Journal Homepage: -www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI:10.21474/IJAR01/12280
DOI URL: http://dx.doi.org/10.21474/IJAR01/12280

ISSN NO. 2320-5407

*RESEARCH ARTICLE*

# A SURVEY ON SOFTWARE PRODUCT-LINE TESTING

**Ayat Yehia Hassan**
Master Student, Department of Computer Science, King Abdul-Aziz University, Jeddah, Saudi Arabia.

## *Manuscript Info*

***Manuscript History***
Received: 05 November 2020
Final Accepted: 10 December 2020
Published: January 2021

***Key words: -***
SPL, Software Engineering, Testing, Feature Inclusion Graph (GIF), Product Prioritization

## *Abstract*

SPL (Software Product Line) is known as a set of software systems that share a mutual set of features. It is a powerful concept to achieve more efficient software system development. One of the necessary steps in software development processes is Testing. It consumes typically more than 50% of the whole development costs. Testing SPLs is challenging due to the exponential number of products in the number of features. Several approaches have been proposed to reduce the number of products to be tested. However, the testing aspect of SPL is still underdeveloped. This study aims at surveying the latest research on SPL testing to identify useful approaches and needs for future research. seven papers are classified concerning the following: the used Approach, the algorithms, and the type of testing that the research focuses on. The survey found that more validation and evaluation research is required to produce a more robust foundation for SPL testing. Finally, directions for future software product line testing recommendations.



## Introduction:-

Software Product Line (SPL) is a software engineering paradigm that facilitates the development of software products that share common functionalities. The main goal of developing a system as a software product line is to create a software structure that is customizable to various needs, by maximizing software reusability [11]. In SPL, a unit of system function is represented as a feature. Features are explicitly defined as common or variable features and utilized throughout the SPL development process. One way to model the commonalities and variabilities in an SPL is by using a Feature Model (FM) based on the feature modeling technique [11],[12].

SPLs bring many benefits such as code reusability, a faster time to market, reduced costs, and flexible productivity [14]. However, SPLs are challenging to test due to the large amount of possible software that can be configured [15]. For instance, 20 optional features lead to 2^22 possible products to configure, meaning more than a million different software products that should be tested independently. Such a testing budget is usually unavailable for economic, technical, or time reasons, preventing the SPL from being exhaustively tested [13]. Thus, defining methods for generating test suites while giving enough confidence in what is tested is required, and is now a very active research topic for the testing community [16,17,18]. In respect of that, recently much research effort is spent on a variety of topics related to product line testing. To get a picture of existing research, we provide an overview of the techniques that are included in some recent papers, Also, brief reviews and discussions for the different approaches. Finally, we suggested a future direction for SPL testing researchers.

**Corresponding Author:-Ayat Yehia Hassan**
Address:-Master Student, Department of Computer Science, King Abdul-Aziz University, Jeddah, Saudi Arabia.

This paper is organized as follows. Section II explains the schema that classifies the SPL testing approaches and surveys seven research papers based on this schema. In Section III, we present a Discussion that contains a critical analysis of the surveyed papers. And We Identify the relative advantages and disadvantages of classified approaches. Section IV concludes this paper and discusses future work based on the survey.

## Software Product Line Testing Approaches

There are many testing Approaches used to reduce the number of products to be tested in SPL. Each of the approaches still has problems with scalability. This section provides an overview of each of them.

***Combinatorial Testing-Based Approach***

Combinatorial Interaction Testing (CIT) [19] is a popular technique that has been applied to SPLs to reduce the size of the test suites. CIT operates by generating only the product configurations exercising feature interactions. Combination testing strategies identify test combinations by combining values of different test object parameters based on some combinatorial strategy Algorithms for selecting the combinations.

The Coverage criteria for combination strategies are:

1. **Each-used (1-wise) coverage:** Every interesting value of each parameter is covered with some test case.
2. **Pair-wise coverage:** Every possible pair of interesting values of any two parameters is covered.
3. **T-wise coverage (Combinatorial Interaction Testing (CIT)):** Every possible combination of interesting values of t parameters is covered where *t* is the interaction strength.
4. **N-wise coverage:** All possible combinations of interesting values of all N parameters are covered.

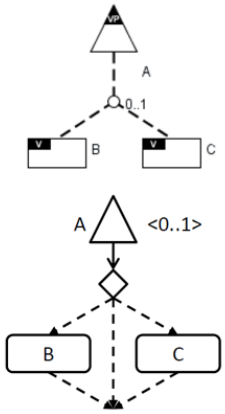


**Figure 1: Feature Inclusion Graph (FIG)**

**Feature Inclusion Graph (FIG)-based Approach**

Feature Inclusion Graph (FIG) [3] is a testing technique based on translating a feature model into a feature inclusion graph as shown in figure 1. Then it associates all features with sets of test cases and then walks this graph to generate a subset of independent paths (or products) that cover the graph for testing. This approach aims to find a "possibly minimal" set of paths to cover all nodes using a selection algorithm. The selection algorithms used to select products to be tested can be divided into two main types. FIG-based algorithms and non-FIG-based algorithm as shown in figure 2.

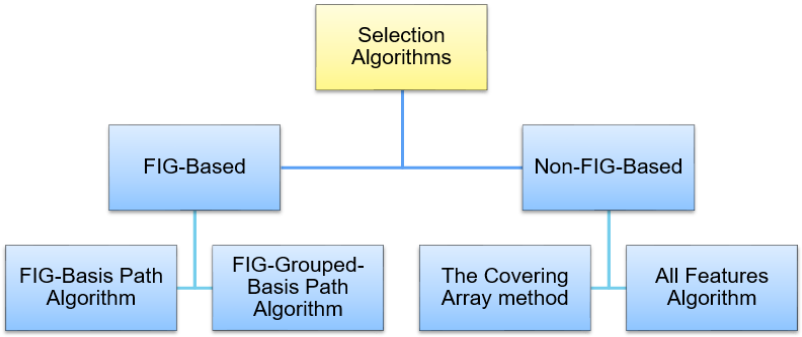


Figure 2: Selection Algorithms

***Product Prioritization Approach***

The order in which products are tested is important because it can increase the early rate of fault detection. The result of each testing algorithm is a set of products that need to be tested. The tester is responsible for which order these products are tested. This approach is used to prioritize products based on different criteria. An example of this approach is Delta-Oriented Product Prioritization Approach [4] which uses a delta to capture the differences between products to calculate product similarity. Then, it selects the most dissimilar product to the previously tested ones, to be tested next. Another example of this approach is Similarity-Based Prioritization [6] Which uses prioritization to be applied to the products before they are generated. It aims at increasing the interaction coverage of an SPL under test as fast as possible over time.

***Factor Interdependency Graphs (FIG)-based approach***

Factor interdependency graphs have been introduced by D.M.Brandon [22] to test web applications inside the Project Evaluation Framework to check if the web application is ready to release or will continue to the next stage. Project evaluation framework (PEF) is a tool that helps to determine if the project in its current state is ready to proceed to the next phase of a slice, ready for release, and/or needs to continue to the next slice of Augmented web Process. The application of the steps of the PEF results in an AND-OR graph called the Factor Interdependency Graph or a FIG. FIG establishes the relationships between the factors affecting Release and Continue stages (also referred to as variables of the project). There can be three types of relationships: AND, OR, and EQUAL. In an AND relationship all the child factors need to be satisfied for the parent factor to be satisfied and even if one child factor is not satisfied then the parent factor is not satisfied. In an OR relationship satisfaction of even one child factor satisfies the parent factor. In an EQUAL relationship, the parent factor takes the same level of satisfaction as the child [22].

| Study Name | Approach | Algorithm | Testing type | Research focus |
|---|---|---|---|---|
| **FIG based Quality Assurance in Software Product Lines [1]** | Factor Interdependency Graphs-based approach | Depth-First search algorithm | Black box testing | - Quality Assurance<br>- Early defect detection<br>- Fault correction |
| **SPLBA: An interaction strategy for testing software product lines using the bat-inspired algorithm [2]** | Combinatorial Testing-Based Approach (- t-way strategy) Using Covering Array (CA) | Bat-inspired algorithm (BA) | Combinatorial t-way testing | - Tests reduction<br>- Early defect detection<br>- Fault correction<br>- Fault detection<br>- Commonality and variability |
| **Improving the testing and testability of software product lines [3]** | - Feature Inclusion Graph (FIG)-based Approach<br>- Non-Feature Inclusion Graph (FIG)-based Approach | - FIG Basis Path Algorithm<br>- FIG Grouped Basis Path Algorithm<br>- The Covering Array method<br>- All Features Algorithm | - Black-box testing<br>- Combinatorial interaction testing | - Effort reduction<br>- Fault detection<br>- Test cases and product reduction<br>- Commonality and variability |
| **Delta-oriented product prioritization for similarity-based product-line testing [4]** | Similarity-based Product Prioritization approach | Sampling algorithms | Combinatorial interaction testing | Prioritization to increase the early rate of fault detection |
| **Efficient testing of software product lines via centralization [5]** | Centralization-based test case generation approach | - | Random testing | Improving testing efficiency by reducing redundancy in test executions. |

| | | | | |
|---|---|---|---|---|
| **Similarity-based prioritization in software product-line testing [6]** | Similarity-based Product Prioritization approach | Prioritization testing applied on three sampling Algorithms CASA [23], Chvatal [24], and ICPL [25] | Combinatorial interaction testing | Prioritization to increase the early rate of fault detection |
| **A Univariate Marginal Approach for Pairwise Testing of Software Product Lines [7]** | Combinatorial Testing-Based Approach (Pair-way strategy) | Univariate Marginal Distribution Algorithm (UMDA). | Combinatorial Testing (Pair-way) | Minimizing the number of generated test configurations |

**Table 1:-** Summary of papers by research approach.

**An example of this approach is the Factor Interdependency Graphs (FIG)-based Quality Assurance approach [1]. It is a systematic testing technique that uses Factor Interdependency Graphs (FIG) to test the product lines to ensure early defect detection, fault correction and to find a common set of testing assets for the subsequent products of the SPL. Figure 3 displays an Example factor interdependency graph for slice-stage.**

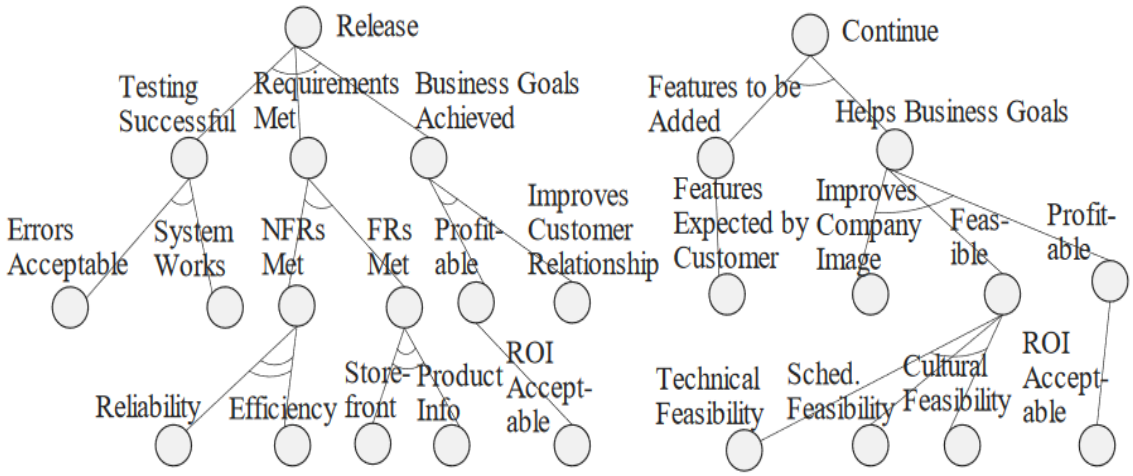


**Figure 3:-** Example factor interdependency graph (FIG) for slice-stage.

**Centralization based test case generation Approach**

The motivation of this approach is that the existing test case generation techniques may redundantly test common code. Moreover, it is difficult to share the test results among multiple product variants. This approach combines multiple product variants from the same SPL and generates test cases for the entire system. By considering all variants, this technique generally avoids generating redundant test cases.

We have classified the reviewed papers according to the following: the approach, the used algorithm, and the type of testing as shown in table1.

## Discussion:-

There has been a lot of work on testing the software product line. In this work, we present an overview of the latest approaches to test a software product line by reviewing seven papers. Some of the research discussed the testability and quality accordance [1,3]. The research study [1], shows that quality matters the most, specifically in product line engineering to keep the family line intact. Also, the study discussed a strategy to evaluate the quality attributes which are critical to the product. The main advantage of the study is that it presents the idea of FIGs in testing the

program product lines for early Fault and error detection. But The strategy discussed in the paper is not new. It has been used in testing web applications previously. It has been altered for the SPL testing to test the external behavior of the product. However, one of the strategy limitations is that it cannot test internal behavior as it can test the external behavior of the item. Also, it uses all the components as input to produce parent and child components, which is at that point used for conformance to user specification. That implies testers must depend on their intuition to make the test cases and breaking them into the child factors.

The research paper [3] introduced the FIG Basis Path approach. They showed by experiment that the FIG Basis Path method is efficient at finding faults and is at least as effective as other techniques. In the product line that they define as less testable due to the alternative features (GPL) they prove that the Basis Path method performed the best. It found as many faults as the other techniques for 60% fewer test cases than the CA technique, and 55% fewer products. Also, in application with a more testable product due to the small number of alternative features, we see that the FIG Basis Path was as effective at finding faults as all other methods, although the cost of computing the FIG Basis Path may be slightly higher than that for All Features, the technique appears to work well for both types of feature model elements (alternative and optional), therefore it is the more robust technique. However Further evaluation is needed to understand the efficiency of using a FIG with higher granularity variants. Also, it needs further analysisof the impact of faults that are embedded inside of the variant portions of the code to fully understand the effectiveness of these techniques. Finally,the small number of constraints application has some influence over the method. So, for that FIG can be used with the Grouped Basis Path algorithm to reduce test effort.

In SPLBA [2], the Authors presented a complete experimental design of the SPLBA system to support their strategies. They evaluated its performance in terms of test suite reduction. The results and the experiment demonstrated the efficiency of SPLBA against all the existing strategy. Their proposed conclusions are critical for two reasons: First, much existing work has not given much focus on the support for constraints interaction with a high number of parameters. Second, existing work has not adequately considered the appropriation of Bat Algorithm as the basis of the t-way execution in line with the current buildup on the modern field called Search-based Software Engineering (SBSE). The authors used a smart mobile system as a case study to explain SPLBA strategy and the related concepts such as feature model diagram, relationships, covering arrays (CA), parameters, and constraints. The experimental results goal was to benchmark SPLBA against existing SPLs strategies to demonstrate its effectiveness. The data size and the frequency of times they run the experiment are sufficient to validate that their strategy can compete with the existing strategies. However, it was not enough to prove that their strategy outperforms the other strategies in most cases. Also, the Authors did not explain why in some cases other strategies outperform SPLBA. The paper presents an attempt at solving the same problem that has many previous solutions. So, it is recommended to include the discussion part to evaluate the results and explain why SPLBA outperforms all strategies in terms of test suite reduction in some cases and why the LOOKUP strategy outperforms all the strategies for the other cases.

Delta-oriented product prioritization for similarity-based [4] made the following contributions: (1) it proposes an approach for product prioritization that uses delta modeling for similarity-based prioritization avoiding the generation of complete products on the solution-space level. (2) it provides a combined approach of configuration and delta-oriented prioritization in similarity-based product prioritization. Authors compare it against random orders as well as the default order of the sampling algorithm MoSo-PoLiTe [23], delta-oriented prioritization outperforms the random ordering (0.863) as well as the default order of MoSo-PoLiTe [23] sampling algorithm. One of the advantages of this study is that it prioritizes products based on their similarity in terms of deltas. Hence, it does not require expert knowledge to prioritize products. One internal threat in [4] is that it is compared 100 times with the random order. Regarding external validity, we cannot ensure that the tested subject SPL is representative of real-world SPLs.

In the research [5], Although this paper focuses on testing multiple product variants of SPLs, the concept and techniques presented in this paper do not depend on domain knowledge of SPLs (such as a feature model). They can generalize to other testing scenarios for multiple similar product variants such as historical program versions from software evolution and co-evolution. However, the representativeness of selected subjects is the primary externalthreat to validity. The research selected three SPLs from different categories that have been widely used in previous studies. It also uses its recent implementation based on Feature House. Another external threat to validity is caused by the randomness of Randoop-a tool forimplementing random test case generation-. But they fix and use the same random seed and run each configuration long enough to diminish this threat.

The research [6] propose similarity-based prioritization to order the products before they are generated and tested. they extend FeatureIDE [37] with support for product line testing and similarity-based prioritization. and they evaluate the activeness of similarity-based prioritization compared to the default order of three sampling algorithms, namely ICPL, CASA, and Chvatal, and to random orders. They found that similarity-based prioritization is better than the default order of each sampling algorithm (CASA, Chvatal, and ICPL) in all SPLs. However,we cannot ensure that the proposed approach will provide the same results for larger feature models. This validity threat comes from the size of feature models and the constraints that may affect the results of their similarity-based prioritization. they evaluatedtheir approach against the default order of each sampling algorithm.

In the research [7] the UDMA-based solution managed to generate a smaller number of test configurations on majority datasets (small size SPL, having feature count less than 100). However, the UDMA-based solution cannot compete with ICPL for the largest dataset (eShop), with a significant difference. their work is intended to demonstrate the feasibility and effectiveness of UMDA in SPL pairwise testing, an experiment was conducted on small and moderate-sized SPLs.

## Conclusion and Future Work: -

Testing every product of a software product line (SPL) is often not feasible due to the huge number of derivable products. To alleviate this problem, numerous contributions have been proposed to reduce the number of products to be tested while still having good coverage. However, the main motivation for this work was to investigate the state-of-the-art in SPL testing, to determine what issues have been studied, and provide a guide to aid researchers in planning future research.

Seven papers are classified according to Approach, algorithms, and the type of testing that the research focuses on. The research covered some of the approaches that handle different and specific aspects in the SPL testing process (quality assurance, testability, tests reduction, Early defect detection, Fault correction, Commonality & variability, and effort reduction), which makes the studies comparison a challenging task since papers do not deal with the same goals or focus. Nevertheless, through this study, we can identify which activities are handled by the existing approaches as well as understanding how the researchers are developing work in SPL testing.

The study showed that in most papers, case studies report small projects, containing results obtained from in the company-specific application, which makes them impracticable in another context, due to the lack of details. This survey points out some topics that are new and need additional investigation, such as Delta-oriented product prioritization. Also, there are some limitations: one of them is that We cannot ensure that all the proposed approaches will provide the same results for larger feature models. Besides, the size of feature models and the constraints may affect the results.

As future work, it is recommended to apply the discussed approaches on enough large test cases in deferent real word SPLs to be able to generalize and make recommendations based on the findings. Also, it is reasonable to work on the implementation of a tool to fully automate the SPL Testing to ensure 100% quality-aware products.

## Reference:-

1. Yousaf, Nazish, Rida Sheikh, and Muhammad Abbas. "FIG based Quality Assurance in Software Product Lines." Frontiers of Information Technology (FIT), 2017 International Conference on. IEEE, 2017.
2. Alsariera, Yazan A., Mazlina A. Majid, and Kamal Z. Zamli. "SPLBA: An interaction strategy for testing software product lines using the bat-inspired algorithm." Software Engineering and Computer Systems (ICSECS), 2015 4th International Conference on. IEEE, 2015.
3. Cabral, Isis, Myra B. Cohen, and Gregg Rothermel. "Improving the testing and testability of software product lines." International Conference on Software Product Lines. Springer, Berlin, Heidelberg, 2010.
4. Al-Hajjaji, Mustafa, et al. "Delta-oriented product prioritization for similarity-based product-line testing." Variability and Complexity in Software Design (VACE), 2017 IEEE/ACM 2nd International Workshop on. IEEE, 2017.
5. Ma, Lei, et al. "Efficient testing of software product lines via centralization (short paper)." ACM SIGPLAN Notices 50.3 (2015): 49-52.
6. Al-Hajjaji, Mustafa, et al. "Similarity-based prioritization in software product-line testing." Proceedings of the 18th International Software Product Line Conference-Volume 1. ACM, 2014.

7. Sahid, MohdZanes, et al. "A Univariate Marginal Approach for Pairwise Testing of Software Product Lines." International Journal of Computer Applications 160.3 (2017).
8. Heimann, David I. "An Introduction to the New IEEE 730 Standard on Software Quality Assurance." Software Quality Professional 16.3 (2014): 26.
9. Kolb, Ronny, and Dirk Muthig. "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. ACM, 2006.
10. Jaring, Michel, Rene L. Krikhaar, and Jan Bosch. "Modeling variability and testability interaction in software product line engineering." Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on. IEEE, 2008.
11. Lee, Kwanwoo, Kyo C. Kang, and Jaejoon Lee. "Concepts and guidelines of feature modeling for product line software engineering." International Conference on Software Reuse. Springer, Berlin, Heidelberg, 2002.
12. Alam, Md Mottahir, Asif Irshad Khan, and Aasim Zafar. "A Comprehensive Study of Software Product Line Frameworks." International Journal of Computer Applications 151.3 (2016).
13. Henard, Christopher, Mike Papadakis, and Yves Le Traon. "Mutation-based generation of software product line test configurations." International Symposium on Search Based Software Engineering. Springer, Cham, 2014.
14. Knauber, Peter, et al. "Quantifying product line benefits." International Workshop on Software Product-Family Engineering. Springer, Berlin, Heidelberg, 2001.
15. Perrouin, Gilles, et al. "Automated and scalable t-wise test case generation strategies for software product lines." Software Testing, Verification and Validation (ICST), 2010 Third International Conference on. IEEE, 2010.
16. do Carmo Machado, Ivan, John D. McGregor, and Eduardo Santana de Almeida. "Strategies for testing products in software product lines." ACM SIGSOFT Software Engineering Notes 37.6 (2012): 1-8.
17. Engström, Emelie, and Per Runeson. "Software product line testing–a systematic mapping study." Information and Software Technology 53.1 (2011): 2-13.
18. Neto, Paulo Anselmo da Mota Silveira, et al. "A systematic mapping study of software product lines testing." Information and Software Technology 53.5 (2011): 407-423.
19. Nie, Changhai, and Hareton Leung. "A survey of combinatorial testing." ACM Computing Surveys (CSUR) 43.2 (2011): 11.
20. Kuhn, D. Richard, Dolores R. Wallace, and Albert M. Gallo. "Software fault interactions and implications for software testing." IEEE transactions on software engineering 30.6 (2004): 418-421.
21. Petke, Justyna, et al. "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing." Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013.
22. Brandon, Daniel M., ed. Software Engineering for Modern Web Applications: Methodologies and Technologies: Methodologies and Technologies. IGI Global, 2008.
23. Garvin, Brady J., Myra B. Cohen, and Matthew B. Dwyer. "Evaluating improvements to a meta-heuristic search for constrained interaction testing." Empirical Software Engineering 16.1 (2011): 61-102.
24. Chvatal, Vasek. "A greedy heuristic for the set-covering problem." Mathematics of operations research 4.3 (1979): 233-235.
25. Johansen, Martin Fagereng, Øystein Haugen, and Franck Fleurey. "Properties of realistic feature models make combinatorial testing of product lines feasible." International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, 2011.