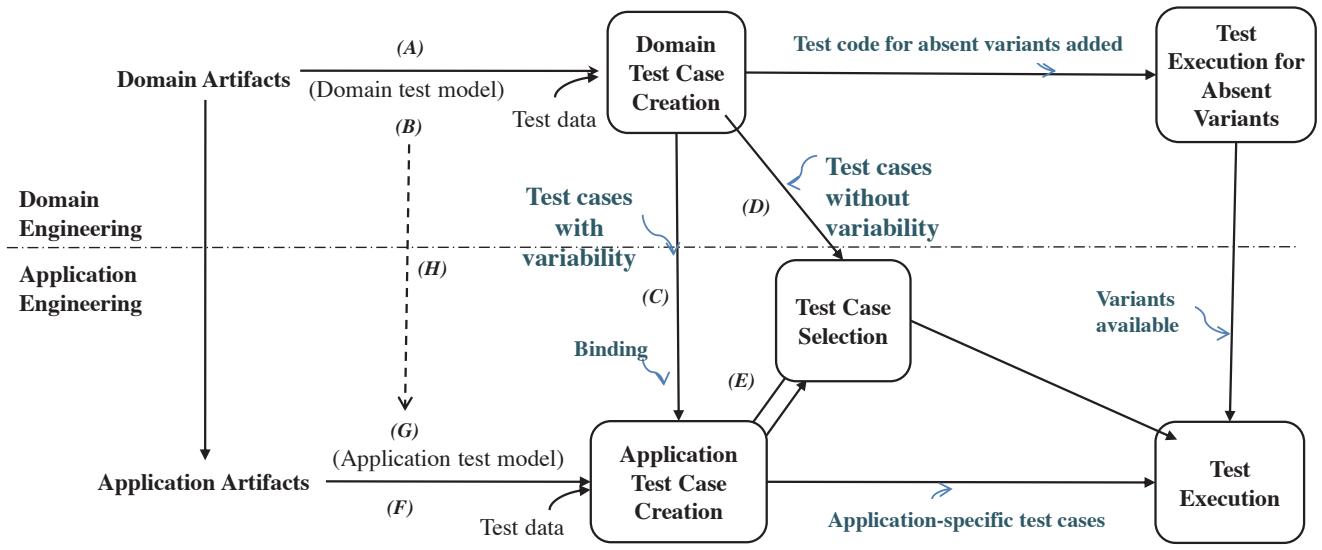


**Figure 2. Relationships of the perspectives P1-P5 in SPL testing process**

As the result, five research perspectives are:

P1: Test case creation;
P2: Test case selection;
P3: Test execution for absent variants;
P4: Variability binding in testing;
P5: Application-specific tests.

Three research perspectives defined for assessing the maturity of approaches are:

P6: Tool support;
P7: Scalability of the approach;
P8: Evidences of the approach.

The survey framework also includes observations and assumptions for the first five perspectives, which are the starting point of our comparison and analysis. For deriving observations to be used as the basis for comparison and analysis, we analyze P1 through P5 based on McGregor [8, 9] and Pohl et al. [3, 7] that provide general insights on SPL testing. Domain test cases can be generated in a form that includes variation points, which have to be resolved later [7], or can be separately generated for each set of variants for the variability [8]. Meanwhile, testing in application engineering reuses domain test cases across different products in the product line. In the case that domain test cases include variants, domain test cases are transformed into application test cases through variability binding and in other cases a mechanism for selecting test cases related to a specific application is necessary.

Domain test cases may be incomplete due to the variants that will be realized during application engineering. Even when test cases are complete, they may not be fully executable because they interact with variable parts [7]. Moreover, because binding times can vary widely spread across different testing phases, their executable phases can be diverse [6, 7]. From these considerations, we can make the following observations on the aforementioned five perspectives (P1 through P5):

O1. Domain test cases can be created either (O1-1) directly from domain artifacts (A in Figure 2) [8] or (O1-2) through domain test models derived from the domain artifacts (B-C in Figure 2) [7]. A domain test model is a test model that preserves variability while an application test model only preserves the values of variability.

O2. Test data is one of categories of test cases. A partial data set is associated with either commonalities or variabilities [8, 9].

O3. A domain test case may (O3-1) include or may (O3-2) not include variability [7, 8].

O4. Application test cases may be created directly from domain test cases by using binding information of application artifacts (C in Figure 2) [7].

O5. Test execution for absent variants can be performed (O5-1) during domain engineering by adding test code for absent variants or may be performed (O5-2) during application engineering when variants are available [7].

O6. A test case can be executed before or after variability binding in products (in an extreme case it can be executed after all variabilities are resolved), and the bindings can occur during development, compiling, linking, or run time [7].

We also make the following assumptions based on the observations and use them as the basis for comparison and analysis together with O1-O6:

A1. Application test cases may be created from application artifacts (F in Figure 2) or through application test models (derived either from H-G or from G in Figure 2).

A2. During application engineering test case selection may be conducted for selecting test cases to execute (D-E in Figure 2). A proper selection mechanism would allow a high degree of reuse of domain test cases.

A3. Application-specific test cases are created and executed during application engineering.

The eight perspectives cover research challenges that O. Edwin [1] proposes through a systematic review of the existing literature except for test process challenge reviewed and determined as an

open research challenge by Lamancha et al. [2] and E. Engström [33]. In addition, Neto et al. [34] and Engström et al. [45] conducted a systematic mapping study and their aspects such as variant binding time, commonality and variability testing are similar. However, our observations and assumptions for comparison and analysis are quite different.

This survey compares and analyzes existing approaches in the literature on the basis of the defined survey framework. Literatures including those screened in the two mapping studies [34, 45] have also been reviewed. We selected approaches to be surveyed based on the following criteria:

- Literature that proposes specific SPL testing approaches related to the five research perspectives with evidence <or>
- Literature that proposes SPL testing tools with well-defined approaches

As analyzed in [48], a few literatures describe experience from the real-world software environments. Among the researches screened in [45], most of the researches published in SPLiT Workshops were not selected because they only provide initial ideas rather than solutions with evidence. The following are the approaches selected for this survey:

- Bertolino et al. [20] – Scenario-based specification and testing of requirements.
- Cohen et al. [17, 18, 26] – Combinatorial interaction testing by considering constraints among features (Lamancha et al. [39] is an approach of the same type).
- Feng et al. [15] – Process-based unit testing.
- Ganesan et al. [16] – Architecture-based unit testing.
- Kakarontzas et al. [13] – Test-driven development.
- Lamancha et al. [32] – Model-driven test generation.
- Mallett et al. [23] – System testing using model-based and variant-management concepts.
- Nebut et al. [21] – System testing using functional variation points at requirements level.
- Neto et al. [11] – Use of regression testing in SPL.
- Olimpiew et al. [35, 36] – Model-based functional test design.
- Reis et al. [19, 31] – Interaction-based integration testing.
- Reuys and Kamsties et al. [3, 14, 22, 27, 30] – Scenario-based test case generation for domain and application system/integration testing.
- Stricker et al. [37] – Data flow based test generation.
- Tevalinna et al. [4, 28] – Framework and framelet-based application testing.
- Uzuncaova et al. [25] – Incremental testing for the possible configuration of a product line.

# 3. THE STATUS OF SPL TESTING RESEARCH

This section describes the results of comparing and analyzing the selected approaches based on the survey framework.

## 3.1 Test case creation

Existing surveys by O. Edwin [1] and Lamancha et al. [2] deal with the test case creation approaches. And Neto et al. [34] analyzes test case creation from regression testing and reusability. But those surveys do not differentiate the contributions of the test creation approaches in the surveyed references. Thus, this paper analyzes differences among test case creation methods presented in the literature based on the observation O1 and the assumption A1. We exclude Ganesan et al. [16] and Tevanlinna et al. [28] from analysis because they respectively focus solely on test execution and on SPL testing tool capability. Feng et al. [15] is also excluded because it starts from the assumption that there is a unit test case repository which includes reusable unit test cases defined at different abstraction levels.

Approaches to test case creation that use domain artifacts (O1-1) have been proposed in [18, 20, 23, 26]. Combinatorial approaches based on feature models, which are major domain artifacts of SPLE, are relevant to this classification [8]. The pair-wise combination method, which greatly reduces the number of combinations necessary for testing, has been presented in [17, 18, 26, 39, 40]. However, this method also has a disadvantage in that it considers interactions that do not exist between variabilities with the result that unnecessary test cases can be generated. The other approaches create generic test cases from domain artifacts that include variation points [20, 23]. These test cases contain a parameter for each variation point that is encountered in the scenario. These approaches define specific notations for modeling requirements such as PLUCs of the PLUTO approach [20] and generic requirements specifications [23], from which test cases can be created. They embed test data in the form of parameter values. However, the category partition method used in PLUCs does not consider the testing order because it just creates test cases through the combination of chosen categories.

Another possible classification of test case creation approaches is one that uses domain test models that include variability [19, 22, 32, 37]. These studies use an extended activity diagram [19, 22] or sequence diagram [32] developed from domain requirements artifacts as domain test models. However, the difficulty that arises because use case dependency is not considered has been observed in [24]. It is also difficult to assure that test models created from domain artifacts by a test engineer are consistent with domain requirements. Moreover, though these approaches should maintain a large number of test models for deriving test cases as mentioned in [8], they do not devise any solutions for this problem. Because the test cases of the approaches include variability, their reuse is complicated due to their management and binding problems. To resolve this, Siemens [27], who adopts the ScenTED approach [14, 19, 22], saves test fragments for all activities to a library before assembling them to automatically generate a test scenario in accordance with the chosen activity. ScenTED-DF (Data Flow) [37] extends ScenTED and considers data dependency together with control flows. However, this approach has high complexity and requires additional efforts for adding data flow attributes to domain test models.

The last classification of test case creation approach is one that uses application artifacts or application test models. An incremental method proposed by Uzuncaova et al. [25] and Kakarontzas et al. [13] creates new test cases based on the differences between the existing products/ components and their test cases. These approaches refine test cases by computing differences reflected in specifications due to the added features after test case generation for the base product (a desired set of features). Regression testing methods have been recommended for these approaches [10, 11] to reduce retesting. There is an approach that uses SPL architecture and code for regression testing in accordance with SPL evolution [11]. It considers critical variability with common parts with high priorities. Application testing modifies or re-generates test cases by analyzing the differences of graphs that are generated for catching code behaviors before and after the modification. However, it ignores that the initiative of SPL architecture evolution is on domain engineering not on application engineering. Nebut et al. [21]

presents an approach that creates test scenarios including test oracles from domain use cases, but it does not reach to test case creation and its reuse. Table 1 describes the contributions made by the approaches by the classification of their references.

**Table 1. Contributions with respect to the reference of test case creation**

| References | Contributions |
|---|---|
| Domain artifacts (O1-1) | • Defining SPL-specific notations for a reference such as Tabular forms or Product line use case (Bertolino et al., Mallett et al.)<br>• Providing algorithm for constructing valid combination of features (Cohen et al.) |
| Domain test models (O1-2) | • Deriving test models by extension of existing models for accepting variability (Lamancha et al., Reis et al., Reuys and Kamsties et al., Olimpiew et al.)<br>• Creating annotated data flow test models (Stricker et al.) |
| Application artifacts (A1) | • Creating generic or specific test scenarios including test oracles (Nebut et al.)<br>• Using differences between applications (Neto et al., Uzuncaova et al.)<br>• Using differences between components according to the component evolution for introducing variability (Kakarontzas et al.) |

**Table 2. Contributions with respect to test case creation for variability**

| Test case creation for variability | Contributions |
|---|---|
| Variability-included test cases (O3-1) | • Abstracting variabilities for enhancing reusability (Bertolino et al., Feng et al.)<br>• Representing domain test cases with variation points (Mallett et al., Lamancha et al., Reuys and Kamsties et al.)<br>• Considering nonfunctional requirements (Feng et al.)<br>• Test inputs in the form of possible parameter values (Bertolino et al., Mallett et al.) |
| Separate test cases for variants (O3-2) | • Reducing test cases related to variability (Cohen et al, Reis et al.)<br>• Analyzing data dependency with variants (Stricker et al.)<br>• Automatic test cases creation including test oracle (Nebut et al.)<br>• Considering nonfunctional requirements (Kakarontzas et al.) |

Table 2 presents analysis results of the selected approaches on how they address variability in their test cases (O3). The 'Test case creation for variability' column in Table 2 is the summary results of the presented solutions that the selected works provide. The first group consists of the approaches that produce variability-included test cases. Most approaches present test cases including variation points. Among them Feng et al. [15] and Reis et al. [31] address test case creation for nonfunctional requirements. Feng et al. [15] proposes creation of hierarchical test cases for the specific nonfunctional requirements. ScenTED-PT proposed in [31] is, an extension of ScenTED to map the variability in the performance requirements to test cases that preserve variability. And there are researches that deal with the verification of nonfunctional aspects [42, 43, 44]. Model-based integration test case scenarios (ITCS) [19] is an approach to reducing test efforts in domain engineering by abstracting variability that does not interact. The focus of ITCS is on the optimization of interactions between integrated components and it does not present concrete test cases. The variability and its relevant parts are considered as a placeholder that is replaced by a variant later. As for test data (O2), only Lamancha et al. [32] mentions test data with test cases, but it does not present any mechanism to inform the relation of test data with products.

The second group consists of the approaches that produce separate test cases for variants. Most of the system testing approaches create separate test cases for each possible variant by considering all possible combinations of variabilities [17, 18] or all combinations of member product relevant variabilities [20, 21, 32]. There is an approach that reuses the test cases derived from the different products [25]. Neto et al. [4] is excluded from analysis of test case creation for variability because it deals with regression test selection technique for SPL. There is an approach to reuse test cases of the different products [25]. However, no approaches in this class consider test data creation.

## 3.2 Test case selection

Existing surveys did not address the test case selection perspective. They focused on whether test cases realize reuse opportunity, but overlooked test case selection that considers reusability together with effectiveness. Accordingly, this paper analyzes SPL testing approaches from the test case selection perspective for realizing effective reuse, rather than just reuse of test cases.

As discussed in O4 and A2, test case selection can be conducted in application testing (D-E in Figure 2) or for reducing retests. For reusing domain test cases, a proper selection mechanism should be provided. After test case creation, test case selection can also be done with respect to test cases and test data. Among the selected approaches Neto et al. [15] and Striker et al. [37] deal with test case selection. Neto et al. [15] selects test cases through a graph comparison, but the selection is in regression testing in accordance with the SPL evolution. In contrast, Stricker et al. [37] provides a concrete test case selection method and considers data-flow with control flow to avoid omission of necessary testing in an application. Feng et al. [15] mentions test case selection from a unit test case repository but it does not provide an explicit mechanism for test case selection. Ensan et al. [44] proposes test case prioritization methods for selection focusing on prioritization of features in domain engineering. So it does not address test case selection for reducing redundant testing in application testing. As for test data, Lamancha et al. [32] provides a DataSelector for selecting test data from a test data pool. Mallett et al. [23] and Bertolino et al. [20] deal with test data selection by using parameterization, but they do not address it in a systematic way.

Therefore, a test case selection approach that includes test data selection is necessary for describing explicitly which domain test cases and variants are related to a specific product. For resolving the issue of reducing redundant testing by test case selection test case creation approaches must give careful consideration to increasing the reusability of test cases and minimizing redundant testing.

## 3.3 Test execution for absent variants

In the middle of domain engineering, components are loosely coupled with each other and there may be no executable products yet. Test execution must be conducted for the parts that include such variants. As mentioned in O5, the first group consists of the approaches that perform additional implementation for executing test cases for the non-executable parts, including absent variants. The Core Flight Software (CFS) product line [16] adapts the approach of additional implementation, adding test code for testing absent variants. It implements a mock or stub for the undeveloped modules (i.e. modules relevant to absent variants) for executing unit testing for the common modules. A mock or stub is replaced by a real one after the module is implemented. CFS performs integration testing incrementally by implementing a mock module when a module being tested includes variability or has an absent variant. The fRamework Integration and Testing Application (RITA) [28] also supports test execution for variability through a virtual binding that involves the use of drivers and stubs.

The second group consists of the approaches that delay test execution until binding. Feng et al. [15] adopts the approach of delaying test execution, testing a component after generating a executable test case that is specific to a AOP platform. While the method in the study does not explicitly refer to test execution for absent variants, we can guess that test execution is delayed until the variants are available. In the case of integration testing it basically executes only the test cases for common interactions and test cases that contain few variable interactions with components that are already realized are executed [7]. As a similar case an incremental combination approach executes all test cases (test cases for commonality and variability) that are created during domain engineering after a product has been implemented, i.e. they are executed at run time [17, 18]. The incremental test generation approach [25] assumes that all products are built by a combination of features. So the approach does not consider absent variants separately. It executes system testing after all features of a product have been combined. None of the integration testing approaches [19, 31] and system testing approaches [3, 14, 20, 21, 22, 23, 27, 30, 32] in the literature mention test execution for absent variants, but they consider that all test scenarios can be executed after a product is fully integrated.

In the regression testing approach [11], tests with absent variants are executed only for the critical variability. Table 3 describes the results for test execution for absent variants. The approaches that do not mention test execution are excluded from analysis.

**Table 3. Contributions with respect to test execution for absent variants**

| Test execution | Contributions |
|---|---|
| Adding test code for absent variants (O5-1) | • Providing experience based principles in unit testing (Ganesan et al.)<br>• Visualizing bound relations for variation points (Tevanlinna et al.) |
| Delaying until when all variants are available (O5-2) | • Incremental combination of available variants (Cohen et al., Uzuncaova et al.)<br>• Narrowing down abstraction level from functional to source code level to test nonfunctional concerns (Feng et al.) |

When adding test code, the test time can be long and the cost can be high. However, when testing for an absent variant is delayed until it is available, correction cost may be high because the faults in a platform have an influence on all products within a product line. Thus, it is necessary to optimize these approaches. Analysis indicates that most approaches except for Neto et al. [11] focus on how to test parts including absent variants or those related to variability without considering the significance level of variability. In the case of Neto et al. [11], possible cases of maintenance or evolution are assumed and SPL regression testing approaches are presented. Neto et al. [11] also offers analysis of the efficiency of each approach, but it mainly deals with regression testing in accordance with SPL evolution.

## 3.4 Variability binding in testing

As discussed in O6 in Section 2, variability binding has a considerable influence on testability because binding can occur at development time, at compiling time, at linking time, at loading time, or at run time. This sub-section deals with test execution in terms of binding decisions. In the case that binding occur at the development time, unit and integration testing can be executed in all application testing phases after implementation, even though all variants are not bound, but system testing cannot [29]. Namely, unit testing and integration testing are possible because we can know a variant to be selected. Approaches that insert test code [16, 28] are relevant to this case. The combinatorial integration approach [18] assumes all variation points are bound at runtime. However, this is only a special case because bindings are possible at all development phases.

There is a research [29] on how variability binding time affects SPL testability, and Neto et al. [34] also points out the lack of evidences on variability binding regarding strategy for handling variability within test assets, effort reduction, and traceability. From test execution aspect, most works consider bindings that occur at development time or runtime. However, they do not mention explicitly how to deal with variabilities that have different binding times. Moreover, no works address the cases in which binding occurs at compiling, linking, or loading time. Therefore, researches considering these binding times that span from development time through to run time are necessary.

## 3.5 Application-specific test

Application engineering is directly relevant to customer products and often needs to deal with changes in customer needs. Domain artifacts may not satisfy the specific product's needs completely. Thus, there are gaps between what is available and what is required. Unsatisfied needs may be met by implementing application-specific artifacts or by adapting domain artifacts to fill the gaps. After application-specific artifacts are developed, they must be tested with each application (A3 in Section 2).

There are few approaches that mention about application-specific test. Regression testing [10, 11] approach, the Test Driven Development (TDD) approach [13], and ScenTED approach [22] deal with testing for application-specifically modified or added requirements. Currently proposed regression testing approaches simply apply regression testing used in the single system. The TDD approach, ScenTED approach, and an approach using the Alloy formula [25] generate application-specific test cases by modifying the existing domain test cases or by adding new test cases. However, they provide test case creation for application–specific requirements, but they did not deal with how to reuse domain test artifacts and how to avoid redundant testing with domain testing.

## 3.6 SPL test tool support

Tool support in SPL testing is essential in reducing test efforts and effective/efficient testing for multiple products. However, variability included in test objects requires extensions of the existing testing tools that have supported the single system testing. Thus far, there are no dominant testing tools for product lines. As for unit testing, the Generative Aspect-oriented Testing framEwork (GATE) tool [15], a prototype tool, has been proposed to generate test cases automatically from the unit test case repository according to the process-based unit test plan (PUPT). In the case of integration testing and system testing, there are tools such as Kesit [25] that generate test cases for products by using SAT-base analysis for the incremental test generation approaches and a tool [26] for creating test cases for all possible products by transforming SPL feature diagrams to Alloy specifications. However, they were developed just for laboratory experiments. Philips developed a tool for deriving system test cases from activity diagrams [30], but this tool does not support transformation from use case document to use case models so the transformation should be performed manually beforehand. As for the ScenTED-DTCD tool [22], in order to use the tool a test engineer should manually create an activity diagram and test scenarios for a product from use cases in advance. As evaluated in the Siemens medical case [27], a shortcoming of the ScenTED technique is the lack of integrated tool support. RITA was introduced in [28], but it was not evaluated through industry application and has a scalability problem.

## 3.7 Assessing the scalability problem

Perrouin et al. [26] demonstrates *Binary Split* and *Incremental Growth strategies* by using AspectOPTIMA, a product that has 20 features, and then compares their efficiency. AspectOPTIMA is a real-world feature model, but it is hard to consider it as having a real scalability. The approach proposed by Cohen et al. [18] provides an example where all bindings occur at runtime, so it is only applicable to one special case of product line development. The ScenTED approach was adapted or automated in the SIENET COSMOS product line [27] of Siemens and Philips [30]. ScenTED provides a tool and a technique that make it possible to automatically generate domain test cases and application test cases from a sequence diagram or an activity diagram. However, it is hard to regard ScenTED as a scalable method because ScenTED requires all the requirements of a large-scale system to be described in sequence diagrams or activity diagrams, include variability. There is a web browser case in Nokia [12] that is not included in the selected approaches because it does not explain the detailed testing approach in its description of the reuse of test suites by applying the regression testing approach.

## 3.8 Assessing the evidences

We analyzed the maturity of SPL testing approaches in terms of the evidence they provide. We classify the justification and validation types by using the classification of empirical studies as used in Zannier et al. [38]. Most approaches only provide an example or lab experiment results. Except for the ScenTED approach, which has been tailored and validated in several fields [27, 30, 31], most approaches do not provide any reproducible validation results. Table 4 describes the evidence level of each approach.

**Table 4. Summary with respect to the level of validation**

| Study type [35] | Approaches |
|---|---|
| Controlled experiment | Reis et al., Stricker et al. |
| Quasi experiment | Cohen et al., Neto et al. |
| Case Study | No selected literatures |
| Exploratory case study | Reuys and Kamsties et al. |
| Example application | Bertolino et al., Feng et al., Kakarontzas et al., Lamancha et al., Mallett et al., Nebut et al., Olimpiew et al., Tevanlinna et al., Uzuncaova et al. |
| Experience report | No selected literatures |
| Meta-analysis | Ganesan et al. |
| Survey | No selected literatures |
| Discussion | No selected literatures |

# 4. RESEARCH OPPORTUNITIES

The existing surveys [2, 34] have already pointed out that there are very few researches on testing of non-functional aspects and test levels other than system testing. To provide further insights this paper derived observations and assumptions for the defined perspectives in Section 2 and described contributions of the selected approaches from Sections 3. Table 5 summarizes the current status of the SPL testing research.

**Table 5. Assessment of the overall status of SPL testing research**

| Perspectives | Not | Marginally | Partially | Fully |
|---|---|---|---|---|
| P1.Test case creation | | | √ | |
| P2.Test case selection | | √ | | |
| P3.Test execution for absent variants | | √ | | |
| P4.Variability binding in testing | | √ | | |
| P5.Application-specific tests | √ | | | |
| P6.Tool support | | √ | | |
| P7.Scalability of the approach | √ | | | |
| P8.Evidences of the approach | | √ | | |

Research opportunities implied by our survey of the SPL testing research can be summarized as follows:

1. Comparison and analysis results from P1: (1) *Construction of test references should be more systematic*. Most approaches prepare manually test references that are the bases for deriving test cases or deal only with test cases creation without test references description or do not mention about them at all. And compatibility problem of test references with existing modeling notations should be solved; (2) *Test coverage of domain and application testing should be clearly defined*. Most approaches are described without distinguishing domain and application testing activities. Thus, it is difficult to catch the scope of domain and application testing; (3) A *standard test case specification template should be defined for both domain testing and application testing. Details describe complete test case context such as inputs, pre- and post- conditions, expected results and variability specific details*. Compared with the test

specification for a single system [41], existing SPL testing approaches missed many elements of test case specification such as inputs, expected results, and execution conditions. Moreover, the forms of test cases as in Cohen et al., Reis et al., Kakarontzas et al., etc., are not clearly presented. Their approaches do not derive concrete test cases; (4) *Research on systematic test data derivation is needed*. Stricker et al. [37] points out that the concrete test data derivation is a general research topic, but test data derivation considering variability is a SPL testing specific topic so it should be conducted.

2. Comparison and analysis results from P2: (1) *Test case selection criteria that reflect variability resolution decisions and application-specific variability should be provided*. For avoiding redundant testing in application testing test case selection criteria for choosing test cases to be executed should be determined. No approaches except for Stricker et al. [37] explicitly consider test case selection in application testing. (2) *Research on regression testing due to the different selections of variants among applications is needed*. In SPL testing, regression testing approaches can be conducted to test a member application that uses platforms already tested in domain testing and whose selected variants are a little different with those of already tested member application. There are regression testing approaches for supporting incremental integration of features or evolution of SPL architecture. However, no approaches address them. (3) *Trade-off analysis should be done for test case selection or test-all alternatives in application testing*. As Neto et al. [15] pointed out, test case selection can be justified when its effort is sufficiently less than executing the entire test cases. Pohl et al. [7] provides overall evaluation results for SPL test strategies, but its focus is not on test case selection and its results are subjective. Thus, researches on objective evaluation for test case selection efforts in application testing are necessary.

3. Comparison and analysis results from P3: (1) *Approaches should support test code creation and their reuse*. SPL testing, especially unit and integration testing might require a large amount of test codes for the absent variants. In SPL, test code may be reusable and their production is not the only responsibility of application testing. Because the behaviors of variabilities are determined in domain engineering many parts of test code should be generated in domain testing and most of them should be reused in application testing. Thus, test execution for absent variants should also be dealt from domain testing with reuse perspective. (2) *Glue code that connects unmatched interfaces should be included in both domain and application testing*. In SPL glue code for accepting variability is an important test item, but no approaches consider it.

4. Comparison and analysis results from P4: (1) *Binding mechanisms are an important factor to determine test approaches, details thus are described in the test case creation, selection, and execution approaches*. Binding mechanisms that provide the means to locate variants and to determine which variants have to be bound [7] may have influence on testing because such mechanisms determine the way of structuring modules, coding, or execution. However, most approaches do not consider this aspect for test case creation and execution. (2) *Binding times should be associated with the relevant domain test assets*. During domain engineering, when binding mechanisms as well as binding time are defined, the domain engineers together with test engineers are responsible for associating binding time with domain test assets so as to let application testing know the binding application times for the test assets.

5. Comparison and analysis results from P5: (1) *Research on reusing or modifying test assets for application-specific testing should be strengthened*. To some extent, testing application-specific parts is similar to the testing in single software development [7]. However, it should not be considered in isolation from domain testing (creation, selection and execution) because commonalities and selected variability should be integrated with application-specific requirements. Thus, the domain or application test assets must be reused or modified for testing application-specific requirements. Regression testing in SPL should also be improved for that. (2) *More research on test generation for application-specific nonfunctional aspects is needed*. In the case that application-specific nonfunctional requirements are newly added it is not easy to decide wether we can reuse exisiting test cases or we have to develop new test cases for them.

6. Comparison and analysis results from the maturity of the selected approaches (from P6 through P8): (1) *Approaches should be applied more actively and validated thoroughly in a reproducible form*. Most of the existing approaches were experimental researches that proposed ideas. There are few validated researches except for the ScenTED approach. (2) *Scalability of approaches should be demonstrated*. In the real world product lines the number of variabilities might be huge, but the existing approaches have been validated by using product lines that have a small number of variabilities. (3) *More realistic case studies should be conducted*. Most of the reported case studies are exploratory ones or example applications according to the classification of study types of [38].

# 5. CONCLUSIONS

This paper defined a survey framework that consists of eight SPL-specific testing perspectives and compared and analyzed the contributions of selected works. It also suggested further research opportunities that have been identified through the comparison and analysis.

As the survey in this paper indicates, most of the original researches on SPL testing focused on solving narrow research challenges. Thus, they presented the problems at the detailed level of techniques but could not provide them from the perspective of the whole SPL testing process from initiation through completion. By defining the SPL testing framework and analyzing the contributions of the existing researches this paper suggested as research opportunities those that are not covered in the existing researches such as domain test reference, domain test specification, test data derivation and selection in SPL testing, test selection in application testing, test code reuse, glue code testing, binding mechanism, and application-specific testing.

# 6. ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (20110025946) and by the KAIST High Risk High Return Project (HRHRP).

## 7. REFERENCES


[1] Edwin, O. 2007. Testing in Software Product Lines, Master Dissertation. School of Engineering, Blekinge Institute ofTechnology, Sweden.

[2] Lamancha, B. P. 2009. Software Product Line Testing - A Systemtic Review. In *Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT 2009)*, 23-30.

[3] Pohl, K. and Metzger, A. 2006. Software Product Line Testing. *Communications of the ACM*, 49, 12 (December. 2006).

[4] Tevanlinna, A., Taina, J., and Kauppinen, R. 2004. Product Family Testing – a Survey. *ACM SIGSOFT Software Engineering Notes*, 29, 2, 12–17.

[5] Li, J.H., Li Q., and Li, J. 2008. The W-Model for Testing Software Product Lines. In *Proceedings of International Symposium on Computer Science and Computaitonal Technology*, 690-693.

[6] Muccini, H. and van der Hoek, A. 2003. Towards Testing Product Line Architectures. *Electronic Notes in Theoretical Computer Science* 82, 6.

[7] Pohl, K., Böckle, G., and van der Linden, F. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer, Chaps. 8, 11, and. 18.

[8] McGregor, J. D. 2001. Testing a Software Product Line. CMU/SEI-2001-TR-022.

[9] McGregor, J. D. 2002. Building Reusable Test Assets for a Product Line. In *Proceedings of 7th International Conference on Software Reuse (ICSR)*, LNCS 2319, 345-346.

[10] Engström, E. 2010. Regression Test Selection and Product Line SystemTesting. In *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST)*, 512-515.

[11] Neto, P., Macado, I., Cavalcanti, Y., Almeida, E., Garcia, V., and Meira, S. 2010. A Regression Testing Approach for Software Product Lines Architectures. In *Proceedings of 4th Brazilian Symposium on Software Components, Architectures and Reuse*, 41-50.

[12] Jaaksi, A. 2002. Developing Mobile Browsers in a Product line. *IEEE Software*, July/August, 73-80.

[13] Kakarontzas, G., Stamelos, I., and Katsaros, P. 2008. Product Line Variability with Elastic Components and Test-Driven Development. In *Proceedings of International Conference on Computational Intelligence for Modelling, Contro and Automation (CIMCA)*.

[14] Kamsties, E., Pohl, K., Reis, S., and Reuys, A. 2004. Testing Variabilities in Use Case Models. In *Proceedings of 5th International Workshop on Software Product-Family Engineering (PFE 2003)*, LNCS 3014, 6-18.

[15] Feng, Y., Liu, X., and Kerridge, J. 2007. A Product Line Based Aspect-Oriented Generative Unit Testing Approach to Building Quality Components. In P*roceedings of 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, 24-27.

[16] Ganesan, D., Lindvall, M., McComas, David, D., Bartholomew, M., Slegel, S., and Medina, B. 2010. Architecture-based Unit Testing of the Flight software Product Line. In *Proceedings of the 14th International Software Product Line Conference (SPLC2010)*.

[17] Cohen, M. B., Dwyer, M. B., and Shi, J. 2006. Coverage and Adequacy in Software Product Line Testing. In *Proceedings of the Role of Software Architecture for Testing and Analysis (ROSATEA)*.

[18] Cohen, M. B., Dwyer, M. B., and Shi, J. 2008. Constructing Interaction Test Suites for Highly-Configurable Systems in thePresence of Constraints: A Greedy Approach. *IEEE Transactions On Software Engineering*, 34, 5, (September/October, 2008), 633-650.

[19] Reis, S., Metzger, A., and Pohl, K. 2007. Integration Testing in Software Product Line Engineering: A Model-Based Technique. In *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering (FASE 2007),* LNCS 4422 , 321-335.

[20] Bertolino, A., Fantechi, A., Gnesi, S., and Lami, G. 2006. Product Line Use Cases: Scenario-Based Specification and Testing of Requirements. Chap. 11 of *Software Product Lines: Research Issues in Engineering and Management*, (Eds.) T. Käkölä and J. C. Duenas, Springer.

[21] Nebut, C., Le Traon, T., and Jézéquel, J.-M. 2006. System Testing of Product Lines: From Requirements to Test Cases. *Software Product Lines: Research Issues in Engineering and Management*, (Eds.) T. Käkölä et al., Chap. 12, Springer.

[22] Reuys, A., Reis, S., Kamsties, E., and Pohl, K. 2006. The ScenTED Method for Testing of Software Product Lines. Software Product Lines: Research Issues in Engineering and Management, (Eds.) T. Käkölä et al., Chap. 13, Springer.

[23] Robinson-Mallett, Grochtmann, C., M., Wegener, J., Kohnlein, J., and Kuhn, S. 2010. Modelling Requirements to Support Testing of Product Lines. In *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation(ICST) Workshop*.

[24] Mohamed Ali, M. and Ramadan, M. 2010. An Approach for Requirements BasedSoftware Product Line Testing. In *Proceeding of the 7th International Conference on Infomatics and Systems (INFOS)*.

[25] Uzuncaova, E., Khurshid, S., and Batory, D. 2010. Incremental Test Generation for Software Product Lines. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* 36, 3, 309-322.

[26] Perrouin, G., Oster, S., Sen, S., Klein, J., Baudry, B., and le Traon, Y. 2011. Pairwise Testing for Software Product Lines-A Comparison of Two Approaches. *Software Quality Journal*, Springer, DOI 10.1007/s11219-011-9160-9.

[27] Reuys, A., Pohl, K., Weingartner, J. 2007. Siemens Medical Solutions. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*, F. van der Linden, K. Schmid, E. Rommes, Springer.

[28] Tevanlinna, A. 2004. Product Family Testing with RITA. In *Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques (NWPER'2004)*, 251-265.

[29] Jaring, M., Krikhaar R.L., and Bosch, J. 2008. Modeling Variability and Testability Interaction in Software Product Line Engineering. In *Proceedings of the 7th International Conference on Composition-Based Software Systems*, 120-129.

[30] "Need Fulfillment Qualities", http://www.esi.es/Families/docum/results/Need Fulfillment Qualities.pdf, 2006.

[31] Reis, S., Metzger, A., and Pohl, K. 2006. A Reuse Technique for Performance Testing of Software Product Lines. In *Proccedings of International Workshop on Software Product Line Testing(SPLiT)*.

[32] Lamancha, B.P., Usaola, M.P., and de Guzmán, I.G.R. 2009. Model-Driven Testing in Software Product Lines. In *Proceeding of the 25th IEEE International Conference on Software Maintenance(ICSM2009)*.

[33] Engström, E. 2010. Exlopring Regression Testing and Software Product Line Testing – Researh and State of Practice. Licentiate Thesis, Department of Computer Science, Lund Univ., 11.

[34] Neto, P., Macado, I., MacGregor, J.D., Almeida, E., and Meira, S. 2011. A Systematic Mapping Study of Software Product Line Testing. *Information Software Technology*, doi:10.1016/j.infsof.2010.12.003.

[35] Olimpiew, E. and Gomaa, H. 2005. Model-based Testing for Applications Derived from Software Product Lines. In *Proceedings of the Workshop on Advances in Model-based Testing*.

[36] Olimpiew, E. and Gomaa, H. 2009. Reusable Model-Based Testing. In *Proceedings of 11the International Conference on Software Reuse(ICSR 2009)*, LNCS 5791, 76−85.

[37] Stricker, V., Metzger, A., and Pohl, K. 2010. Avoiding Redundant Testing in Application Engineering. In *Proceedings of Software Product Ling Conference (Software Product Lines: Going Beyond, SPLC2010)*, LNCS 6287, 226-240.

[38] Zannier, C., Melnik, G., Maurer, F. 2006. On the success of empirical studies. In *Proceedings of the International Conference on Software Engineering( ICSE2006)*, 341-350.

[39] Lamancha, B.P. and Usaola, M.P. 2010. Testing Product Generation in Software Product Lines using Pairwise for Features Coverage. In *Proceedings of the 22nd International Conference on Testing Software and Systems*, 111-125.

[40] Hervieu, A., Baudry, B., and Gotlieb, A., "PACOGEN: Automatic Generation of Pairwise Test Configurations from Feature Models", In Proceedings of 22nd IEEE International Symposium on Software Reliability Engineering, pp.120-129, 2011.

[41] ISO/IEC 29119: Software and Systems Engineering – Software Testing – Part 3: Test Documenation. Draft DIS, 2012.

[42] Ghezzi, C. and Sharifloo, A.M. 2011. Verifying Non-Functional Properties of Software Product Lines: Towards and Efficient Approach Using Parametric Model Checking. In *Proceedings of Software Product Ling Conference(SPLC2011)*, 170-175.

[43] Sinha, S., Dasch, T., and Ruf, R. 2011. Governance and Cost Reduction through Multi-tier Preventive Performance Tests in a Large-scale Product Line Development. In *Proceedings of Software Product Ling Conference(SPLC2011)*, 295-302.

[44] Ensan, A., Bagheri, E., Asadi, M., Gasevic, D., and Biletskiy, Y. 2011. Goal-Oriented Test Case Selection and Prioritization for Product Line Feature Models. In P*roceedings of 8th International Conference on Information Technology: New Generations*, 291-298.

[45] Engström, E. and Runeson, P.. 2011. Software Product Line Testing - A Systematic Mapping Study. *Information and Software Technology*, 53, 1, 2-13.

[46] Dueñas, J.C., Mellado, J., Cerón, R., Arciniegas, J.L., Ruiz, J.L., Capilla, R. 2004. Model Driven Testing in Product Family Context. In *Proceedings of the 1st European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, 91-96.

[47] Weingärtner, J. 2002. Product Family Engineering and Testing in the Medical Domain—Validation Aspects, In *Proceedings of Product Family Engineering(PFE2001)*, LNCS 2290, 383-387.

[48] Da Mota Silveira Neto, P. A., Runeson, P., do Carmo Machado, I., de Almeida, E. S., de Lemos Meira, and S. R., Engstrom E. 2011. Testing Software Product Lines. *IEEE Software*, 28, 5, 16-20.