Towards a Formal Framework for Product Line Test Development

Sungwon Kang, Jihyun Lee, Myungchul Kim, Woojin Lee
School of Engineering
Information and Communications University, Korea
{kangsw, puduli, mckim, wjlee}@icu.ac.kr

Abstract

Product line test development is more complicated than the conventional test development for a single application. There were numerous research works in the past that address various issues and aspects that arise in product line testing and test development. However, we still lack a coherent framework that can guide product line test development and link various product line development concepts to relevant product line testing concepts. In this paper we provide a basis for a formal framework for product line test development by linking product line development concepts such as feature, variability, product line architecture, component and use case scenario to product line test concepts such as test architecture, variability for test and test scenario and by providing a systematic way for deriving product line tests, adapting product line tests to a specific product and deriving product specific tests.

1. Introduction

Product line approach to software development is a paradigm for producing a family of related products that maximizes reusability in a systematic way and thereby reduces the overall cost of development. While in the economic respect it promises to return many benefits, at the same time there are many aspects that need be addressed to make the economic expectation come true. First of all, the two phases of development consisting of product development preparation phase called domain engineering and the actual product development phase called application engineering requires related but distinct treatments. Such treatment does not stop at development but should extend also to testing.

Accordingly test development for product should be done in two phases, i.e., in product line development phase and in product development phase and the test cases common for the product line should be reused as the basis for tests for a product. We say ‘basis’ because the product line tests may have to be adapted to make them suitable for the specific product and additional tests specific to the product have to be added. Also we wish such test development to be as much as automatable as possible. Therefore we need systematic ways for (1) deriving product line tests, (2) adapting product line tests to a specific product and (3) deriving product specific tests. Moreover, since in product line approach what distinguishes one product from another is so called features, development and adaptation of tests should be aligned with how the feature set for the product line is defined and structured and how a specific product is specified and developed based on the product platform developed at the product line preparation phase.

This paper provides a formal framework for product line test development that encompasses such aspects. The framework we propose in this paper has the following characteristics: It provides a formal description mechanism for product line features and product features; It provides a formal description notation that when added to an existing architecture description languages can express product line architecture and product architecture. Extended sequence diagram notation is provided to represent use case scenarios and variability and sequence diagram is used as the basis for formal derivation of test scenario given a test architecture. A mechanism for selecting relevant product test scenarios is provided.

The remainder of the paper is organized as follows: In Section 2, test development process for product line is outlined. In Section 3, formal models of the various artifacts necessary in the test development process are defined. Then in Section 4 we exhibit the actual steps of test development. In Section 5, related works are surveyed. Finally, in Section 6 we conclude our paper by discussing the contributions of the paper.

2. Test Development Process

Test development process depends on the software development process. So we need to fix development
process first in order to have concrete discussion on test development.

Most researchers on product line test development based test development on use cases and use case scenarios[6][3][2][7][9]. Since Kang et al. [5] first introduced feature based domain analysis for product line, feature modeling for product line has become the accepted starting point for technical development of product line. So we take the following as the main artifacts for product line development:

1) Feature Model for product line
2) Use cases and use case scenarios representing system requirements
3) Components as implementations of functionality specified by use cases

In this development scheme, a feature maps to a set of use cases [3]. Each use case is elaborated into a set of use case scenarios. In order to do that elaboration, we first need to identify the components or the objects that participate in a particular use case scenario. Since we are mainly interested in the product line test derivation for system testing and integration testing, we will simply regard the entities that appear in use case scenarios as components. A component can have further internal structure and our approach to be developed in the subsequent sections of this paper can be similarly applied to a component and its subcomponents.

The overall test development process we propose is shown in Figure 1. Typical product line engineering process includes realization (or implementation) step but since we focus mainly on system testing and integration testing in this paper, Figure 1 does not include realization step. The development process in Figure 1 shows basically system test development but the process can be easily applied to integration test development as explained in Section 4.1. We use UML sequence diagram to describe use case scenarios but to represent variability at use case scenario level we extend sequence diagram notation. The resulting sequence diagram is called variability extended sequence diagram. Product line (PL) architecture once determined leads to PL test architecture depending on the desired test capability. Then from use case scenarios in the form of sequence diagram combined with PL test architecture PL test scenarios are generated. We classify PL test scenarios based on features. If a certain PL use case scenario contributes to a feature, then the PL test scenario for that PL use case scenario belongs to the test scenarios for that feature so that later when test scenarios for a specific product is needed, all the relevant test scenarios are selected. Overlapping test scenarios can be easily removed by giving a unique name to each PL test scenario. Actual test data can be fixed at the PL test scenario level. Then a product test scenario becomes a product test case.

3. Formal Models of Artifacts for Test Development

3.1. Feature Model for Product Line

In order to develop a product line, all the features relevant for the products in the product line and their relationships should be identified. A model that captures all such information is called a feature model. A feature model contains the set of features common to all products in the family, the set of product-specific features and a set of constraints on how the common and product-specific features can be related.

In order to organize the necessary information, a feature model may introduce the notion of Variation Point (VP) so that product specific feature arrangements can be organized around common features via VPs. So we can think that a product line platform, i.e. the common development base from which the actual product can be conveniently developed, consists of common features and VPs together with the descriptions of the how variation can occur at the VPs.

When \( f \) is a feature and \( \{ f_1, \ldots, f_k \} \) a set of features, the typical feature variabilities at a VP are optional and mandatory variabilities. Products should also satisfy dependencies among features. Optional variability and
the kinds of mandatory variability and feature dependency we will consider are defined as follows:

Optional Variability
- **optional** \( \{f\} \): Either \( f \) is selected or none is selected

Mandatory Variability
- **exclusive** \( \{f_1, \ldots, f_k\} \): Exactly one of \( f_1, \ldots, f_k \) should be selected
- **inclusive** \( \{f_0, \ldots, f_k\} \): Any subset of \( f_0, \ldots, f_k \) can be selected

Dependency
- \( f \) requires \( \{f_1, \ldots, f_k\} \): If \( f \) exists, then all of \( f_1, \ldots, f_k \) should exist
- \( f \) excludes \( \{f_1, \ldots, f_k\} \): If \( f \) exists, then none of \( f_1, \ldots, f_k \) may exist

Given such a notion of variability, how can we formally describe a feature model for a product line? Figure 3 shows a product line description model. In terms of features, a product line is defined as a quadruple consisting of a set of common feature, and a set of optional features, a set of mandatory features and feature dependencies. Since an optional feature can be described via an implicit VP without explicitly introducing its name, VP Specification is not used. Feature can be either an aggregation of other features, called feature group, or a base feature, which cannot be decomposed into other features. Then a specific product is viewed simply as a set of base features.

```
Product_Line = < Common_Features, {optional: Feature_Specification}, Mandatory_Features, {Feature_Dependency} >
Common_Features = {Base_Feature}
Mandatory_Features = {VP_Specification}
VP_Specification = < VP_Name, Variability_Type, {Feature_Specification} >
Variability_Type ::= exclusive | inclusive
Feature_Specification ::= Base_Feature | <optional: Feature_Specification> | VP_Specification
Base_Feature ::= Feature_Name
Feature_Dependencies = < Base_Feature, Dependency_Type, {Feature_Specification} >
Dependency_Type ::= requires | excludes
Product = {Base_Feature}
```

**Fig 2. Product line feature model description**

The description model in Figure 2 defines *Product_Line* and *Product* in terms of base features and VPs. It allows in the description only base feature names and VP names, thereby not allowing names for groups of features. In order to specify feature groups in addition to base features, we can augment the model in Figure 2 with a feature tree similar to the feature model of [5]. Elements of a feature tree are as follows:

```
Mandatory     Optional       Exclusive or
Inclusive or   ---->          ----<
                    Requires        Excludes
```

Then by replacing the edges of a usual tree with the above elements, we can draw feature trees.

### 3.2. Product Line Architecture

The OVM model [8] lets us depict VPs and variants for the VPs together with constraints on them. We utilize the OVM notation for product line architecture description. We mark the box for features with "<<feature>>" stereotype.

In actual product line development the relationship between the set of features and the set of components is, in general, many-to-many. So it is desirable in PL approach that specification and manifesto of each component contain a list of relevant features. Then its implication to product development would be that if any one of the features is included in the product then the associated components should be included in the implementation of the system.

```
Fig 3. Example of inclusive variability to product line architecture
```

![Feature Tree](image)
![Features-to-Components Mapping](image)
![Product Line Architecture](image)
![Product Architecture](image)
Since in general a feature would be realized with a set of components, when a VP is realized with components, in general a set of components for that feature would be involved. So when we move from the feature model to the architecture model, a product line architecture would have one or more VPs left open for realizations. When a specific product is developed, that VP should be replaced with a set of components (together with associations for them).

Let us see how we can describe PL architecture. In this paper, instead of giving a specific PL Architecture Description Language (ADL), we just show how the existing ADLs can be extended to be used as product line ADLs. The outermost box indicates the system boundary. In this case we only consider a fragment of the system and the boundary is the boundary for the fragment. A component is represented with a box. Solid lines indicate associations.

The difference between product line architecture and the architecture for a stand-alone application lies in that the former contains VPs. In the PL architecture of Figure 3(c), there is one VP, which is inclusive. The product architecture is obtained by replacing the VPs in it with the relevant selected components.

3.3. Features, Use Cases and Use Case Scenarios

The components that were involved in one use case scenario may be involved in other use case scenarios. In the special case when a feature maps to multiple use cases but not vice versa, it is straightforward to select test cases for a specific product. Since each feature maps to a set of use cases and a use case maps to a set of use case scenarios, it suffices to simply select use case scenarios for the use cases of the selected feature. Then test scenarios for the use case scenarios become the test scenarios relevant to the feature.

But in the general cases when the mapping is many-to-many, selecting test scenarios relevant to a feature becomes more complicated. So we assume that attached with each use case is a set of features.

3.4. Variability Extended Sequence Diagram

For many applications, use case scenario is a powerful way of elaborating system requirements. Sequence diagram makes use case scenario precise and formal thereby expediting automatic test generation. We present a systematic way to express variability at use case scenario level. UML has “Combined Fragment” construct. The construct can be extended to depict optional, exclusive and inclusive variability. Figure 4 shows how each variability defined in Section 3.1 can be represented in an extended UML sequence diagram notation.

Figure 5 is an example use case scenario represented with this sequence diagram extended with variability description mechanism. The use case scenario of Figure 5 has one VP. Its kind is exclusive and there are two variants. The variant is chosen depending on whether the Customer Class is regular or premium.
test development, multiple test architectures can be used. In the special case of product line development where each feature is realized with a single use case, feature variability (per feature) is completely mapped to variability of a single use case and therefore use case scenarios for that use case, variability of them and test scenarios are straightforwardly linked to features.

However, in the general case when a feature to use cases relation is many to many, e.g. a use case incorporates several features, the features are spread among different use case scenarios. Then individual use case scenarios should reflect potential optionality, inclusion, exclusion and dependencies that will differ from product to product. This will make product line level test scenario structure complicated and also make development of product level test cases complicated.

4.1. Test Architecture

For test development, architecture plays an important role because test scenarios depend on test architectures and the test architecture depends on system architecture. Figure 6 shows a product line test architecture for the product line architecture in Figure 3. In Figure 6, test components TC1, TC2 and TC3 are introduced. A test component has the ability to determine whether the observed phenomena are correct or not and can be either passive testers or active testers. A passive component just observes events or messages but an active tester can both observe and inject them. Test components can communicate with each other. Figure 6 shows one possible test architecture in which TC1 is an active tester and TC2 and TC3 are passive testers. In order to introduce them for actual testing, the internals of other components such as Validation component and service components need be modified.

4.2. Test Scenarios Development

With the test architecture in Figure 6, the sequence diagram in Figure 5 is converted to the test scenario in Figure 7. In the figures, message names are not shown to emphasize the differences from Figure 5.

4.3 Test Scenarios Selection for Product

In the special case that each use case is associated with a single feature, the use case for the feature become either common, optional, inclusive or exclusive since each feature would be either common or have one of the three variabilities. So grouping of use case scenarios for a feature is straight-forward. The remaining consideration is dependency. However, since dependency is also inherited from features to use cases and to the use case scenarios for the use cases, grouping is also straightforward.

In the general case that a use case is associated with more than one feature, a set of base features for the product becomes relevant (to the product). So whether or not a use case is relevant is determined by the presence of the feature in the set of relevant features for the use case. Formally, we select a use case \( U_{Ci} \) for the product \( P \) if and only if the following is true:

\[
\text{Features}_P \cap \text{Relevant_Features}(U_{Ci}) \neq \emptyset
\]  

----(*)

A use case consists of a set of use case scenarios but we can associate each use case scenario with the set of relevant features to select relevant scenarios with finer granularity. Then we select a use case scenario \( U_{CSj} \) for the product \( P \) if and only if the following is true:

\[
\text{Features}_P \cap \text{Relevant_Features}(U_{CSj}) \neq \emptyset
\]  

----(**)

Since there is 1-1 correspondence between use case scenarios and system test scenarios, we select \( TS_j \) which is derived from \( U_{CSj} \), if and only if (** is true. Then actual test cases are derived from the selected test scenarios. In general, more than one test case can be developed for a test scenario.
5. Related Works

McGregor et al. [6] discusses the overall issues for product line testing and assumes that product line test development starts from requirements captured with use case and use case scenarios. Bertolino et al. [2] introduces the use case description mechanisms of PLUC (Product Line Use Case) and PUC (Product Use Case) to extend use case description. The paper also provides a method to derive scenarios to be tested, called PLUTO (Product Lines Use Case Test Optimization). However, the derived scenarios are just the cases to be tested and actual test scenarios have to be developed from them. Nebut et al. [7] proposes a method to generate behaviors specific to a product by describing variability points in use cases, which is similar in purpose to the method of the paper [2]. Both methods address system testing only. Reuys et al. [9] presents ScenTED (Scenario based Test case Derivation) approach. ScenTED uses UML activity diagram to represent all possible use case scenarios for a use case. Reuys et al. [9] does not explicitly prescribe how to describe domain architecture and how to reflect domain architecture to architecture scenarios for integration test development. On the other hand, the approach of modeling all use case scenarios creates a new problem, which is the explosion of the number of variabilities to consider for product test cases. To solve the problem, Kamsties et al. [4] takes the approach of preserving variability into domain test cases and, for deriving domain test cases, proposes different strategies to deal with differently expressed variabilities. Pohl et al. [8] related test case scenario modeled with sequence diagram with orthogonal variability models. But when it derives test case scenarios, it does not consider that various test architectures are possible and should be reflected in the actual test scenarios for it and can affect test verdicts. Also how to extend sequence diagram with variability is not completely prescribed.

Most of the existing researches focused on system testing. ScenTED is the exception in that it also covers integration testing to some extent. However, still it is not completely clear how to apply prescribed techniques in a step-by-step manner to get desired test cases. In general, most of the current researches leave many gaps in traceability and do not precisely link product line development concepts and test development concepts in a systematic way.

6. Conclusion

The contributions of the paper are as follows: Firstly, we proposed a test development process for product line such that various standard development artifacts are linked to test artifacts. Secondly, we showed a way to describe product line architecture by adding VPs to the usual ADLs. Then we argued that product line test architecture should first be obtained from product line architecture and then projected to use case scenarios to produce test scenarios. Incorporating test architecture is important for achieving test automation. Thirdly, we provided a systematic way to extend UML sequence diagram for product line with variability description mechanism. Then we used it to directly derive test scenarios. The variability explosion occurs when use case scenarios are merged to form an activity diagram. At the level of individual use case scenario, usually the number of variabilities is small. With our approach, deriving test case scenarios is simpler and more economical when compared with the ScenTED approach of using activity diagram.

7. Acknowledgments

This work was partially supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract (2007-SW-14-DM-11).

8. References