SOFTWARE PRODUCT LINE TESTING
A Systematic Review

Beatriz Pérez Lamancha
Software Testing Centre (CES), School of Engineering
University of the Republic,Montevideo, Uruguay

Macario Polo Usaola, Mario Piattini Velthius
Alarcos Research Group, Information Systems and Technologies Department
University of Castilla-La Mancha, Ciudad Real, Spain

Keywords: Testing, Software product line, Systematic review.

Abstract: Software product lines constitute a new paradigm where industrial production techniques are adapted and applied to software development. Reuse and the maintenance of traceability between the different artefacts in the line are fundamental requirements in this paradigm, articulating the best practices for software development in an environment that is perfectly controlled by software engineering methods. This article presents a systematic review of the literature which deals with testing in software product lines. The objective is to analyse the existing approaches to testing in software product lines, discussing the significant issues related to this area of knowledge and providing an up-to-date state of the art which can serve as a basis for innovative research activities. The paper also analyses how SPL research can contribute and dynamise the research in software testing.

1 INTRODUCTION

A software product line (SPL) is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way (Clements and Northrop, 2002). In Europe, the term product family (PF) or system family (SF) is used to refer to an SPL (van der Linden, 2002). In software engineering, SPL represents a recent development paradigm, in which the reuse is proactive and predictive and not opportunistic as with classic development (in which, typically, the software is first constructed and encapsulated and then reuse is considered) (McGregor et al., 2002; Krueger, 2006). SPL development requires the intensive use of models, processes, automated support, etc., all with the goal of having the SPL investment (which will be high) and its corresponding products (which will be lower than classic development) recompense the individual development of each product. In SPL, the best practices and techniques in Software Engineering will be articulated and applied. The research and state of the art must be improved and integrated in this context.

Recently, Bertolino (Bertolino, 2007) presented a general analysis of the state of the art in testing research which serves as a roadmap for the most relevant challenges. This work begins with some important past achievements, while its destination consists of four identified goals which research tends, but which remain unreachable. She calls these dreams. The routes from achievements to dreams are paved by outstanding research challenges. The four dreams are: universal test theory, test-based modelling, 100 percent automatic testing and efficacy-maximized test engineering. She also distinguishes the transversal challenges that run through all four of the identified dreams. One of them is testing within the emerging development paradigm, in which the software product line can be categorised.

This work presents a systematic review of the literature (Kitchenham, 2004) which deals with testing in software product lines. Our objective is to analyse the existing approaches to testing in software product lines, discussing the significant issues related to

Pérez Lamancha B., Polo Usaola M. and Piattini Velthius M. (2009).
SOFTWARE PRODUCT LINE TESTING - A Systematic Review.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 23-30
DOI: 10.5220/0002248400230030

this area of knowledge and providing an up-to-date state of the art that can serve as a basis for innovative research activities. As mentioned earlier, SPL articulates and applies the best practises and techniques. This work also analyses whether SPL can help to achieve the challenges defined by Bertolino to reach the four dreams in testing research.

The paper proceeds as follows: Section 2 presents the steps followed to do the systematic review, showing the inclusion and exclusion criteria. Section 3 categorises and summarises the primary studies found following the systematic review. Section 4 studies the way in which the studies found in software product line testing help to achieve the testing challenges defined by Bertolino. Finally, the conclusions are outlined and future work is described.

2 SYSTEMATIC REVIEW

A systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, topic area or phenomenon of interest (Kitchenham, 2004). We followed the guidelines defined by Kitchenham (Kitchenham, 2004), the template defined by Biolchini et al. (Biolchini et al., 2005) and the procedure defined by Pino et al. (Pino et al., 2007) for the primary studies selection (there are the individual studies contributing to a systematic review, which is a form of secondary study). The goal of this systematic review is to identify experience reports and initiatives carried out in Software Engineering related to testing in software product lines. The question is: Which initiatives have been carried out to deal with testing in the Software Product Lines? The keywords identified and their synonyms are the following:

- Software Product Line: product line, software product lines, product families, product family, software family, system families
- Testing: test

With these keywords, the search string defined was:
((”product line” OR ”software product lines” OR ”product families” OR ”product family” OR ”software family” OR ”system families”) **AND** (”testing” OR ”test”)**)**

Only studies written in English were included. The source search method was to research each of the selected sources using web search engines. The selected source list of electronic databases is:

- SCOPUS document database (http://www.scopus.com),
- Science@Direct on the subject of Computer Science (http://www.sciencedirect.com),
- Wiley InterScience on the subject of Computer Science (http://www.interscience.wiley.com),
- IEEE Digital Library (http://www.computer.org)
- ACM Digital Library (http://portal.acm.org)

In addition to the source list, we hand-searched the following projects and web pages:

- Engineering Software Architectures, Processes and Platforms for System-Families - ESAPS (http://www.esi.es/esaps/)
- From Concepts to Application in System-Family Engineering - CAFE (http://www.esi.es/Cafe)
- FAct-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering - FAMILIES (http://www.esi.es/Families)
- RITA - Environment for Testing Framework-based Software Product Families (http://www.cs.helsinki.fi/group/rita)
- Franhoufer IESE, PuLSE (http://www.iese.fraunhofer.de/)
- Software Engineering Institute, Product Line Systems Program (http://www.sei.cmu.edu/programs/pls)

The inclusion criterion was based on the review of the title, abstracts and keywords of the articles identified in the search. All the studies related to the research topic were selected except editorials, prefaces, article summaries and summaries of tutorials, workshops, panels and poster sessions. The procedures for the definition of the studies is the following: the search strings must be executed in the selected sources. To select an initial set of studies, the abstract of everything obtained from web search engines is read and evaluated according to the exclusion and inclusion criteria. To refine this initial set of studies, their full text is read. The search strings were adapted to the search engine for each source. Due to the lack of standardisation between the electronic resources, we had to implement the search strategy individually for each database. We applied the search terms to the titles, abstracts and keywords of the articles in the identified electronic databases. In some database, we were not allowed to restrict the search to those fields. In that case, we searched all the fields.

The results of the execution of the search strings in each of the source databases is **23** primary studies. All selected studies were assumed to be of high quality, given that they were published in the selected sources.

This means that the accepted publications have gone through a strict review procedure which guarantees their quality.

3 SOFTWARE PRODUCT LINE TESTING

This section summarises the information extracted in the systematic review and presents the state of the art relating to testing in software product lines. The primary studies were categorised in:

- Unit Testing (UT)
- Integration Testing (IT)
- Functional Testing (FT)
- SPL Architecture Testing (AT)
- Embedded systems Testing (ET)
- Testing Process (TP)
- Testing effort in SPL (TE)

Once the primary studies were chosen, extraction of the relevant information for the systematic review was carried out. The extracted information is summarised in Table 1 which shows - for each paper - the category, how variability is dealt with, the testing technique used, if there is a tool or prototype to support it, if an example is shown, if the proposal was tested in an artificial setting and if the proposal was tested in an industrial setting.

A huge interest in SPL testing level is seen in the papers found. In particular, functional testing and variability testing with UML models and use cases are the most popular topics. Several proposals exist for test automation but they are generally prototypes. Overall, there are very few experiments or case studies documented that put theoretical proposals into practise. The studies relating to testing processes in SPL are guidelines, but there is no testing process defined for SPL. In particular, topics like testing planning, effort and management are not included in the existing literature.

In the following, a state of the art summary of software product lines is presented, organised by the categories defined above.

3.1 Unit Testing

There are no specific techniques for unit testing, only recommendations which have not been tested in artificial or industrial environments. McGregor (McGregor, 2001) defines unit-test plans, test cases and reports that become part of the documentation that accompanies a core asset. The software units must be tested when the core assets are created.

3.2 Integration Testing

For integration testing McGregor (McGregor, 2001) proposes two techniques that can be used to mitigate the problem of testing the integration of all possible combinations of all variations. Combinatorial test designs can be used to greatly reduce the number of combinations that need to be tested, while the other technique is to perform integration testing incrementally, in which case the number of combinations that must be covered is much smaller. The RITA Tool, presented in (Tevanlinna, 2004), is a prototype which supports integration testing of product families and frameworks. RITA works with the commonality and variability in product families and the polymorphism in framework based software. In (Kauppinen et al., 2004), two criteria for framework-based product families are defined based on traditional structural coverages.

3.3 Functional Testing

The largest number of works were found for functional testing. Several authors define techniques for test case derivation in SPL, most of them from use cases modified to represent variability in the line. This is the case with Bertolino et al. (Bertolino et al., 2004), McGregor (McGregor, 2001), Nebut et al. (Nebut et al., 2004), Olimpiew el al.(Olimpiew and Gomaa, 2005) and Reuys et al. (Reuys et al., 2005). Only the last work was applied in an industrial environment.

Bertolino et al. (Bertolino et al., 2004) adapt the use cases to SPL (PLUCs - Product Line Use Cases) in which the variations are described explicitly with tags. The test cases are derived manually using the Category Partition method. This methodology is called PLUTO (Product Line Use Case Test Optimisation). Nebut et al. (Nebut et al., 2004) have a very similar proposal, they propose associating contracts to the SPL use cases to express the dependencies adding UML tagged values and the pre and post conditions are UML notes expressed in first-order logic. They present a tool to automate the test case derivation.

McGregor (McGregor, 2001) creates generic test cases from the use-case scenarios and then specific test cases are derived for the product. The variability combination is resolved with the orthogonal arrays technique.

Olimpiew el al. (Olimpiew and Gomaa, 2005) create test models from use cases. They first create activ-

ity diagrams from the use cases, then create decision tables from the activity diagrams and finally, create test templates from the decision tables and from the path in the activity diagram. Test data is generated to satisfy the execution conditions of each test template.

Reuys et al. (Reuys et al., 2005) obtain the test cases from the use cases, using activity diagrams to define all the use case scenarios. Test case scenarios are specified in sequence diagrams without specifying concrete test data. Later, they are refined to include detailed test inputs, expected results, additional information and test scripts relevant to this scenario. This technique is called ScenTED (Scenario based TEst case Derivation) and has been applied in a case study at Siemens AG Medical Solutions HS IM. They also present a tool to support this method.

Related to test case derivation from sequence diagrams, Nebut et al. (Nebut et al., 2003) propose a method supported by a toolset, in which behavioural test patterns (behTP) are obtained from high-level sequences which are used to automatically generate test cases specific to each product. Each behTP represents a high-level view of some scenarios which the system under test may engage in. Another work by Kang et al. (Kang et al., 2007) extends sequence diagram notation to represent variability in use case scenarios. From a test architecture defined using the Orthogonal Variability Model (OVM)(Pohl et al., 2005) and sequence diagrams, test scenarios are derived. The Orthogonal Variability Model (OVM) is also used by Cohen et al. (Cohen et al., 2006) to define a coverage test model for SPL which takes into account the coverage of variability combinations.

Two works propose to define testing meta-models. Dueñas et al. (Dueñas et al., 2004) define a metamodel to represent the variability in scenarios and test cases and use state transition diagrams to model the test case behaviour. The variability is defined by an algebraic expression. Baerisch (Baerisch, 2007) defines two models to represent SUTs and the tests executed on the SUTs: test models, which express the intention of the test, represent the domain-specific requirements that are verified by the test and system models which describe the SUT and must include information about the features and interfaces that are relevant for the definition and the execution of tests.

For functional testing automation, Ardis et al. (Ardis et al., 2000) present a case study at Bell Labs. There three test strategies suitable for general use in product line testing automation based on a design for testability were developed. They require an architecture that minimises the cost of both the generation and the testing of each family member.

For legacy systems, Geppert et al. (Geppert et al., 2004) obtained a family of generic test cases by generalising existing (or new) test cases driven by the parameters of variation of the commonality analysis. They use a decision tree to generate the specific test cases.

An interesting result is the controlled experiment described by Denger et al. (Denger and Kolb, 2006) to investigate the effectiveness and efficiency of code inspection and functional testing by detecting different defect types in the common and variable parts of reusable code components in the context of product line development. They conclude that code inspections and functional testing find different types of defects with different effectiveness and efficiency. The results indicate that standard techniques are not sufficient to address variant-specific defects.

3.4 SPL Architecture Testing

To improve the testability, Kolb et al. (Kolb and Muthig, 2006) propose designing the product line architecture in such a way that it supports the testing of the reusable product line components and the different products, considering testing in the architecture design phase.

3.5 Embedded System Testing

The works found for testing in embedded systems are generally specific to a particular domain and have been tested in artificial environments or industries. Kim et al. (Kim et al., 2006) present a tool that supports the development of SPL for embedded systems of control, using FORM (Feature-Oriented Reuse Method) and using simulation for testing. Kishi et al. (Kishi and Noda, 2006) apply formal verification techniques (model checking) to verify the design of embedded systems in SPL. They use Gomaa et al.'s proposal to represent the variability in UML models and present a tool that supports this approach. Pesonen et al. (Pesonen et al., 2006) use aspects to implement specialisations at the core assets level in embedded systems for smoke testing the devices. They present experiment results with the Symbian OS. Trew (Trew, 2005) presents an analysis of the causes of the problems reported in a Philips television SPL and defines a set of rules to use from this.

3.6 Testing Process

With respect to the testing process, there is no defined and proven methodology. The existing proposals are guidelines for testing activities. For McGregor (McGregor, 2001), testing in the context of a product

Table 1: Primary studies summary.

Paper	Category	Variability	Technique	Tool	Example	Artificial setting	Industrial setting
(Ajila and Dumitrescu, 2007)	TE		Change in SPL	NO	NO	NO	YES
(Ardis et al., 2000)	FT		Scenarios, Drivers	NO	YES	NO	YES
(Baerisch, 2007)	FT	System metamodel	Testing metamodel	NO	NO	NO	NO
(Bertolino et al., 2004)	FT	Use Cases	Category Partition	NO	YES	NO	NO
(Clements and Northrop, 2007)	FT			NO	YES	NO	NO
(Cohen et al., 2006)	FT	Orthogonal Var. Model	Coverage and adequacy metrics	NO	YES	NO	NO
(Denger and Kolb, 2006)	FT		Inspections, functional testing	NO	NO	YES	NO
(Dueñas et al., 2004)	FT	Classes	Testing metamodel, state chart	NO	NO	NO	NO
(Geppert et al., 2004)	FT	Legacy systems	Decision tree	NO	YES	NO	YES
(Kang et al., 2007)	FT,TA	Sequence diag.	OVM in Architecture, Sequence diag.	NO	YES	NO	NO
(Kauppinen et al., 2004)	IT		Coverage metrics	NO	YES	NO	NO
(Kim et al., 2006)	ET	FORM	Simulation	YES	YES	NO	NO
(Kishi and Noda, 2006)	ET	UML models	Model checking	YES	NO	YES	NO
(Kolb and Muthig, 2006)	AT	SW Architecture	testability model	NO	NO	NO	NO
(McGregor, 2001)	FT, IT, UT, TP	Use Cases	Combinatory design testing	NO	YES	NO	NO
(Nebut et al., 2003)	FT	Use Cases, Sequence diag.	Sequence diag.	YES	NO	YES	NO
(Nebut et al., 2004)	FT	Use Cases	Use Cases	YES	NO	YES	NO
(Olimpiew and Gomaa, 2005)	FT	Use Cases, Activity diag.	Decision tables	NO	NO	NO	YES
(Pesonen et al., 2006)	ET	Aspects	Smoke test	NO	YES	NO	YES
(Pohl and Metzger, 2006)	FT, TP	Use Cases	Activity diag.	NO	NO	NO	NO
(Reuys et al., 2005)	FT	Use Cases, Activity diag.	Sequence diag.	YES	YES	NO	YES
(Tevanlinna, 2004)	IT			YES	NO	NO	NO
(Trew, 2005)	ET			NO	YES	NO	YES

line includes testing the core assets, the product specific software and their interactions. Pohl et al. (Pohl and Metzger, 2006) outline six essential principles for SPL system testing that should be taken into account when developing test techniques for SPL engineering. These principles are:

(1) Preserve variability in domain test artefacts, (2) Test commonalities in domain engineering, (3) Use reference applications to determine defects in frequently used variants, (4) Test commonalities based on a reference application, (5) Test correct variability bindings and (6) Reuse application test artefacts across different applications. The SEIs Framework (Clements and Northrop, 2007) proposes a separate framework for testing in SPL, defining the following guidelines for this testing: Structure the set of testing processes to test each artefact as early as possible, structure test artefacts to accommodate the product line variation, maintain the test artefacts, structure the testing software for traceability from the test code itself to the code being tested, reuse product line assets for system integration testing and automate regression testing.

3.7 Testing Effort

Ajila et al. (Ajila and Dumitrescu, 2007) make a study of the changes in the product line architecture of a large telecommunications equipment supplier. They conclude that code size is not a good predictor of testing effort at either product or product line levels and that testing effort does not seem to depend on the product's target market.

4 SOFTWARE TESTING RESEARCH AND SPL

Bertolino(Bertolino, 2007) presents the achievements, challenges and dreams in testing research. The software product lines can be categorised in the transversal challenge: testing within the emerging development paradigms. Below, the challenges are described.

- Explicit test hypotheses: Make explicit the test practise behind the selection of every finite test set, by which a sample is taken as the representative of several possible executions
- Test effectiveness: Provide analytical, statistical, or empirical evidence of the effectiveness of the test-selection criteria in revealing faults, in order to understand the classes of faults for which the criteria are useful.
- Empirical body of evidence: Produce an empirical body of knowledge which is the basis for building and evolving the theory for testing.
- Compositional testing: Understand how we can reuse the test results observed in the unit testing, and what conclusions can be inferred about the system resulting from the composition, and which additional test cases must be run on the integration.
- Model-based testing: How can we combine different styles of modelling, and the need for ways to combine model-based criteria with other approaches.
- Anti-model-based testing: Parallel to model-based testing, several efforts are being devoted to novel forms of testing which lay directly on the analysis of program executions.
- Test oracles: Find more efficient methods for realising and automating oracles.
- Test input generation: Automatic generation of test inputs.
- Domain-specific test approaches: Extend domain specific approaches to the testing stage, and in particular to find domain-specific methods and tools to push test automation.
- On-line testing: Monitoring a system's behaviour in real life operation, using dynamic analysis and self-test techniques.
- Controlling evolution: Strategies to scale up regression testing to large composite systems.
- Leveraging user population and resources: Augment in-house quality assurance activities by using data dynamically collected from the field.
- Testing patterns: Patterns offer well-proven solutions to recurring problems, making explicit the successful procedures, which is highly desirable.
- Understanding the costs of testing: Incorporate estimation functions of the cost/effectiveness ratio of available test techniques.

Table 2 enumerates the challenges in testing to achieve the dreams as defined by Bertolino (Bertolino, 2007) and the challenges to enriching the state of art in testing. The table was completed with the studies found in the systematic review presented here and several challenges are addressed today in SPL testing. Especially, model based testing and test input generation seem to be fundamental for testing in SPL. Unfortunately, other challenges are not covered for SPL today. The state of the art in SPL testing is still not mature, mainly in test oracle definition and testing management topics, like testing effort or testing costs in the SPL context and in general, there is a lack of empirical results in the existent works.

5 CONCLUSIONS

This paper has presented an analysis of the current state of the art in software product lines testing, linking them with recent research in software testing. In general, SPL in Software Engineering is a young discipline, but a very promising one, proving that most of the results and benefits obtained from SPL can be extrapolated to other methodologies or development paradigms. In the case of testing, Bertolino (Bertolino, 2007) has pointed out a transversal challenge to the development of testing techniques and their reuse from emerging paradigms, as product lines may well be. Among the other research lines identified by this author is a tendency towards the use of models for the design of systems and the derivation of test cases from them, which had also been noted as a necessity by Harrold (Harrold, 2000) nearly a decade ago. Another critical research line for SPL is test automation and re-execution.

In our case, we are implementing a model-driven development approach to the design of an experimental software product line as a basis to automate the generation of test cases using model-based testing techniques adapted to software product lines. In the proposal, we are interested in the automated derivation of test cases from design models using QVT transformations (OMG, 2008), all while paying special consideration to the reuse of the definition of the oracle, which is the most costly in the test case generation (Bertolino, 2007; Baresi and Young, 2001).

Table 2: Software product line testing studies according to the dreams and challenges of testing.

Dream: Universal test theory			
Explicit test hypotheses	Test effectiveness	Compositional testing	Empirical body of evidence
(Kolb and Muthig, 2006), (Ardis et al., 2000)	(Denger and Kolb, 2006)	(McGregor, 2001), (Geppert et al., 2004)	(Reuys et al., 2005), (Denger and Kolb, 2006)
Dream: Test-based modeling			
Test oracles	Model-based testing		Anti-model-based testing
	(McGregor, 2001),(Nebut et al., 2003),(Nebut et al., 2004),(Bertolino et al., 2004), (Kang et al., 2007),(Olimpiew and Gomaa, 2005),(Dueñas et al., 2004), (Reuys et al., 2005),(White and Schmidt, 2006), (Baerisch, 2007)		
Dream: 100 percent automatic testing			
On-line Testing	Test input generation		Domain-specific test approaches
(Kishi and Noda, 2006)	(Nebut et al., 2003),(Nebut et al., 2004), (Reuys et al., 2005),(Ardis et al., 2000),(Tevanlinna, 2004), (Dueñas et al., 2004)		(Pesonen et al., 2006), (Kim et al., 2006),(Kishi and Noda, 2006),(Trew, 2005)
Dream: Efficacy-maximized test engineering			
Understanding the costs of testing	Controlling evolution	Leveraging user population and resources	Testing patterns
(Ajila and Dumitrescu, 2007),(Cohen et al., 2006)			

ACKNOWLEDGEMENTS

This research is financed by the projects: PRALIN (PAC08-0121-1374) and MECCA (PII2I09-00758394),"Cons. de Ciencia y Tecnol. de la Junta de Comunidades de Castilla-La Mancha".

REFERENCES

Ajila, S. and Dumitrescu, R. (2007). Experimental use of code delta, code churn, and rate of change to understand software product line evolution. *The Journal of Systems and Software*, 80(1):74–91.

Ardis, M., Daley, N., Hoffman, D., Siy, H., and Weiss, D. (2000). Software product lines: a case study. *Software Practice and Experience*, 30(7):825–847.

Baerisch, S. (2007). Model-driven test-case construction. *Foundations of Software Engineering*, pages 587–590.

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *Internation Conference on Software Engineering*, pages 85–103. IEEE Computer Society.

Bertolino, A., Gnesi, S., and di Pisa, A. (2004). Pluto: A test methodology for product families. *Software Product-family Engineering: 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003: Revised Papers*.

Biolchini, J., Mian, P., Natali, A., and Travassos, G. (2005). Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, 679(05).

Clements, P. and Northrop, L. (2007). A framework for software product line practice, version 5.0.

Clements, P. C. and Northrop, L. M. (2002). Salion, inc.: A software product line case study. Technical Report CMU/SEI-2002-TR-038.

Cohen, M., Dwyer, M., and Shi, J. (2006). Coverage and adequacy in software product line testing. *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 53–63.

Denger, C. and Kolb, R. (2006). Testing and inspecting reusable product line components: first empirical results. *Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 184–193.

Dueñas, J., Mellado, J., Cern, J., Arciniegas, J., Ruiz, J., and Capilla, R. (2004). Model driven testing in product family context. Technical Report ISSN 1381 - 3625, University of Twente.

Geppert, B., Li, J., RoBler, F., and Weiss, D. (2004). Towards generating acceptance tests for product lines. *Software Reuse: 8th International Conference, ICSR 2004, Madrid, Spain, July 5-9, 2004: Proceedings*.

Kang, S., Lee, J., Kim, M., and Lee, W. (2007). Towards a formal framework for product line test development. *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 921–926.

Kauppinen, R., Taina, J., and Tevanlinna, A. (2004). Hook and template coverage criteria for testing framework-based software product families. *Proceedings of the International Workshop on Software Product Line Testing*, pages 7–12.

Kim, K., Kim, H., Ahn, M., Seo, M., Chang, Y., and Kang, K. (2006). Asadal: a tool system for co-development of software and test environment based on product line engineering. *International Conference on Software Engineering*, pages 783–786.

Kishi, T. and Noda, N. (2006). Formal verification and software product lines. *Communications of the ACM*, 49(12):73–77.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele University, UK, Technical Report TR/SE-0401-ISSN*, pages 1353–7776.

Kolb, R. and Muthig, D. (2006). Making testing product lines more efficient by improving the testability of product line architectures. *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 22–27.

Krueger, C. (2006). The emerging practice of software product line development. *Military Embedded Systems*, (2nd semester):34–36.

McGregor, J. (2001). *Testing a Software Product Line*. Carnegie Mellon University, Software Engineering Institute.

McGregor, J., Northrop, L., Jarrad, S., and K, K. P. (2002). Initiating software product lines. *IEEE Software*, 19(4):24–27.

Nebut, C., Fleurey, F., Le Traon, Y., Jezequel, J., and de Beaulieu, C. (2004). A requirement-based approach to test product families. *Software Product-family Engineering: 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003: Revised Papers*.

Nebut, C., Pickin, S., Le Traon, Y., and Jezequel, J. (2003). Automated requirements-based generation of test cases for product families. *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 263–266.

Olimpiew, E. and Gomaa, H. (2005). Model-based testing for applications derived from software product lines. *Proceedings of the 2005 workshop on Advances in model-based testing*, pages 1–7.

OMG (2008). Meta object facility (mof) 2.0 query/view/transformation, v1.0. Technical Report formal/2008-04-03.

Pesonen, J., Katara, M., and Mikkonen, T. (2006). Production-testing of embedded systems with aspects. *LECTURE NOTES IN COMPUTER SCIENCE*, 3875:90.

Pino, F., Garcia, F., and Piattini, M. (2007). Software process improvement in small and medium software enterprises: A systematic review. *Software Quality Journal*.

Pohl, K., Bckle, G., and van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer.

Pohl, K. and Metzger, A. (2006). Software product line testing. *Communications of the ACM*, 49(12):78–81.

Reuys, A., Kamsties, E., Pohl, K., and Reis, S. (2005). Model-based system testing of software product families. *Pastor, O.; Falcao e Cunha, J.(Eds.): Advanced Information Systems Engineering, CAiSE*, pages 519–534.

Tevanlinna, A. (2004). Product family testing with rita. *Proceedings of the Eleventh Nordic Workshop on Programming and Software Development Tools and Techniques (NWPER'2004)*, pages 251–265.

Trew, T. (2005). Enabling the smooth integration of core assets: Defining and packaging architectural rules for a family of embedded products. *LECTURE NOTES IN COMPUTER SCIENCE*, 3714:137.

van der Linden, F. (2002). Software product families in europe: the esaps and cafe projects. *Software, IEEE*, 19(4):41–49. TY - JOUR.

White, J. and Schmidt, D. (2006). Fireant: A tool for reducing enterprise product line architecture deployment, configuration, and testing costs. *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*, pages 507–508.