Use Case-based Testing of Product Lines

Antonia Bertolino
ISTI-CNR
Area della Ricerca di Pisa
via Moruzzi, 1, 56124, Pisa, Italy
+39 050 3152914
antonia.bertolino@isti.cnr.it

Stefania Gnesi
ISTI-CNR
Area della Ricerca di Pisa
via Moruzzi, 1, 56124, Pisa, Italy
+39 050 3152918
stefania.gnesi@isti.cnr.it

ABSTRACT
This paper presents PLUTO, a simple and intuitive methodology to manage the testing process of product lines, described as Product Lines Use Cases (PLUCs). PLUCs are an extension of the well-known Cockburn's Use Cases, a notation based on natural language descriptions of requirements. The proposed test methodology is based on the Category Partition method, and can be used to derive a generic Test Specification for the product line, and a set of relevant test scenarios for a customer specific application.

Categories and Subject Descriptors
D.2.1 **[Software]**: Requirements/Specifications. D.2.5 **[Software]**: Testing and Debugging - *Testing tools (e.g., data generators, coverage testing)*.

General Terms
Verification.

Keywords
Product Lines, Use Cases, Black box testing.

1. INTRODUCTION
World's leading manufacturers of software-intensive systems, such as Philips, Alcatel, Siemens, Nokia, just to name a few, have timely recognized the need to introduce a rational policy of product family production, to manage the proliferation of variants and local customizations of mass marketed products.

Applications sharing similar functionality and user requirements form what is called a *product family* or a *product line* (PL) [6], [5]. The huge patrimony of processes, patterns, models, artifacts, interfaces, etc., pertaining to a general production line of a company, for instance a digital switching system, or a mobile cell phone, constitute the assets of a family, and are organized into a structured repository. To develop new products of the family, components are selected from the repository and integrated.

Differently from conventional single product development, the definition process of a customer specific application hence is influenced not only by the customer requirements, but also by the capabilities of the product line. Consequently, more sophisticated requirement analysis and processing methods are needed.

Following the PL Process Reference Model defined in the CAFÉ project [6], product line development is characterized by two correlated processes: Domain engineering and Application engineering. Domain engineering is the process aiming at developing the general concept of a product line together with all the assets that are common to the whole product line, whereas Application engineering is the process aiming at designing a specific product of the family and eventually produces a customer specific application as in a traditional process.

The most evident and perhaps most urging question in this process is how to handle and represent *variability* [4]. Behind the many commonalities, product family instances in fact necessarily yield variable features, because these constitute precisely what allows for achieving different variants and customized applications. This challenge is reflected in the large attention that specification of product family requirements has drawn in the literature, e.g., [1], [3].

Little attention has been devoted instead to a closely related problem that is *how to test product families*. It is now well recognized that testing takes a predominant amount of development resources and schedule. Therefore, also reuse of test assets is a crucial issue in production processes. In the same manner that a product family specification and design must tackle variability, this need applies for testing as well.

In this view, we propose a simple and quite intuitive methodology to manage the testing process of product lines, based on the requirements expressed in the well-known formalism of Use Cases [2]. In the next section we introduce the notation adopted for expressing PL requirements: we extend Use Cases to PLUCs. In Section 3 we present our test approach called the PLUTO methodology. Conclusions are drawn in Section 4.

2. USE CASES FOR PRODUCT FAMILIES
Use Cases are a powerful tool to capture functional requirements. They provide a means to specify the interaction between a certain software system and its environment and allow for structuring requirements according to the user goals.

An effective and widely used technique for specifying Use Cases was presented by Cockburn in [2]. The technique is based on natural language specification for scenarios and their extensions, which makes requirements documents easy to understand and communicate, even to non-technical people.

In [1] we have presented an extension of Cockburn's Use Cases to deal with specifications of PL requirements, called the PLUCs (Product Line Use Cases). PLUCs allow variations to be described, by explicitly enclosing into the sections of the Use Cases some **tags** that indicate the variable parts of the PL requirements.

More specifically, the tags can represent three kinds of variability: *Alternative, Parametric, and Optional*.

1. **Alternative tags**: they express the possibility to instantiate the requirement by selecting an instance among a predefined set of possible choices. The selection is independent from other variation points;
2. **Parametric tags**: their instantiation is connected to the actual value of a parameter in the requirements for the specific product, each of them depending on the occurrence of a condition;
3. **Optional tags**: their instantiation can be done by selecting indifferently among a set of values, which are optional features for a derived product.

We observe that in PL specifications, some functional requirements bypass the modeling capabilities of the simple formalism of PLUCs and can span across several Use Cases. We could refer to them as cross-cutting features. When a scenario in a PLUC interacts with a scenario in another PLUC, we introduce a textual note like "see UC name" that allows elements of different Use cases to be related with one another.

An example of a PLUC is presented in Figure 1. We propose the description of the GamePlay Use Case applicable to different mobile phones belonging to a same PL. We assume that the products differ at least for the set of games made available to the user and for the provision or not of WAP connectivity.

Curly brackets are introduced into the Use Case elements, and tags (here **[Vo]**, **[V1]**, and **[V2]**) indicate the variation points within the Use Case. Moreover, the possible instantiations of the variable parts and the type of the variations are defined within the Variations section of the PLUC.

3. TESTING OF PRODUCT FAMILIES

In this section we illustrate the methodology that we propose for planning and managing the tests cases of a PL. We call it PLUTO, which stands for Product Line Use Case Test Optimisation.

As identified in [4], the phase in which the majority of variation points are introduced is the requirement specification phase. Accordingly, we believe that planning ahead for testing within the Application engineering process must start from the product requirements: this is why we attack the problem of PL testing based on the requirements specifications.

To develop the PLUTO approach we refer to the Category Partition (CP) method, which is a well-known and quite intuitive black-box test approach proposed in the late eighties [7]. CP was originally conceived as a stepwise methodology to derive a suite of functional tests from the specifications written in structured, semiformal language. We find that the method lends itself quite naturally to application for test derivation from the requirements expressed as Use Cases, but of course it must be extended to tackle PL variabilities.

The first step of the CP method is to analyse the functional requirements to identify the "functional units" that can be separately tested. In the original CP method, a functional unit can be a high-level function or a procedure of the implemented system. In our case, we separately consider each PLUC.

For each functional unit (here PLUC), the tester identifies the environment conditions (the required system properties for a certain functional unit) and the parameters (the explicit inputs for the unit) that are relevant for testing purposes: these are called the *categories*. For each category, the significant (from the tester's viewpoint) values that it can take are then selected, that in CP are called the *choices*. A suite of test cases is obtained by taking all the possible combinations of choices for all the categories.

PL USE CASE GamePlay

Goal: Play a game on a **[Vo]** Mobile Phone and record score

Scope: The **[Vo]** Mobile Phone

Level: Summary

Precondition: The **[Vo]** Mobile Phone is on

Trigger: Function GAMES has been selected from the main menu

Primary actor : The Mobile Phone user

Secondary actors: The {**[V0]** Mobile Phone} (the system)
The Mobile Phone Company

Main success scenario

1. The system displays the list of the {**[V1]** available} games
2. The user selects a game
3. The user selects the difficulty level
4. The user starts the game and plays it until completion
5: The user records the score achieved {**[*V2*]** and sends the score to Club XXX via WAP}

Extensions

1a. No game is available:
1a1. return to main menu
3a. The user starts the game and plays it until an incoming call arrives. See CallAnswer.

Variations

V0: Alternative:
0. Model0
1. Model1
2. Model2

V1: Parametric
if V0=0 then display msg "No game available" else if V0=1 then Snakell or Space Impact else if V0=2 then Snakell or Space Impact or Bumper.

V2: Optional
when V0=2

Figure 1. Example of a Use Case in the PLUC notation

To prevent the construction of redundant, not meaningful or even contradictory combinations of choices, the choices can be annotated with *constraints*, which can be of two types: *i*) either properties or *ii*) special conditions. In the first case, some properties are set for certain choices, and *selector* expressions related with them (in the form of simple IF conditions) are associated with other choices: a choice marked with an IF selector can then be combined only with those choices from other categories that fulfill the related property.

The second type of constraints is useful to reduce the number of test cases: some markings, namely “error” and “single”, are coupled to some choices. The choices marked with “error” and “single” refer to erroneous or special conditions, respectively, that we intend to test, but that have not to be combined with all possible choices.

The list of all the choices identified for each category, with the possible addition of the constraints, is called the **Test Specification**. It is not yet a list of test cases, but contains all the necessary information for instantiating them by unfolding the constraints.

In PLUCs, we use the variation tags similarly to the original notion of CP property constraints, i.e., in the Test Specification we associate the variability tags to the corresponding choices; then, in the process of test derivation we instantiate the tag values so to establish the combinations that are relevant with respect to a customer specific application.

Another characteristic of test cases derived from Use Cases is the presence of several scenarios, i.e., the main success scenario and in addition the possible extensions. Of course all of them must be exercised during testing. Therefore the Test Specification of PLUCs will normally include a category “Scenarios”, in which the main scenario and all the specified extensions are listed.

For illustration purposes, we now apply the Pluto approach to the GamePlay PLUC in Figure 1. As a first step, from an analysis of it we identify, for instance, the following Categories: “Mobile Phone Model”, “Games”, “Difficulty Level”, “Scenarios”, “Club”. Then we proceed with partitioning these categories into the relevant choices, i.e., we single out for each of the categories the values that are the relevant cases to be considered in specific tests. The complete Test Specification is shown in Figure 2.

When applying the CP method to PLs, in general we will have that some of the choices will be available for all the products of the family. On the other hand, some of the categories are specialized into choices that depend on the specific product considered. For instance, the category “Club”, which relates to the capability to exchange the achieved game score with other Club affiliates, is relevant only for those models that support WAP connection. Hence it cannot be tested for any potential applications of the family, but only for those supporting this feature. This is specified in the GamePlay PLUC by means of the **V2** optional tag. Hence, when the test cases are being derived, we make use of this tag similarly to the “constraint” formalism of the CP method. As shown in Figure 2, we derive the possible choices pertaining to the “Club” category, but we annotate them with the appropriate selector, which is a simple IF condition stating that these choices are of interest only when property P2 is satisfied (which happens for Model2).

If we now apply to this test specification a generator which takes out all the possible combinations of choices, we would obtain a list of test cases, which correspond to the whole PL. This list would in fact include all the potential test cases for all the products of the family relative to the PLUC under consideration. However, what is more interesting in our opinion, is that we can instead derive directly the list of tests for a specific product of interest. This is obtained very easily by just instantiating the relative tags. So, for instance, if we are interested to test the Model2 product of this family, we set property P2 to true and derive all and only the combinations that remain valid.

```
PLUC GAMEPLAY TEST SPECIFICATION
[V0]: Mobile Phone Model:
0. Model0                   [Property P0]
1. Model1
2. Model2                   [Property P2]

Games:
None            [if P0]
SnakeII         [if NOT P0]
Space Impact    [if NOT P0]
Bumper          [if NOT P0] [if P2]

Difficulty Level:           [if NOT P0]
easy
medium
expert

Scenarios:
Main                        [if NOT P0]
ext: no game available      [if P0]
ext: a call arrives         see CallAnswer [single]

[V2]: Club:
      WAP connection on             [if P2]
      WAP connection off            [if P2]
```

Figure 2. Main Test Categories for the GamePlay PLUC

As an example, we list below in Figure 3, some of the test cases that would be so obtained for different products, i.e., for different tag assignments. We show these as abstract descriptions and leave to the reader the obvious transformation of these into the corresponding functional test scenarios.

In Figure 3, the test cases Ti, Tj1, Tj2 all refer to a simpler situation in which the features in a PLUC do not depend on the features of another PLUC. Test Tk instead needs further consideration. It considers the choice “a call arrives” of the Scenarios category, which has a specific “see CallAnswer” annotation. This is an example of a cross-cutting feature, whose notion we have introduced in Sect. 2. We now see how this can be handled in the Pluto methodology.

When considering the repository of all Use Cases specified for a PL, it will often be the case that some scenarios in a Use Case depend on other scenarios in another Use Case, because of the presence of cross-cutting features. Referring to the example used so far, let us suppose that the Mobile Phone PL under consideration provides for some applications the capability to save the current status of a game being played in the case that an incoming call arrives. The user may answer or refuse the call. Then, after the communication is closed, the game can be resumed from the status in which it was interrupted.

This case depicts a cross-cutting feature arising from a functional dependency between the GamePlay PLUC and another Use Case that describes the handling of incoming calls (the CallAnswer PLUC, not showed here for size limitations and referred in Figure 1). Also for this Use Case a list of test scenarios could be derived, similarly to what we have done for GamePlay.

Tag V0=0
Ti:
Mobile Phone Model: Model 0
Games: None
Scenarios: ext: no game available

……..

Tag V0=2
Tj1:
Mobile Phone Model: Model 2
Games: SnakeII
Difficulty Level: easy
Scenarios: main
Club: WAP connection on

Tj2:
Mobile Phone Model: Model 2
Games: Bumper
Difficulty Level: expert
Scenarios: main
Club: WAP connection on

……..

Tk:
Mobile Phone Model: Model 1
Games: Space Impact
Difficulty Level: medium
Scenarios: ext: a call arrives - *see CallAnswer*

Figure 3. Some Test Scenarios

It is clear however that the two PLUCs GamePlay and CallAnswer are related with respect to the possibility to interrupt and then retrieve a game play because a call arrives. To identify that a dependency exists, as said, when we elicited the Use Cases we have annotated the related scenario in the GamePlay PLUC with the note "See CallAnswer".

Correspondingly, in the process of deriving the test cases from the GamePlay Test Specification (see Figure 2) the case that a call arrives is contemplated in all those tests in which for the "Scenarios" category the choice "ext: a call arrives" is taken. In Figure 3 the test case Tk for instance selects this choice.

This test is not yet complete: it must be further refined into several related test cases, considering each of the possible combinations of choices offered in its turn by the CallAnswer Test Specification.

More in general, whenever a test specification includes a directive "See another PLUC", the derivation of test cases is made by combining the relevant choices from the two related PLUCs. Note that the annotation is made in the PLUC that triggers the test cases, in our example the GamePlay PLUC.

4. CONCLUSIONS AND FUTURE WORK

The production process in product lines is usually organized with the purpose of maximizing the commonalities of the product family and minimizing the cost of variations. Much work has been done in this direction for the requirement engineering phase of product lines. On the contrary, how to deal with the testing phase of a product line is still a neglected research topic.

In this paper, we have proposed the PLUTO methodology for PL testing, that is inspired by the Category Partition method, but expands it with the capability to handle PL variabilities and to instantiate test cases for a specific customer product. As our approach is based on structured, natural language requirements, the test derivation has to be done partially manually. In particular, the identification of relevant Categories and of the Choices to be tested is left to the tester's judgment, and this is natural. However, lexical and syntactical analyzers for natural language requirements could be used to extract useful information to identify the relevant Categories.

With regard to the derivation of the test cases from the Test Specification, instead, this task can be easily automated, and we are currently working at a PLUTO tool implementation. We plan also to investigate the integration of some of the available lexical/syntactical analyzers in the PLUTO tool to further automate the test generation process.

The work is clearly a first step towards a more comprehensive testing strategy for PLs. On the other hand, the topic of PL testing is complex and relatively new, and therefore this paper is also intended as a contribution to trigger further research.

5. ACKNOWLEDGMENTS

This work was partially supported by the Eureka Σ!2023 Programme, ITEA (ip00004, Project CAFÉ).

6. REFERENCES

[1] Bertolino, A., Fantechi, A., Gnesi, S., Lami, G. and Maccari,, A., Use Case Description of Requirements for Product Lines. REPL'02, Essen, Germany, Avaya Labs Technical Report ALR-2002-033, September 2002.

[2] Cockburn, A., Writing Effective Use Cases. Addison Wesley, 2001.

[3] Halmans, G., and Pohl, K., Communicating the Variability of a Software-Product Family to Customers. Journal of Software and Systems Modeling 2, 1 (2003), 15-36.

[4] Jaring, M., and Bosch, J., Representing Variability in Software Product Lines: A Case Study. In Chastek G. J. (Ed.): Proc. Software Product Lines, 2nd Int. Conf, SPLC 2, San Diego, CA, USA, August 19-22, 2002, LNCS 2379, 15-36.

[5] Jazayeri, M., Ran, A., and van der Linden, F., Software Architecture for Product Families: Principles and Practice. Publishers: Addison-Wesley, Reading, Mass. and London, 1998.

[6] van der Linden, F., Software Product Families in Europe: The ESAPS & Café Projects. IEEE Software (July/August 2002), 41-49.

[7] Ostrand, T.J., and Balcer, M.J., The Category Partition Method For Specifying and Generating Functional Tests. ACM Comm. 31 (6), June 1988, 676-686.