By Klaus Pohl and Andreas Metzger

SOFTWARE PRODUCT LINE TESTING

Exploring principles and potential solutions.

Software product line (SPL) engineering has proven to enable organizations to develop applications with less effort, in shorter time, and with higher quality when compared with the development of single software systems [2, 7, 11]. There are two essential differences between SPL engineering and the development of single software systems (see [7] for details):

- Differentiation between two SPL development processes: In the domain engineering process, the commonalities and the variability of the SPL are defined and the domain artifacts are realized (see Figure 1). In the application engineering process, actual SPL applications are derived from the domain artifacts.
- Explicit definition and management of variability: The central concepts for defining and documenting the variability of a SPL are variation point and variant. A variation point indicates and specifies what can vary, such as the

communications protocol of a mobile phone. A variant defines a concrete variation, for example, the UMTS protocol. In application engineering, the variation points are bound by selecting the variants that satisfy the application-specific requirements. Thereby, SPL applications are derived from the domain artifacts.

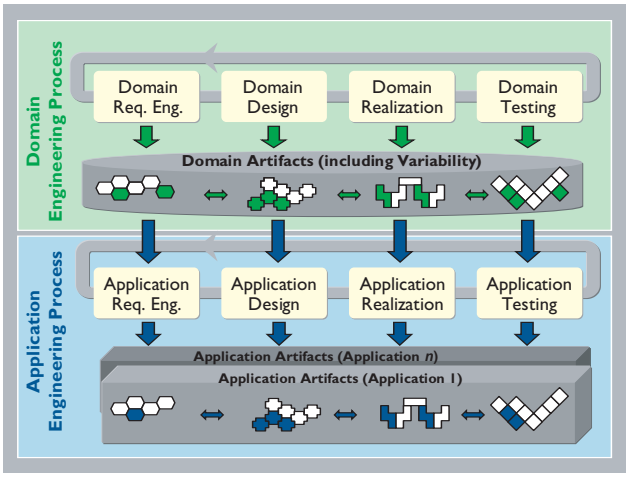

Figure 1. Framework for SPL engineering [7].

Like in the development of single software systems, the aim of testing in SPL engineering is to uncover the evidence of defects. However, the two key differences described here lead to distinct challenges faced in SPL testing (see [3, 9, 10] for details):

Which artifacts should be tested in domain engineering and which ones in application engineering? As testing is part of both product line engineering processes (see Figure 1), an obvious answer to this question would be to test the domain artifacts in domain testing and the application artifacts in application testing. However, due to the variability defined in the domain artifacts, completely testing the domain artifacts in domain testing is impossible except for trivial cases.[1]

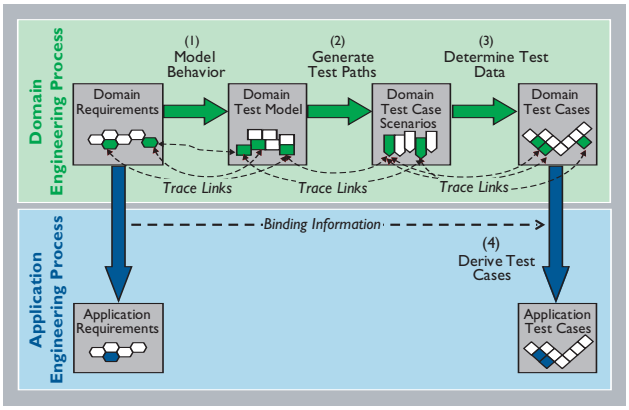

Figure 2. Requirements-based SPL testing with ScenTED.

How to facilitate the reuse of SPL test artifacts? Domain test artifacts (test case designs, test data) should be reused in application testing. But how can we perform such reuse in the presence of variability and application-specific variability bindings? For example, how should the variability binding in the application requirements and the application architecture be taken into account when deriving application test cases from domain test cases?

How to ensure correct variability bindings? Application testing should establish evidence that the binding of the variability in the produced application conforms to the variability defined in the application requirements. For example, testing should establish evidence that a variant that is not included in the application requirements is not accidentally bound in the application.

In this article, we outline six essential principles for SPL system testing that address these challenges and that should be taken into account when developing test techniques for SPL engineering. The principles are based on our experience in SPL testing and the research results established in the European ITEA/Eureka projects ESAPS, CAFÉ, and FAMILIES [5]. Our SPL testing technique, ScenTED, which was successfully applied in industry, demonstrates how we utilized these principles.

Principles for SPL System Testing

P-1: Preserve Variability in Domain Test Artifacts. System tests are performed to evaluate if a system

[1] As an example, if the domain artifacts contain 15 variation points with two variants each (which can be combined without any constraints), approximately one billion possible bindings for these variation points are possible; either none of the variants, one out of two variants, or both variants can be chosen for any variation point, leading to $(1 + 2 + 1)^{15} \approx 10^9$ combinations.

Step	Description
(1)	**Model Behavior:** Based on the behavior defined in domain use cases and use case scenarios, domain test models for the SPL are derived. The domain test models are defined as extended UML activity diagrams. To facilitate the definition of the variability in these test models, we have extended UML activity diagrams by introducing special concepts for modeling variation points and variants (see [9]). The explicit representation of the variability in the domain test model is a prerequisite for the derivation of application test cases (see step (4)).
(2)	**Generate Test Paths:** From the domain test model, domain test case scenarios are generated. The generated test case scenarios again include variability. The generation of the test case scenarios is guided by a test quality criterion. Currently, we use an adapted branch coverage criterion to guide the test scenario generation (see [9]).
(3)	**Determine Test Data:** For the generated domain test case scenarios, the test inputs and expected results are defined. This step is not automated so far. The key obstacle for automation is the lack of detail in the description of the inputs and the domain use cases and use case scenarios.
(4)	**Derive Application Test Cases:** Application test cases are derived by binding the variability in the domain test cases according to the binding of the variability in the application requirements. This derivation is supported by the trace links between the different domain artifacts (see Figure 2). The trace links facilitate the propagation of the binding of the variability defined for a particular application at the requirements level to the test level, that is, the application test cases can be derived from the domain test cases based on the variability bindings in the requirements defined for the application.

Table 1. Key steps of the ScenTED technique (see Figure 2).

complies with its requirements [1, 6]. System test artifacts (including system test cases) should thus be derived from the system requirements. In addition to the domain requirements, the variability defined for the SPL must be taken into account when deriving system test artifacts for SPL applications.

To consider the variability in system testing, we suggest explicitly defining the variability in domain test artifacts and interrelating this variability with the variability defined in the domain requirements. Application test artifacts can then be derived by employing the variability binding of the application requirements to bind the variability defined in the domain test artifacts (see Figure 2).

P-2: Test Commonalities in Domain Engineering. An undiscovered defect in a commonality of a SPL will affect all SPL applications and thus will have a severe effect on the overall quality of the SPL. We therefore suggest testing the commonalities of the SPL as early as possible, ideally in the domain engineering process.

Unfortunately, due to the variability defined in the domain artifacts, typically no executable system, which could be tested, exists in domain engineering. One solution for testing the commonalities is to develop placeholders for the variable parts in the domain artifacts and to define test cases that consider these placeholders. Another solution will be introduced with principle P-4.

P-3: Use Reference Applications to Determine Defects in Frequently Used Variants. If a variant is used in most of the SPL applications, an undiscovered defect in this variant can have a nearly as severe effect on the SPL quality as a defect in a commonality. We thus recommend testing all variants that are likely to be used in many SPL applications as early as possible.

To facilitate the testing of such variants in domain testing, reference applications that contain these variants should be created in parallel to the development of the domain artifacts (see [10]).

Principle	Addressed by ScenTED	Support Offered by ScenTED	Challenges for SPL Testing		
			1. Testing in both SPL Processes	2. Reuse of Test Artifacts	3. Ensure Correct Binding
P-1: Preserve Variability in Domain Test Artifacts	Yes	Variability is explicitly defined in the domain test cases (see Figure 1 and Table 1).		X	
P-2: Test Commonalities in Domain Engineering	Partial	During the derivation of the domain test cases, the ScenTED technique generates test paths without variability as intermediate artifacts. These test paths can be used for testing the commonalities of the product line, once placeholders for variability have been developed.	X	X	
P-3: Use Reference Applications to Determine Defects in Frequently Used Variants P-4: Test Commonalities based on a Reference Application	Yes	The mechanism provided by ScenTED for deriving application test cases can of course be applied to derive test cases for the reference applications.	X	X	
P-5: Test Correct Variability Bindings	Yes	ScenTED explicitly supports the derivation of application test cases used to test the presence and/or the absence of variants in an application (see [3] for details).	X		X
P-6: Reuse Application Test Artifacts across Different Applications	Under Development	We are currently integrating regression testing techniques from the development of single systems into ScenTED. We do this by extending existing regression testing techniques to determine if the SPL variability or an application-specific extension has side effects on potentially reusable test artifacts.	X	X	

Table 2. The realization of the SPL testing principles in ScenTED.

P-4: Test Commonalities based on a Reference Application. If a reference application is used to test frequently used variants, the reference application can also be used to test the commonalities of the SPL. Thereby, the additional effort required to implement placeholders (see P-2) can be reduced.

P-5: Test Correct Variability Bindings. When binding the variation points in the domain artifacts to derive SPL applications, errors can be made. For example, an SPL application could include variants that should not be included in the application. Similarly, a variant can be omitted that should have been bound for the application.

The omission of desired variants can be uncovered by normal system tests. If a variant is missing, the application lacks functionality and thus the system test will fail. However, system tests will typically not uncover the accidental inclusion of a variant.

To uncover the undesired inclusion of a variant, the test engineers must define additional test cases. If one wants to test that a particular variant is not included in the application, a system test case for testing the functionality provided by the variant should be defined. If this test passes, the variant was accidentally bound in the application under test.

P-6: Reuse Application Test Artifacts Across Different Applications. Two or more SPL applications can have the same bindings for one or more variation points and, as a result, these applications will contain a common set of variants. In such a case, the test cases and the test results that consider these variants might be reused across the applications. Therefore, the test effort can be significantly reduced.

However, similar to regression testing for the development of single software systems, the test engineers must ensure that no side effects on the reusable test cases and test results exist. For example, such side effects can be caused by differences in the binding of other variation points and/or application-specific extensions.

The ScenTED Technique

Our ScenTED technique (Scenario-based Test case Derivation) facilitates the systematic, requirements-based derivation of system test cases in SPL engineering. The system test cases are derived from domain requirements, more precisely, from domain use cases enriched with variation points and variants.

Figure 2 depicts the major steps of ScenTED, which are briefly described in Table 1. Table 2 shows how ScenTED employs the principles that have been described in this article to address the challenges for SPL testing. Details about ScenTED can be found in [4, 7, 9].

Early evaluation of the ScenTED technique in industry indicates that ScenTED significantly supports the derivation of system test cases in SPL engineering, as has been confirmed by test engineers and test managers [9]. In addition, our experience supports previous observations that the systematic derivation of test cases from domain requirements leads to requirements specifications of higher quality (see [8]). C

References

1. Bühne, S., Lauenroth, K., and Pohl, K. Modelling requirements variability across product lines. In J.M. Atlee, Ed., *Proceedings of the 13th IEEE International Conference on Requirements Engineering* (Paris, France, Sept. 2005), IEEE Computer Society, 2005, 41–50.
2. Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Reading, MA, 2001.
3. Käkölä, T. and Dueñas, J.C. *Research Issues in Software Product Lines—Engineering and Management*. Springer, Heidelberg, 2006.
4. Kamsties, E., Pohl, K., Reis, S., and Reuys, A. Testing variabilities in use case models. In F. van der Linden, Ed., *Proceedings of the 5th International Workshop on Software Product-Family Engineering, PFE-5* (Siena, Italy, Nov. 2003), Springer, Heidelberg, 6–18.
5. van der Linden, F. Software product families in Europe: The ESAPS and CAFÉ projects. *IEEE Software 19*, 4 (2002), 41–49.
6. Myers, G.J. *The Art of Software Testing, Second Edition*, Revised and updated by Badgett, T. and Thomas, T.M., with Sandler, C., Wiley, Hoboken, NJ, 2004.
7. Pohl, K., Böckle, G., and van der Linden, F. *Software Product Line Engineering—Foundations, Principles, and Techniques*. Springer, Berlin, Heidelberg, New York, 2005.
8. Pretschner, A., Prenninger, S., Wagner, S., et al. One evaluation of model-based testing and its automation. In G.C. Roman, W.G. Griswold, and B. Nuseibeh, Eds., *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)* (St. Louis, MO, May 2005), ACM, 392–401.
9. Reuys, A., Kamsties, E., Pohl, K., and Reis S. Model-based system testing of software product families. In O. Pastor, and J. Falcao e Cunha, Eds., *Proceedings of the 17th Conference on Advanced Information Systems Engineering, CAiSE 2005* (Porto, Portugal, June 2005), Springer, Heidelberg, 519–534.
10. Tevanlinna, A., Taina, J., Kauppinen, R. Product family testing—A survey. *ACM SIGSOFT Software Engineering Notes 29*, 2 (2003), 12–17.
11. Weiss, D.M. and Lai, C.T.R. *Software Product Line Engineering—A Family-Based Software Development Process*. Addison-Wesley, Reading, MA, 1999.

Klaus Pohl (pohl@sse.uni-due.de) is the scientific director of Lero (The Irish Software Engineering Research Center) and a professor for software systems engineering at the University of Duisburg-Essen, Germany and for software engineering at the University of Limerick, Ireland.

Andreas Metzger (metzger@sse.uni-due.de) is a senior research assistant in the Software Systems Engineering group at the University of Duisburg-Essen in Germany.

This research was supported by Science Foundation Ireland under the CSET grant 03/CE2/I303_1.