Standing on the Shoulders of Software Product Line Research for Testing Systems of Systems

Antonia Bertolino
ISTI–CNR
Pisa, Italy
antonia.bertolino@isti.cnr.it

Francesca Lonetti
ISTI–CNR
Pisa, Italy
francesca.lonetti@isti.cnr.it

Vânia de Oliveira Neves
Universidade Federal Fluminense
Niterói, Brazil
vania@ic.uff.br

***Abstract*—The complex and dynamic nature of Systems of Systems (SoSs) poses many challenges on their validation and testing, but so far few effective test strategies exist to address them. On the other hand, extensive research has been conducted in the testing of Software Product Lines (SPLs), which present interesting convergence points with SoSs, as both disciplines aim at reducing development costs and time-to-market thanks to extensive reuse of existing artifacts. In this paper, we outline commonalities and differences between the SoS and SPL paradigms from the point of view of testing and investigate how existing methods and tools from SPL testing could be leveraged to address the challenges of SoS testing.**

***Index Terms*—System of Systems; Software Product Line; Testing**

I. Introduction

More than a decade ago, the highly-cited FOSE roadmap of software testing research [1] listed among a few transversal challenges the need to identify proper testing approaches that could be applicable within the next emerging development paradigm. This challenge perpetually continues to trouble software testing researchers as new paradigms of development catch on that to pose conceptual and technical issues and demand appropriate testing methods and tools. The challenge is further exacerbated by the fact that usually at each new paradigm launch, the cost and complexity of testing activities grow [2]. In fact, the new software development processes are generally conceived as the means for achieving higher flexibility, dynamism, and adaptation within faster time-to-market, though without reducing the expectations in terms of quality and reliability requirements.

Among relatively new paradigms, a System of Systems (SoS) [3] is developed by combining previously existing *constituent systems*, aiming at achieving some global goal, or *mission*, which is beyond their individual scopes. Although the early literature on SoSs can be dated back to the 90's [4], not much research on SoS testing has yet been carried out. Concerning SoS testing, though, do we actually need to invent new strategies, as previously asked by the authors of [5]? Or, can we maybe leverage existing methods and tools from other paradigms?

Along with such questions, in this paper we examine the literature produced by software testing researchers in the Software Product Line (SPL) area, who in the last twenty years have proposed a rich collection of approaches. We believe, in fact, that –after pointing out the needed distinctions between respective goals and concepts– there exist quite interesting convergences.

Aiming at reducing development costs and time-to-market, while increasing quality, both SPL and SoS paradigms are strongly centered on reuse of existing artifacts: in SPL, the components are customised and integrated into a specific product exhibiting the desired features; in SoS, the constituent systems cooperate to achieve some global goal. Hence our investigation aims to ascertain whether researchers in SoS testing, which is a younger topic, could usefully learn from the more mature SPL testing literature, and more specifically, which methodologies and technical approaches could be imported and adapted.

The paper is structured as follows: in Section II, we present SoS testing challenges at different testing levels, while in Section III we present an overview of SPL research on functional testing. Then in Section IV, we outline commonalities and differences between the SoS and SPL paradigms from the point of view of testing, pointing out how existing methods and tools from SPL testing could be leveraged for each different SoS testing level. Finally, we draw conclusions and future work in Section V.

II. Concepts and Challenges in SoS Testing

According to [4], [6], SoSs are classified into four types of architectures that take into account the existence of a central governing unit and the way in which the Constituent Systems (CSs) relate to it to meet the SoS missions. In *directed* SoSs, the normal mode of operation of the constituent systems is subordinate to a central unit, although these systems are capable of operating independently; despite constituent systems are subordinated to a central authority, in *acknowledged* SoSs, they maintain their independent ownership and objectives; in *collaborative* SoSs, the systems are not forced to follow the central unit, that is, they collaborate voluntarily to attend SoS missions; finally, in *virtual* SoSs there is no central unit and the SoS has no explicit purpose.

SoSs are usually very large systems, which exhibit characteristics of operational and managerial independence, geographic distribution, evolutionary development, and emergent behaviour. For these reasons, SoS testing at all levels present significant challenges, as we briefly discuss in the following.

At the **unit level** of SoS testing, each individual CS should be tested in the SoS environment. However, the unavailability of the CS source-code and of its test cases could make it difficult to ensure that the CS has the adequate quality to participate in the SoS. Even if the constituent system's supplier provides the original test cases for testing the CS, their number can be very high, and in addition many of them could aim at testing features that are not used by SoS, and hence not in scope [7]. Also, SoS can be composed of different types of CSs, not always deterministic, such as in cyber-physical systems or artificial intelligence applications using machine learning approaches, making it difficult to establish a common quality criterion for all of them.

To ensure that each CS cooperates properly with its peers, the SoS must be tested at the **integration testing** level. The large number of CSs, which can join or leave the SoS at any time, can generate an exponential number of interactions and execution flows that need to be tested, requiring an even higher efficiency for test cases generation and orchestration strategies. The authors of [8], [9] propose frameworks to address these problems. In particular, the authors of [8] identify all relevant SoS entities and their interfaces and the flow of information between the CSs, then they define combinatorial testing strategies for optimizing the testing and SoS evaluation. The work in [9] uses a randomization approach to design a test plan for SoS that minimizes the number of possible test cases during the integration of the CSs. CSs can also vary in terms of the offered functionality and can use different protocols for communication and data transmission, bringing up interoperability issues that should be also considered during the integration testing. Also, there may exist interdependences between the CSs and, depending on the integration strategy used, it may be necessary to create mocks, which may not be straightforward.

To verify that the SoS fulfills its missions, we proceed with the SoS **system testing**. In this phase, the absence of requirements models and the scarce documentation makes it difficult to define the scope of what needs to be tested and undermines the application of formal verification methods. CSs may show wrong behaviour when operated in a particular SoS configuration due to emergent properties insufficiently captured during the requirements gathering phase. Also, the number of states and settings that a SoS can reach prevents the use of exhaustive or exploratory testing. The work in [10] defines an approach similar to white-box testing to generate test case suites for a SoS. They use a basic path testing approach, modelling the CSs as the nodes of a control flow graph; however they do not carry out case studies and do not discuss how their approach can scale up in a scenario with many CSs.

Due to the highly dynamic and evolutionary nature of SoSs, it is impossible to obtain a permanent state as well as to foresee all the possible changes. In fact, the individual CSs can evolve, and the whole SoS can change its composition, or even start using new features. Hence, in the context of SoS testing, **regression testing** should be used to ensure that each new SoS configuration does not cause any inappropriate emergent behaviour. However, once again, the large number of CSs can make existing regression testing techniques not scalable [11]. The authors of [12], [13] present the PATFrame framework that aims to predict when a test system needs to be adapted using the information learned during the test process. The proposal of [11] deals with a conceptual framework to govern regression testing for collaborative and acknowledged SoS, based on the regression test objectives for each phase of the SoS and using an orchestration graph.

TABLE I
SPL TESTING CONCEPTS

Group of strategies [14]	Product specific testing
	Incremental testing of SPL
	Reusable asset instantiation
	Division of responsibilities
Investigation topic [18], [17], [19]	Domain engineering
	Application engineering
SPL testing interest [20], [17]	Feature-based
	Product-based

III. Software Product Line Testing Approaches

Testing software product lines represent a consolidated research direction as evidenced by the wide variety of testing strategies and several surveys published in the last two decades, e.g., [14], [15], [16], [17]. In the following we provide a brief overview of main concepts: for readability, these are also summarised in Table I.

The work in [14], published in 2004, represents one of the initial analyses on existing SPLs testing practices and has been considered for a long time a reference work in the field. It investigates how to make use of the special characteristics of product families for testing purposes and identifies four different groups of strategies for integrating testing into product family engineering that are: i) *product specific testing*, which focuses on the validation of a specific product and does not exploit the benefits of reuse in product families. In this strategy group, the tests for each derived application are developed independently from each other, but this is extremely costly; ii) *incremental testing of product families*, in which the first product is tested individually and the following products are tested using regression testing techniques; iii) *reusable asset instantiation*, in which extensive test assets and abstract test cases are created; iv) *division of responsibilities*, in which the different testing activities are distributed among the different engineering units.

More recent surveys [15], [16] present similar general test strategies and use them for classifying the analysed primary studies. According to [18], [17], [19], studies on SPL testing can belong to two investigation topics: *domain engineering* or *application engineering*. The former focuses on the entire software product line while the latter focuses on individual system applications (i.e., the members of the SPL). In domain

engineering, extensive variable test assets are created taking as input the variability defined for the product line. In application engineering, where an application is instantiated according to the requirements, the abstract test cases are extended or refined to test the product-specific aspects. A lot of research in SPL testing has been conducted at both levels, but only a few works span over both topics.

The main interests of SPL testing can be summarised as *feature-based* and *product-based* [20], [17]. The approaches following the first interest are based on feature interaction coverage, namely they aim to check that every feature combination is consistent with the specification and does not violate the stated constraints. In this context, *Combinatorial Interaction Testing (CIT)* [19] supports the selection of the test set covering the combinations of features that will be present in most products or in which the interaction faults are more likely to occur. Towards this goal, *t-wise* feature coverage has been applied in conjunction with SAT solvers, to reduce the set of possibilities to a reasonable and representative set of product configurations. In particular, many of the works analysed in [17] applied pairwise as a test case reduction heuristic: instead of combining all the features, they describe the input model as a combination of a couple of features and try to satisfy the constraints between them, so to reduce the test set and find inconsistencies.

The approaches following the product-based interest aim to check the correctness of functionalities of each product. Testing approaches following this second interest define test cases and test scenarios by leveraging two main features that are: *variability* and *asset reuse*. In the former case, model-based strategies in which test models are able to capture variability among products [21] can be used. For instance, parameterised UML models can be adopted annotating the variability, by means of parameters or stereotypes. In the latter case, approaches aim to reuse test cases, test scenarios, test results, and test data, either among products or from a core asset base. The main idea is to consider a SPL as a core module, that is an implementation of a valid product, and a set of delta modules that represent changes to be applied to the core module to obtain further products. Testing is focused on these changes and on the differences between product instances; this allows for an increased reuse of test assets and test results among products, based on regression testing principles.

Finally, the authors of [19] investigate the usage of Search-Based Software Engineering (SBSE) techniques into SPL and identify testing as the main application of SBSE. In particular, greedy algorithms and genetic algorithms are the most common SBSE techniques used for domain testing. The same authors also address prioritisation and optimisation problems of SPL test suites, aiming at the minimisation of their sizes and the maximisation of their t-wise coverage.

IV. From SPL testing towards SoS testing

Based on the above overviews, it should be evident how SPL and SoS paradigms share several underlying principles:

i. large scale reuse of software components or constituent systems;
ii. reduced development costs and time-to-market;
iii. capability of dynamic and fast (re)configuration and customisation.

Due to such characteristics, one of the major problems in SPL testing is the unfeasibly large number of variations that should be executed. In fact, several SPL testing approaches are conceived to avoid the exhaustive testing of each single product independently. Similarly, many of the challenges in SoS testing are due to a large number of possible behaviours that an SoS can expose when considering all possible ways in which its constituent systems could interact. We can understand each entry, exit, replacement, or even evolution of a CS as a new SoS configuration that we need to test.

However, there also exist notable differences: while variability is a fundamental aspect of SPL, in SoS it can happen but is not central. Indeed, both in SPL and SoS a same component/CS can be used in more collaborations (product/mission). However, in each new product, a SPL component is used under differing configurations using differing features; in SoS a same existing individual is reused as it is, usually without customisation, but relying on different functionalities among the offered services, depending on the mission.

In SoS a fundamental aspect is the autonomy of the CSs, whereas in SPL the components are generally provided by a central owner. In this sense, SPL appears more similar to the case of directed or acknowledged SoS architectures.

In Section II, we identified four testing levels for SoS: unit, integration, system, regression. In Table II we attempt a speculation of how some of the results in SPL testing research could be considered and adapted to the testing of SoSs.

Concerning testing SoS at the unit level, this entails the testing of each single CS independently from the SoS in which it is involved and traditional testing strategies apply. At this level, we do not see neither peculiar SoS testing concerns, nor specific techniques from SPL testing literature [22], [17] that would be relevant. Hence, the table does not include a row for unit level, and in the following, we focus on integration, system and regression testing.

Concerning SPL feature-based testing interest, as explained in [17], *"...when checking the properties or configurations of a SPL, every feature combination have to be consistent with the specification and must not violate the stated constraints"*. In the context of SoS, similar concerns apply for SoS integration and system testing levels.

In SoSs, the number of possible combinations of all functionalities of the involved CSs increases exponentially with the number of involved CSs. Then, testing all possible combinations is, therefore, time-consuming and costly.

Specifically, at integration testing level, as evidenced in Table II (second row), an important challenge is to reduce the vast potential configurations in which CSs could collaborate. For this, we envisage that the broad SPL literature on Combinatorial Interaction Testing (CIT) and Search-Based Software Engineering (SBSE) targeting the domain engineering scope

TABLE II
FROM SPL TESTING TOWARDS SOS TESTING

SoS testing level	SoS goal	SoS challenge	Related SPL testing research scope	Related SPL strategies	SPL approaches that could be adapted	Adoption of SPL approach in SOS testing
Integration	testing the integration of all possible combinations of all variations of CSs	reducing the set of possibilities to a reasonable and representative set of SoS configurations	domain engineering	feature-based	combinatorial interaction testing (CIT)	compute test suites for all possible combinations of t selected and (un)selected functionalities of CSs
					search-based software engineering (SBSE)	select a smaller test set for a subset of possible SoSs
		creating CSs mocks	-	-	-	-
System	testing the proper behaviour or mission of a whole SoS	difficulty of providing a system model	domain engineering	product-based	variability model	define reusable test assets for SOS
			application engineering		asset reuse	instantiation of the test asset for deriving test cases for the specific SoS mission
Regression	ensure that updates and evolution of the CSs or of their interactions do not cause unintended emergent behaviours	orchestration of test cases	domain engineering	feature-based	prioritization	reordering SoSs to be tested and their test cases for early fault detection
		finding those test cases for the specific SoS that are more likely to spot inappropriate or unexpected behaviours	application engineering	product-based	asset reuse	instantiation of the test asset for deriving test cases for the specific SoS

and proposing feature-based solutions, could be considered and adapted. In the former case, the goal is to derive test suites for all possible combinations of t features of a SPL feature model, corresponding to a set of products where the interaction faults are most likely to occur. In particular, pair-wise feature coverage (in which the value of t is 2) in conjunction with SAT solvers represents the most used technique for deriving test cases. For instance, the authors of [23] provide a constraint-based testing tool for automatic generation of test configurations that cover all pairwise interactions in a feature model. Similar approaches could be used in SoS testing context for reducing the number of combinations of CSs configurations to be tested in SoSs. In the latter case, greedy algorithms or genetic algorithms are used for keeping as small as possible the number of tests that are generated for the product line. As an example of such category, the work in [24] leverages an objective function based on cyclometric complexity metric adapted to feature models. Similar approaches could be used for searching the possible configuration space of a set of CSs and then selecting a limited number of the most effective configurations to be addressed by the tests in the derived test suite.

As sketched in Table II, another challenge of SoS testing at integration testing level is the creation of mocks that could be used to test the interaction of a specific CS with other CSs that are not available at testing stage. The creation of mocks or stubs is also needed in SPL testing for replacing undeveloped modules of absent variants [25]. However, at the best of our knowledge, no specific technique tailored at the SPL context is adopted for the mocks creation.

Concerning product-based testing, this would entail [17] *"...to check the set of correctness properties of each product. Given that a built software artifact can be used by a range of products, an uncovered defect may be propagated to the many products that include it"*. In the SoS context, the research aims at approaches for validating the proper behaviour of each CS when interacting within the emerging SoS architecture. At SoS system testing level, this amounts to derive from the SoS model a set of end-to-end test cases for verifying the SoS missions.

At system testing level, for the purpose of testing, each unique mission for a SoS can be understood as a different product in a SPL. Therefore, we can assimilate testing approaches investigated by SPL researchers for application engineering to mission-oriented testing research in SoS. Further, we can use a same set of CSs to compose different SoS pursuing other missions. An example is presented in [26], in which we show how the same collection of CSs can be organised in different ways in order to fulfil different missions from three SoSs.

Therefore, as sketched in Table II (third row), a challenge of SoS system testing is that of having a system model expressing all variable functionalities of CSs, ensuring that only the required functionalities are included in the resulting SoS. Moreover, a common functionality in different SoSs may require different tests. In the context of SPL domain engineering, feature-based modelling approaches are used to enable the annotation of commonalities and variability of products in test models. Some approaches, such as [27], [28], [29] propose test case derivation from requirements modelled using enhanced UML use cases, and allow to specify variability in the test cases and test scenarios [30]. In the context of SPL application engineering, the key idea is to develop the domain models of the product line and then instantiate them to a given product. These approaches could be exploited for defining reusable test assets (test cases or test scenarios) for all SoSs in which a set of CSs in involved and then instantiating these assets so to derive the test cases for the specific SoS.

With reference to the SoS regression testing level (see the fourth row of Table II), the main goal is to identify, select, and possibly combine regression test cases in order to assess

if SoSs keep working correctly even after changes have been introduced either in their CSs or in the interactions among CSs [11]. Specifically, considering the testing of different SoSs emerging from the collaboration of several CSs, one important challenge is defining an orchestration framework in order to select and compose the existing test cases of SoSs according to a dependency graph and automate the regression testing activities [11]. In the context of SPL domain engineering testing, starting from a feature model, the set of products to be tested is selected (e.g., by using t-wise), and then, this set of products is prioritised according to multiple criteria determining the execution order of the test cases. The work in [31] provides a comparison of prioritisation approaches that could be leveraged in the SoS context for providing an ordered set of SoSs to be tested aiming to detect critical faults as soon as possible, e.g., those causing failures in a higher number of SoSs (corresponding to the products of a SPL).

Concerning the regression testing of a specific SoS instead, the challenge is to identify those test cases that are more likely to spot inappropriate or unexpected behaviours. However, due to a large number of SoS configurations, to ensure its correctness, in general, it is not feasible to design and conduct the testing of a specific SoS in isolation. In the context of SPL application testing, reuse approaches for testing [32] exist that focus on the differences of product instances with respect to a core module represented by a valid product. In the same way, as also showed in [26], a set of SoSs can be originated by a core SoS and a set of additional CSs or variations in the existing CSs able to originate new and valid SoSs. Then, the approaches of SPL variability models and asset reuse could be adopted for deriving test cases for a new SoS only focusing on the deltas and reusing the test assets of missions and configurations already tested for other SoSs.

V. Conclusion and Future Work

In this paper, we investigated how the research on SoS testing could benefit from the extensive literature on SPL testing. We identified several commonalities and convergences among SPL and SoS paradigms, due to the high configurability and reusability of the components of both types of systems. After analysing the literature on SPL testing, we identified the main strategies that could be leveraged to address the challenges at the different levels of SoS testing. The most challenging testing levels in which we identified potential SPL testing approaches that could be leveraged for improving SoS testing are integration, system and regression, whereas neither peculiar SoS testing issues, nor specific SPL techniques have been found for unit testing level.

In this paper, we focused on functional testing. However, also non-functional testing is very relevant in the system level of SoS, for instance, also related to the availability, security concerns, and reputation of the CSs. On the other hand, testing of well-known quality attributes, such as response time, performance, availability, and scalability is crucial in SPL for any derived resource-constrained product [15].

It is worth mentioning that the application of SPL testing approaches and tools for SoS testing is not straightforward, and it is necessary to adapt them to the SoS particularities. As future work, we plan to extend our work offering guidelines on how these approaches can be adjusted as well as deriving an SoS case study in which we will show the practical applicability of specific SPL testing approaches at different testing levels.

In the future, we are also interested in investigating more about non-functional test approaches adopted in SPL and how they can be leveraged for non-functional testing of SoS. Moreover, we speculate that perhaps the cross-fertilisation could happen in the opposite direction: SPL research could learn approaches used for non-functional testing of SoS, e.g. as in [33], to enhance their security.

In this paper, after a very quick review of the literature, we identified a significant set of papers useful to detect a possible correlation between SoS and SPL from the point of view of testing. Moreover, it would be interesting in the future to conduct a Rapid Review (RR), following the approach presented in [34] that also involves the collaboration of practitioners, in order to investigate more about the convergence and differences between SPL and SoS testing and provide in a timely way evidence of our findings.

VI. Acknowledgements

This paper has been partially supported by the Italian MIUR PRIN 2017 Project: SISMA (Contract 201752ENYB).

References

[1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, ser. FOSE '07. USA: IEEE Computer Society, 2007, p. 85–103. [Online]. Available: https://doi.org/10.1109/FOSE.2007.25

[2] A. Bertolino and P. Inverardi, "Changing software in a changing world: How to test in presence of variability, adaptation and evolution?" in *From Software Engineering to Formal Methods and Tools, and Back*, ser. LNCS, M. H. ter Beek, A. Fantechi, and L. Semini, Eds., vol. 11865. Springer, 2019, pp. 56–66. [Online]. Available: https://doi.org/10.1007/978-3-030-30985-5_5

[3] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–41, 2015.

[4] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[5] V. O. Neves, A. Bertolino, G. De Angelis, and L. Garcés, "Do we need new strategies for testing systems-of-systems?" in *Proc. Int. Workshop on Software Engineering for Systems of Syestems*, 2018, p. 29–32.

[6] J. S. Dahmann and K. J. Baldwin, "Understanding the current state of US defense systems of systems and the implications for systems engineering," in *Systems Conference*. IEEE, 2008, pp. 1–7.

[7] B. Miranda and A. Bertolino, "Testing relative to usage scope: Revisiting software coverage criteria," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 3, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3389126

[8] S. Luna, A. J. Lopes, H. Y. S. Tao, F. Zapata, and R. Pineda, "Integration, Verification, Validation, Test, and Evaluation (IVVT&E) Framework for System of Systems (SoS)." in *Complex Adaptive Systems*, ser. Procedia Computer Science, C. H. Dagli, Ed., vol. 20. Elsevier, 2013, pp. 298–305.

[9] Q. Liang and S. H. Rubin, "Randomization for testing systems of systems," in *2009 IEEE International Conference on Information Reuse Integration*, Aug 2009, pp. 110–114.

[10] F. Zapata, A. Akundi, R. Pineda, and E. Smith, "Basis path analysis for testing complex system of systems," *Procedia Computer Science*, vol. 20, pp. 256–261, 2013.

[11] A. Bertolino, G. De Angelis, and F. Lonetti, "Governing regression testing in systems of systems," in *Proc. of Int. Symposium on Software Reliability Engineering Workshops*. IEEE, 2019, pp. 144–148.

[12] J. T. Hess and R. Valerdi, "Test and evaluation of a SoS using a prescriptive and adaptive testing framework," in *2010 5th International Conference on System of Systems Engineering*, June 2010, pp. 1–6.

[13] S. Ferreira, R. Valerdi, N. Medvidović, J. Hess, I. Deonandan, T. Mikaelian, and G. Shull, "Unmanned and autonomous systems of systems test and evaluation: Challenges and opportunities," in *IEEE Systems Conference*, 2010, p. 15.

[14] A. Tevanlinna, J. Taina, and R. Kauppinen, "Product family testing: a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 2, pp. 1–6, 2004.

[15] P. A. d. M. S. Neto, I. do Carmo Machado, J. D. McGregor, E. S. De Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *Information and Software Technology*, vol. 53, no. 5, pp. 407–423, 2011.

[16] E. Engström and P. Runeson, "Software product line testing–a systematic mapping study," *Information and Software Technology*, vol. 53, no. 1, pp. 2–13, 2011.

[17] I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. De Almeida, "On strategies for testing software product lines: A systematic literature review," *Information and Software Technology*, vol. 56, no. 10, pp. 1183–1199, 2014.

[18] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[19] R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed, "A systematic mapping study of search-based software engineering for software product lines," *Information and software technology*, vol. 61, pp. 33–51, 2015.

[20] I. do Carmo Machado, J. D. McGregor, and E. Santana de Almeida, "Strategies for testing products in software product lines," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–8, 2012.

[21] K. L. Petry, E. Oliveira Jr, and A. F. Zorzo, "Model-based testing of software product lines: Mapping study and research roadmap," *Journal of Systems and Software*, p. 110608, 2020.

[22] B. Pérez, M. Polo, and M. Piatini, "Software product line testing-a systematic review," in *4th International Conference on Software and Data Technologies (ICSoft 2009), Sofia, Bulgaria*, 2009.

[23] A. Hervieu, B. Baudry, and A. Gotlieb, "Pacogen: Automatic generation of pairwise test configurations from feature models," in *Proc. of IEEE 22nd International Symposium on Software Reliability Engineering*. IEEE, 2011, pp. 120–129.

[24] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary search-based test generation for software product line feature models," in *Proc. of International Conference on Advanced Information Systems Engineering*. Springer, 2012, pp. 613–628.

[25] J. Lee, S. Kang, and D. Lee, "A survey on software product line testing," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*, 2012, pp. 31–40.

[26] A. Bertolino, G. D. Angelis, F. Lonetti, V. O. Neves, and M. A. Olivero, "EDUFYSoS: A factory of educational system of systems case studies," in *15th IEEE International Conference of System of Systems Engineering, SoSE 2020, Budapest, Hungary, June 2-4, 2020*. IEEE, 2020, pp. 205–210. [Online]. Available: https://doi.org/10.1109/SoSE50414.2020.9130551

[27] A. Bertolino and S. Gnesi, "Use case-based testing of product lines," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 5, pp. 355–358, 2003.

[28] E. Kamsties, K. Pohl, S. Reis, and A. Reuys, "Testing variabilities in use case models," in *International Workshop on Software Product-Family Engineering*. Springer, 2003, pp. 6–18.

[29] C. Nebut, Y. Le Traon, and J.-M. Jézéquel, "System testing of product lines: From requirements to test cases," in *Software Product Lines*. Springer, 2006, pp. 447–477.

[30] S. Kang, J. Lee, M. Kim, and W. Lee, "Towards a formal framework for product line test development," in *CIT 2007: 7th IEEE International Conference on Computer and Information Technology*, 11 2007, pp. 921 – 926.

[31] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés, "A comparison of test case prioritization criteria for software product lines," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 2014, pp. 41–50.

[32] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental model-based testing of delta-oriented software product lines," in *International Conference on Tests and Proofs*. Springer, 2012, pp. 67–82.

[33] M. A. Olivero, A. Bertolino, F. J. D. Mayo, M. J. Escalona, and I. Matteucci, "Addressing security properties in systems of systems: Challenges and ideas," in *Software Engineering for Resilient Systems - 11th International Workshop, SERENE 2019, Naples, Italy, September 17, 2019, Proceedings*, ser. Lecture Notes in Computer Science, R. Calinescu and F. D. Giandomenico, Eds., vol. 11732. Springer, 2019, pp. 138–146. [Online]. Available: https://doi.org/10.1007/978-3-030-30856-8_10

[34] B. Cartaxo, G. Pinto, and S. Soares, "The role of rapid reviews in supporting decision-making in software engineering practice," in *Proc. of the 22nd International Conference on Evaluation and Assessment in Software Engineering*, 2018, pp. 24–34.