Journal of Computer Science

Literature Reviews

Combinatorial Interaction Testing of Software Product Lines: A Mapping Study

[1]Mohd Zanes Sahid, [2]Abu Bakar Md Sultan, [2]Abdul Azim Abdul Ghani and [2]Salmi Baharom

[1]Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM), Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), Malaysia

Article history
Received: 22-03-2016
Revised: 17-10-2016
Accepted: 19-10-2016

Corresponding Author:
Mohd Zanes Sahid
Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM), Malaysia
Email: zanes@uthm.edu.my

Abstract: Software Product Line (SPL) is a software engineering paradigm that is inspired by the concept of reusability of common features, formulated for different software product. Complete testing on entire SPL is known to be unfeasible, due to the very large number of possible products to be produced, configured using a subset or all possible features in the SPL. This paper reports a Systematic Mapping Study (SMS) of relevant primary studies as the evidence on the application of Combinatorial Interaction Testing (CIT) for SPL. In CIT, one has to construct a covering array, which is a set of configurations having valid feature combinations and every combination of t features appears at least once in the array. This is also known as t-wise testing. By following the systematic mapping study guidelines, we have selected and filtered 44 primary studies for review. The most prominent CIT techniques in aiding SPL testing are those based on greedy algorithms followed by meta-heuristics algorithms. The motivation of SPL testing is to anticipate the feature interaction problem, in which the majority of the works were reported to leverage test configuration selection approach, while some employed test configuration prioritization approach. Numerous works have been reported, but only few works managed to demonstrate their scalability, as most primary studies only deal with low strength (t is less than 4) of t-wise testing.

Keywords: Systematic Mapping Study, Secondary Study, Combinatorial Interaction Testing, Software Product Line

Introduction

In real world, many software products developed for various domains carry some similar functionalities. These software share similar functionalities due to the fact that they have been developed based on the same kind of input and output types. The similarity in the internal program structure due to identical user requirements also contribute to the commonalities among these software. Because of this scenario and based on the benefit of reuse principles, Software Product Line (SPL) has been developed as a software development paradigm to produce software inspired by product line approach. Clements and Northrop (2002) defined SPL as follows:

"A software product line 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 manner" (Clements and Northrop, 2002).

In other words, SPL is a software engineering paradigm for creating a collection of similar software systems from a shared set of software assets using a well-defined production process. The main processes for SPL software development has been developed and refined over time by various researchers (Weiss and Lai, 1999; Czarnecki and Eisenecker, 2000; Thiel and Peruzzi, 2000). Two main processes in SPL are Domain Engineering and Application Engineering. In the Domain Engineering phase, few crucial activities are performed which includes, but not limited to, identifying systems domain and specifying commonalities and variabilities. In the Application Engineering phase, the products are configured and generated by utilizing software assets developed from previous phase. The most important principle that spans throughout the software engineering process is reusability based on the concept of function similarities.

Apart from similarities of functional properties among software, the differences of functionalities are

also of main concern, because only with these different functionalities, each software product is distinguishable from the other products and based on this commonalities and variabilities, each logical functionality is referred as feature (Lee *et al*., 2002) in SPL. Features can be defined as an abstraction of function, module or aspect of a system that represent a unit of program construction. Features can be generally classified into two types; core or compulsory features and product specific or optional features.

The process of identifying, defining and documenting these features based on the principles of commonalities and variabilities are known as variability modeling (Czarnecki *et al*., 2012). The most common variability model developed for SPL is Feature Modeling (FM) (Lee *et al*., 2002). It is a notation that represents features and its dependencies. Feature Models are normally visually presented using Feature Diagram in a form of tree structure, where nodes are the SPL features and edges are the relationship between features. Apart from Feature Modeling, features can also be modeled using Decision Modeling (DM) (Atkinson and Muthig, 2002) approach. Another more recent approach to variability model is Orthogonal Variability Model (OVM) (Lauenroth and Pohl, 2005) which focused on orthogonal features of an SPL.

SPL is also known as software product family, in which each individual software product can be generated from SPL using feature configuration process, where a number of relevant and suitable features are selected from the collection of all features based on the product requirements.

In correlation to that, an important characteristic in SPL that attracted significant attention among researchers is feature interaction (Calder *et al*., 2003). Feature interaction is a situation in which more than one features are combined and utilized together in a single configuration. This could result in an unspecified functionality and might lead to incorrect execution. Hence, it is crucial to test all possible feature configurations in order to reduce the potential misbehavior of interacting features. But, to test all possible feature configurations is unfeasible. In most trivial case, small number of features in a FM will results in small number of possible feature configurations. However, the number of feature configurations increase dramatically as the size of FM increased (Kim *et al*., 2011). Therefore, exhaustively testing all feature configurations for large-scale FM is not practical.

Applying existing testing techniques to each product separately is difficult and requires enormous resources (Reis *et al*., 2007). On most cases, with the presence of a moderate numbers of features, it will results in exponential number of feature interactions. Moreover, it might end up with redundant test effort. Based on that phenomenon, there have been quite a number of attempts to reduce the space of feature configuration testing through feature-based test configuration selection. Towards that, a generally accepted idea is to select a small subset of products where the maximum possible features interactions are most likely to occur. This is the main principle of Combinatorial Interaction Testing (CIT) such that not all input or configuration options contribute to every fault in a system. It is often the case that a fault is caused by interactions among a few inputs or small combination of configuration options.

Using CIT, one has to construct a Covering Array (CA). A CA is a two-dimensional array, where each column represents a software input/parameter (or *feature* in SPL context) and each row represent a test case (or test configuration in SPL context). The strategy is to construct the CA based on *t*-wise strength, where *t* indicates the coverage strength (1,2,3,..,*n*) and it will determine the number of feature combination that should appear at least once in the CA. For more details on CA, readers can refer to numerous literatures such as (Sloane, 1993; Kuhn *et al*., 2009; 2010).

CIT has successfully been applied in test input generation and parameter combinations of single product software development (Nie and Leung, 2011). One of the main strengths of CIT is that it enables a significant reduction of the number of test cases without compromising functional coverage. Similarly, in SPL, several works have been reported that apply and evaluate the effectiveness of CIT to reduce the testing effort by selecting a set of representative products (Johansen *et al*., 2012a; 2012d; Galindo *et al*., 2014).

This paper is structured as follows; in section 2, the details on the systematic mapping process are presented, which includes definition of the research questions, conducting the search process, filtering the evidences based on selection criteria and extracting particulars from selected primary studies into different categorizations. In section 3, results based on the extracted information are presented, followed by a discussion on the results in section 4. In section 5, threats to the validity of this review are presented, followed by remarks on related works in section 6. Finally, conclusion is presented in section 7 to wrap up this review study and suggest ideas for future endeavor.

Research Method

Overview

This study is conducted based on Systematic Mapping Study (SMS), a systematic process of planning, identifying, selecting, categorizing, analyzing and interpreting all available research evidence for a particular area of interest. SMS is originally established in medical research and has been adapted into software engineering field with some adjustments and extensions

(Kitchenham and Charters, 2007; Petersen *et al*., 2008). A systematic mapping study is applicable as a research method to investigate the current achievement in the respective area and useful in identifying regions that demand more studies to be conducted (Kitchenham and Charters, 2007). This technique is chosen because it can guide one to perform review by means of systematic process, repeatable steps and unbiased analysis. In the planning phase of this review, requirement of study and research questions are constructed and it is important as the driving factor of this whole review process. While the requirement of this study is to categorize all existing information about the adaptation of Combinatorial Interaction Testing (CIT) in Software Product Lines testing, the formulation of research questions are further described in the following sections.

Research Questions

Specialized approaches and techniques for testing SPL systems are deemed necessary because the development process is different from single product development. Some identified testing techniques are derived from Combinatorial Interaction Testing and Search-Based Testing (Harman *et al*., 2014), Specification Based Testing (Scheidemann, 2006), Logic Based and various hybrid testing techniques (Perrouin *et al*., 2010; Hervieu *et al*., 2011a; Henard *et al*., 2013; Al-Hajjaji *et al*., 2014; Sánchez *et al*., 2014a). Due to the availability of various approaches, it is the motivation of this mapping study to collect and review those works, which leads to the first research question:

RQ1: How CIT techniques have been adapted and applied in SPL Testing?

To further examine the first research question, the following sub-questions have been defined:

RQ1.1: What types of approaches are used to generate Covering Array (CA) for CIT?
RQ1.2: What is the mostly reported category of t-wise strength?
RQ1.3: Does constraint in feature model being handled explicitly?

Details on experiments conducted to the tool and their data sets are gathered in order to get a better understanding on the evaluation aspect of Combinatorial Interaction Testing techniques in SPL. Selection of data sets is crucial as it is used as subject for validation of the proposed technique. Techniques that have been implemented as a tool were identified and the performance of the proposed techniques is analyzed based on the measurement attributes employed by each study. To understand these, the following questions were raised in this review and will be answered in section 4:

RQ2: How was the proposed testing technique being evaluated?

It is deemed necessary to further elaborate the discussion on evaluation aspect by finding information with respect to:

RQ2.1: The size and type of data sets utilized (Industrial, Open-Source or Synthetic data sets)
RQ2.2: Name of other compared technique
RQ2.3: The evaluation metrics used to measure the performance

The final and more general evidence that this review is seeking to discover is in terms of the contribution made by the selected primary studies with respect to software testing effort reduction. Thus, it leads to the third main research question, as follows:

RQ3: What is the main contribution of the proposed technique with respect to test effort reduction?

In the next section, this paper reports the process that has been carried out during evidence searching, as this could assist readers to re-produce the review process whenever necessary.

Search Process

We obtained the main sources of primary studies for this mapping study from online databases. Five major online databases have been searched using specific search terms related to the criteria extracted from research questions. The search process was done separately on each database. The keywords used in the search string were constructed based on multiple keywords that are relevant to Combinatorial Interaction Testing. Table 1 defines the keyword used in the search process.

A series of filtering work have been performed to exclude papers that are not related to this mapping study. Two phases of initial filtering were applied in this search process. The first filter excludes papers based on its title, in which we excluded papers of type front matters, survey, overview, report and duplicate title. The second filter excludes papers based on its abstract, introduction and conclusion. We omitted those papers not within the scope of this mapping study, which is mainly focusing on CIT testing technique within the domain of software product line. Table 2 shows the details of the online database searched, with the distinct search string.

Table 1. Keyword types and values

Keyword type	Value	Alternative keywords
First Term	"software product line"	
Second Term	"combinatorial testing"	"combinatorial interaction testing" "CIT" "pairwise testing" "t-wise" "interaction testing"

Table 2. List of online databases searched with specialized search string

Online database	Search string
ACM Digital Library http://dl.acm.org	Search Type: Advanced Search Search Field and Values: ALL: "software product line" ANY: "combinatorial testing" "combinatorial interaction testing" "CIT" "pairwise testing" "t-wise" "interaction testing"
IEEE Explore http://ieeexplore.ieee.org/	Search Type: Command Search Search String: ("software product line") AND ((combinatorial ONEAR/2 testing) OR (pairwise ONEAR/1 testing) OR ("twise") OR (interaction ONEAR/1 testing))
Science-Direct http://www.sciencedirect.com/	Search Type: Expert Search Search String: ("software product line") AND ((combinatorial PRE/2 testing) OR (pairwise PRE/1 testing) OR ("t-wise") OR (interaction PRE/1 testing))
Springer Link http://link.springer.com	Search Type: Basic Search Search String: ("software product line") AND ((combinatorial ONEAR/2 testing) OR (pairwise ONEAR/1 testing) OR ("twise") OR (interaction ONEAR/1 testing))

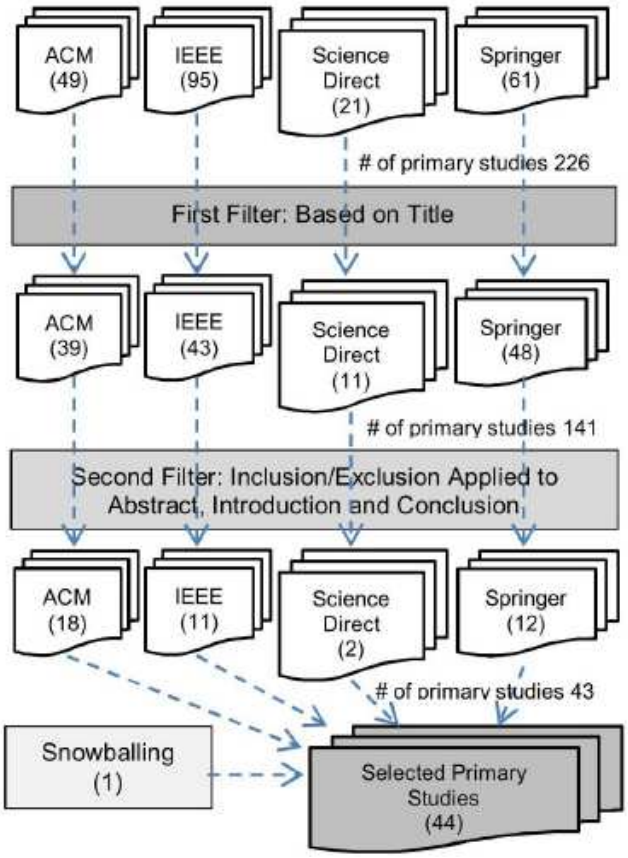

Fig. 1. Primary studies selection and filtering process

Selection of Primary Studies

After the relevant primary studies have been selected based on preliminary searching, we performed a more strict selection. This selection process is guided by two criteria called as inclusion and exclusion criteria (Kitchenham and Charters, 2007). Inclusion and exclusion criteria were defined based on the research question. The importance of inclusion and exclusion criteria is to ensure that the evidence can be reliably extracted and that they classify studies correctly.

Inclusion Criteria

All primary studies that focused on specific and clear contribution types in the form of method, model or strategies that adopted, adapted or improved CIT for SPL are included.

Exclusion Criteria

- All primary studies that focused on review on general SPL testing topics, mapping studies and open issues are excluded
- Exclusion is also made to primary studies on non-testing techniques or testing technique but does not employ CIT approach.
- Primary studies on CIT approach but does not meant for software product line domain are also excluded.

Selection Process

We performed two phases of search and selection work to filter only those relevant primary studies. The total number of results returned by online databases is 226 publications. Titles are identified and a number of publications are omitted due to either irrelevant or redundant. This first filtering process eliminates 85 papers, leaving 141 papers for next filtering phase. In the second filtering, we applied exclusion criteria and the process excluded papers based on its abstract, introduction and conclusion. About 98 publications that are not within the scope of this mapping study are excluded, hence shrank down the relevant publication to 43 papers. Finally, snowballing selection has been carried out, which results in inclusion of another one paper, making the total pertinent numbers of primary studies to 44 papers. The complete listing of selected primary studies is presented in Table 3. The selection process and results are depicted in Fig. 1.

The extraction and classification are based on seven categories of data attributes, which we defined as follows:

- *General*. The title, the authors, summary of the problem, gist of the contribution and publication venue
- *Test selection.* CA generation techniques, category of t-wise strength and constraint handling technique
- *Test prioritization.* Prioritization technique, prioritization criteria and prioritization goal
- *Parallelization*. Technique and infrastructure
- *Implementation*. Tool name
- *Data sets*. Name of data sets, nature and size
- *Evaluation*. Name of other technique compared, type of performance measurement

Table 3. List of the selected primary studies

Id	Title	Event/Publisher	Year	Reference
S1	A comparison of test case prioritization criteria for software product lines	ICST (IEEE)	2014	(Sánchez *et al.*, 2014a)
S2	A parallel evolutionary algorithm for prioritized pairwise testing of software product lines	GECCO (ACM)	2014	(Lopez-Herrejon *et al.*, 2014b)
S3	A systematic test case selection methodology for product lines: Results and insights from an industrial case study	ESE (Springer)	2014	(Wang *et al.*, 2014)
S4	A technique for agile and automatic interaction testing for product lines	ICTSS (Springer)	2012	(Johansen *et al.*, 2012c)
S5	An algorithm for generating t-wise covering arrays from large feature models	SPLC (ACM)	2012	(Johansen *et al.*, 2012a)
S6	An improved meta-heuristic search for constrained interaction testing	SSBSE (IEEE)	2009	(Garvin *et al.*, 2009)
S7	Automated and scalable t-wise test case generation strategies for software product lines	ICST (IEEE)	2010	(Perrouin *et al.*, 2010)
S8	Automated incremental pairwise testing of software product lines	SPLC (Springer)	2010	(Oster *et al.*, 2010)
S9	Bow tie testing: a testing pattern for product lines	EuroPLoP (ACM)	2012	(Johansen *et al.*, 2012b)
S10	Bypassing the combinatorial explosion: Using similarity to generate and prioritize T-Wise Test Configurations for Software Product Lines	TSE (IEEE)	2014	(Henard *et al.*, 2014)
S11	Combinatorial approach for automated platform diversity testing	ICSEA (IEEE)	2009	(Sisodia and Channakeshava, 2009)
S12	Combinatorial test generation for software product lines using minimum invalid tuples	HASE (IEEE)	2014	(Yu *et al.*, 2014)
S13	Combinatorial testing for feature models using CitLab	ICSTW (IEEE)	2013	(Calvagna *et al.*, 2013)
S14	Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines	CEC (IEEE)	2014	(Lopez-Herrejon *et al.*, 2014a)

Table 3. Continue

S15	Constructing interaction test suites for highly-configurable systems in the presence of constraints-a greedy approach	TSE (IEEE)	2008	(Cohen *et al.*, 2008)
S16	Constructing test cases for n-wise testing from tree-based test models	SoICT (ACM)	2013	(Do *et al.*, 2013)
S17	Cost-effective test suite minimization in product lines using search techniques	JSS (ScienceDirect)	2015	(Wang *et al.*, 2015)
S18	Covering SPL behaviour with sampled configurations: An initial assessment	VaMoS (ACM)	2015	(Devroey *et al.*, 2015)
S19	Evaluating improvements to a meta-heuristic search for constrained interaction testing	ESE (Springer)	2011	(Garvin *et al.*, 2011)
S20	Feature interaction testing of variability intensive systems	PLEASE (IEEE)	2013	(Patel *et al.*, 2013)
S21	Generating better partial covering arrays by modeling weights on sub-product lines	MODELS (Springer)	2012	(Johansen *et al.*, 2012d)
S22	Industrial evaluation of pairwise SPL testing with MoSo-PoLiTe	VaMoS (ACM)	2012	(Steffens *et al.*, 2012)
S23	Interaction testing of highly-configurable systems in the presence of constraints	ISSTA (ACM)	2007	(Cohen *et al.*, 2007)
S24	Minimizing test suites in software product lines using weight-based genetic algorithms	GECCO (ACM)	2013	(Wang *et al.*, 2013)
S25	Model-based pairwise testing for feature interaction coverage in software product line engineering	SQJ (Springer)	2012	(Lochau *et al.*, 2012)
S26	MoSo-PoLiTe: Tool support for pairwise and model-based software product line testing	VaMoS (ACM)	2011	(Oster *et al.*, 2011b)
S27	Multi-objective optimal test suite computation for software product line pairwise testing	ICSM (IEEE)	2013	(Lopez-Herrejon *et al.*, 2013)
S28	Multi-objective test generation for software product lines	SPLC (ACM)	2013	(Henard *et al.*, 2013)
S29	PACOGEN: Automatic generation of pairwise test configurations from feature models	ISSRE (IEEE)	2011	(Hervieu *et al.*, 2011b)
S30	Pairwise feature-interaction testing for SPLs: Potentials and limitations	SPLC (ACM)	2011	(Oster *et al.*, 2011a)
S31	Pairwise testing for software product lines: Comparison of two approaches	SQJ (Springer)	2012	(Perrouin *et al.*, 2012)
S32	Practical pairwise testing for software product lines	SPLC (ACM)	2013	(Marijan *et al.*, 2013)
S33	Properties of realistic feature models make combinatorial testing of product lines feasible	MODELS (Springer)	2011	(Johansen *et al.*, 2011b)
S34	PROW: A pairwise algorithm with const raints, order and weight	JSS (ScienceDirect)	2015	(Lamancha *et al.*, 2015)
S35	Reusable Model-Based Testing	CAiSE (Springer)	2009	(Olimpiew and Gomaa, 2009)
S36	Similarity-based prioritization in software product-line testing	SPLC (ACM)	2014	(Al-Hajjaji *et al.*, 2014)
S37	Strategies for product-line verification: Case studies and experiments	ICSE (IEEE)	2013	(Apel *et al.*, 2013)
S38	Testing a data-intensive system with generated data interactions	CAiSE (Springer)	2013	(Sen and Gotlieb, 2013)
S39	testing product generation in software product lines using pairwise for features coverage	ICTSS (Springer)	2010	(Lamancha and Usaola, 2010)
S40	Testing variability-intensive systems using automated analysis: An application to Android	SQJ (Springer)	2014	(Galindo *et al.*, 2014)
S41	The Drupal framework: a case study to evaluate variability testing techniques	VaMoS (ACM)	2014	(Sánchez *et al.*, 2014b)
S42	Towards efficient SPL testing by variant reduction	VariComp (ACM)	2013	(Kowal *et al.*, 2013)
S43	Using feature model knowledge to speed up the generation of covering arrays	VaMoS (ACM)	2013	(Haslinger *et al.*, 2013)
S44	Variability testing in the wild: the Drupal case study	SoSyM (Springer)	2015	(Sánchez *et al.*, 2015)

Event/Publisher Description:

CAiSE - International Conference on Advanced Information Systems Engineering
CEC - Congress on Evolutionary Computation
ESE - Journal of Empirical Software Engineering
EuroPLoP - European Conference on Pattern Languages of Programs
GECCO - The Genetic and Evolutionary Computation Conference
HASE - International Symposium on High-Assurance Systems Engineering
ICSE - International Conference on Software Engineering
ICSEA - International Conference on Software Engineering Advances
ICSM - International Conference on Software Maintenance
ICST - International Conference on Software Testing, Verification and Validation
ICSTW - International Conference on Software Testing, Verification and Validation Workshops
ICTSS - International Conference on Testing Software and Systems
ISSRE - International Symposium on Software Reliability Engineering
ISSTA - International Symposium on Software Testing and Analysis
JSS - The Journal of Systems and Software
MODELS - International Conference on Model Driven Engineering Languages and Systems
PLEASE - International Workshop on Product Line Approaches in Software Engineering
SPLC - International Software Product Line Conference
SQJ - Software Quality Journal
SSBSE - International Symposium on Search Based Software Engineering
SoICT - International Symposium on Information and Communication Technology
SoSyM - Software & Systems Modeling
TSE - Transactions On Software Engineering
VaMoS - International Workshop on Variability Modeling of Software-Intensive Systems
VariComp - International Workshop on Variability and Composition

Results

Various approaches have been reported and evaluated towards feature configuration testing of SPL systems. Most solutions were mainly built using the Combinatorial Interaction Testing approach, followed by its integration with other optimization approach such as Search-Based and Logic-Based. This section presents what have been achieved so far and how it was done in terms of effective utilization of CIT approach in SPL domain.

Adaptation and Application of Combinatorial Interaction Testing to SPL Testing

Combinatorial Interaction Testing (CIT) is one of the most common and promising test configuration selection approach employed to reduce the number of selected products hence the test configuration in SPL testing (Lamancha *et al*., 2013). The goal of test configuration selection is to reduce the set of feature combinations to a reasonable but representative set of products achieving a high coverage of feature interactions. Test configuration are selected in a way that guarantees that all combinations of *t* features are tested, this is called as *t*-wise testing (Perrouin *et al*., 2010). One of the well-known variants of Combinatorial Interaction Testing is the 2-wise (or pairwise) testing approach (Lamancha and Usaola, 2010). In pairwise testing, one generates all possible combinations of pairs (two) of features based on the observation that most faults originate from interaction of two features.

Following the CIT strategy, one has to construct a Covering Array (CA) that consists of complete or partial *t*-wise sub-array where *t* is defined as the strength of feature combination. The strength is simply the number of features considered or chose to be the subject of testing. For pairwise/2-wise, the strength is 2, 3-wise having strength of 3 and so on. Generating such CA is known as NP-hard problem (Johansen *et al*., 2011a), because the number of possible features to be combined grows exponentially with the number of features designed for a particular SPL. For this reason, there is a need for an efficient and practical strategy for *t*-wise generation in order to get the most optimum set of combinations within an affordable testing cost. Thus, there is a trade-off between the completeness of *t*-wise test generation and minimization of testing effort. One might settle with partial *t*-wise test generation in order to achieve acceptable and affordable cost of testing, especially for large SPLs. This obstacle has driven many to search for viable approach to construct covering array.

Types of Approaches Employed to Generate Covering Array (CA) Generation Techniques and T-Wise Strength

There are five groups of CA generation techniques as presented in Table 4. The most prominent approach, which is based on greedy algorithm, are proposed in 25 primary studies. It is followed by meta-heuristics algorithms, employed in 11 primary studies. Three works are categorized in constraint programming approach, whereas two and one primary studies are reported in divide and compose and integer programming technique, respectively. Refer to Table 3 for the primary studies title and citation.

As previously mentioned, an important attribute in CIT is the strength of *t*-wise. The strength should be carefully selected during the construction of covering array in CIT as it will determine the extensiveness of combination or interaction of different features; that eventually will be evaluated in the testing process. The most investigated strength reported in the selected primary studies is pairwise (2-wise) strength, which accounted for 55% (24) from all reviewed work. 13 publications (30%) have empirically evaluated their solutions for up to 3-wise covering array. Three publications (7%) reported the evaluation for up to 4-wise covering array and only two works managed to scale their work for up to 6-wise strength.

Most works claimed that they able to scale for higher strength (t>=4), however only five works are proven to be viable. Although it is proven that lower strength of covering array (t<4) reveals most faults, bear in mind that higher strength of covering array (t>=4) could reveal residual faults especially for large SPL systems with high number of features. Here we define the notion of residual faults as the remaining faults that are not detected or revealed by CIT exercising *t*-wise of strength less than 4.

In terms of overall trend, it seems that greedy-based algorithms dominated the CA generation for up to 3-wise. For higher strength, based on the evidence, it seems difficult, if not impossible, to generate CA, since only single work is reported for each meta-heuristics and greedy algorithms.

Techniques to Handle Constraints in Feature Model

An important attribute in a Feature Model, apart from features and its relationship (*mandatory*, *optional*, *or*, *exclusive or*), is feature constraints (*requires*, *excludes*). Constraints are used to define relationships between features that are difficult, if not impossible, to be sketched in the Feature Diagram. The presence of constraint is unavoidable as it determines the usability and practicability of an SPL. Relationships and constraints can normally be specified using Conjunctive Normal Form (CNF) defined using

Boolean Logic using AND, NOT and OR. Table 5 shows the mapping table used to transform the Feature Model into CNF.

The Feature Diagram in Fig. 2 depicted how constraints can be included in the definition of a Feature Model. One can use either Propositional Logic, or Boolean Logic, or both to define the feature constraints.

On most cases, one can reduce the number of feature combinations if constraints are introduced in the Feature Model. Constraints impose significant influences to a Feature Model. It can make a number of feature configurations invalid with respect to some strict requirements of an SPL.

Since constraint handling is crucial towards selection of valid feature configuration, it is therefore beneficial to extract and report all available techniques to handle constraint in this review work. The result in Table 6 shows that only 26 primary studies (59%) mentioned or dealt with constraint handling in guiding them to generate valid feature configurations. The most frequently employed technique is using Boolean satisfiability (SAT) solver. A total of 17 primary studies employed this technique; where mostly (nine) works apply this with Greedy covering array generation technique. One paper reported for each Model Checker and Invalid Tuple approach. There is also one paper suggesting to manually specify the rule, given by the domain expert. The remaining six papers mentioned about including constraint handling treatment but did not explicitly specify the details.

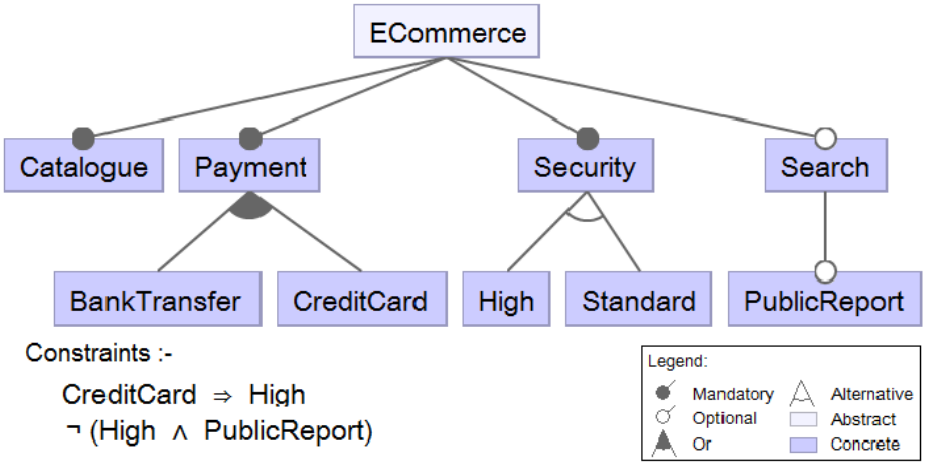

Fig. 2. An example of a constrained feature diagram for ECommerce SPL

Table 4. Distribution of primary studies based on covering array generation techniques and t-wise strength

	CA generation techniques				
t-wise Strength	Integer programming	Divide and compose	Constraint programming	Meta-heuristics	Greedy
Pairwise/2-wise	S27	S7, S38	S29, S32, S40	S1, S2, S3, S6, S14, S17, S24, S28	S4, S8, S13, S20, S22, S25, S26, S34, S39, S42
3-wise				S23	S5, S9, S11, S12, S15, S18, S21, S30, S36, S37, S41, S43
4-wise				S19	S33, S44
6-wise				S10	S16

Table 5. Propositional and Boolean logic mapping table for relationship and constraints

Relationship	Propositional logic	Boolean logic
Optional	$f_i \Rightarrow f_j$	$\neg f_i \vee f_j$
Mandatory	$f_i \Leftrightarrow f_j$	$(\neg f_i \vee f_j) \wedge (f_i \vee \neg f_j)$
Or	f_i OR f_j	$f_i \vee f_j$
Exclusive Or	f_i XOR f_j	$(\neg f_i \wedge f_j) \vee (f_i \wedge \neg f_j)$
Constraint		
Requires	$f_i \Leftrightarrow f_j$	$\neg f_i \vee f_j$
Excludes	f_i XOR f_j	$\neg (f_i \wedge f_j)$

Table 6. Employment of constraint handling and covering array generation techniques

	Covering array generation techniques		
Constraint handling techniques	Meta-heuristics	Greedy	Integer programming
SAT	●●●●●●●	●●●●●●●●●	●
Model checker		●	
Invalid tuple		●	
Given rule		●	
Unnamed	●	●●●●●	

Evaluation of the Proposed Techniques

Since most of current works are focusing on lower strength (less than 4) of *t*-wise combinations of features (Henard *et al*., 2014), it is interesting to ask whether this phenomenon has any co-relation with the size of the data sets that were utilized in the experiments.

Type and Size of Data sets Utilized (Industrial or Open-Source Data Sets)

Currently, a wide sets of data or case studies have been developed and published publicly by the community. In this review, those data sets that are published publicly are considered as open source data sets. Else, it is considered as industrial data sets. The open source data sets were either pulled out from academic publications or other repositories. A number of open source repositories that contain various types and sizes of feature models are identified. The main purpose of the repository is to encourage knowledge sharing among research community members and ultimately improve research quality. The five main repositories are SPLOT, Reverse Engineering Feature Models, SPL CONQUEROR, Feature House and SPL2GO. Those anonymous repositories are categorized as Others.

Based on the selected primary studies, there are a total of 111 different data sets utilized. About 89% (99) of all data sets are coming from open source feature models. About 12 data sets are identified as industrial data sets. For open source data sets, majority of the primary studies validated their works on lower strength (2 and 3-wise) CIT, which accounted for 80% (79 data sets). Only four data sets (Cellphone, Linux, FreeBSD and eCos) have been utilized against 6-wise and the remaining 17 data sets with 4-wise. This shows that current techniques that were evaluated against open source data sets are somehow limited to lower strength t-wise. It is to our surprise that only single technique (S10) managed to generate 6-wise for the biggest open source data set i.e., Linux (having 6888 number of features). The trend for industrial data sets corroborated this phenomenon, where only single (out of 12) data set, named as OSEK-OS, is reported to be utilized in measuring 6-wise CIT for SPL. Nearly 60% of all industrial data sets were reported to be utilized in measuring only 2-wise CIT.

Looking at the size of data sets, only four data sets (Linux, GCC_2, FreeBSD and eCos) are considered as large in this review, which is having number of features greater than 1000. The majority of the data sets which is 68% (76 data sets) is coming from small data sets having number of features less than or equal to 50.

Comparison and Evaluation Metrics

A number of tools implementing their respective techniques for CIT-based SPL testing are available. Some of the earliest published tools are mAETG (S23), CASA (S6) and MoSo-PoLiTe (S8). It is one the objective of this review to find all evidence of the empirical works that perform comparison between the proposed solution in each primary studies with some other tool (s). This is important as it provides proof on the performance of the proposed technique with respect to other earlier techniques.

Based on the information presented in Table 7, the most frequently chosen techniques that were used as comparison are variant of CASA (in 5 primary studies), variant of ICPL in 4 primary studies and variant of AETG in 3 primary studies. Most of the papers (6 studies) performed the comparison with random technique, whereas three studies employ human knowledge in defining the selection of test cases, to be compared with their proposed techniques. Apart from that, it is also to our surprise that some of the works had been using a number of non-SPL based testing tools (ACTS in 4 primary studies, PICT in 3 primary studies, TestCover in two studies, Chvatal and Jenny each in single primary study) in their empirical works.

To gauge the achievement of current techniques, we also collect information on the evaluation metrics that were used to measure the performance. Table 7 and Table 8 shows 12 different evaluation metrics extracted from the selected primary studies. The most frequently employed evaluation metrics are Covering Array Size (CAS) and Overall Execution Time (OET), in which each accounted for almost 30% of all evaluation metrics. The Feature Pairwise Coverage (FPC) metrics appeared in 7 papers and as the name implies, it is only used in pairwise CIT testing. Apart from FPC, other evaluation metrics that measure the performance in terms of covering array size are Test Minimization Percentage (TMP) (in 3 papers) and *t*-wise Coverage (TWC) (in 4 papers).

Table 7. List of tool names, compared techniques and evaluation metrics

Primary Studies	Tool name/ Algorithm	Compared With	Evaluation metrics												
			AEF	APFD	CAS	Cost	FDC	FPC	GD	HV	OET	TCS	TMP	TWC	Other
S1	Enhanced-SPLAR	Random		√											
S2	PPGS	pICPL			√						√				
S3	IPT (Import Plugin and Transformation)	Manual	√				√	√			√		√		
S4	CVL-based Eclipse Plugin	none													√
S5	ICPL	Alg.1, CASA, ACTS, MoSo-PoLiTe			√						√				
S6	CASA	mAETG (modified AETG)			√						√				
S7	*Perrouin*	none												√	
S8	MoSo-PoLiTe	none			√						√				
S9	*Johansen*	none			√										
S10	Enhanced-SPLCAT	ACTS, CASA, SPLCAT, CASA-n			√										√
S11	*Sisodia*	Jenny												√	
S12	LOOKUP	PICT, SPLCA			√						√				
S13	CitLab	ACTS			√						√				
S14	*Lopez-Herrejon*	none			√			√	√	√	√				
S15	*Cohen*	AETG			√						√				
S16	FOT-nw	FOT			√										
S17	TEMSA	Random	√				√	√			√		√		
S18	VIBeS	none			√										
S19	Enhanced CASA	mAETG (modifiedAETG)			√						√				
S20	MPFM	none					√				√				
S21	pICPL	none													√
S22	MoSo-PoLiTe	none			√						√				
S23	mAETG and SA_SAT	PICT, TestCover			√						√				
S24	*Wang*	Random	√				√	√			√		√		
S25	*Lochau*	none			√						√				
S26	MoSo-PoLiTe	none													√
S27	*Lopez-Herrejon*	none			√						√				
S28	*Henard*	Random			√	√		√							
S29	PACOGEN	MoSo-PoLiTe			√										
S30	MoSo-PoLiTe	none			√										
S31	*Perrouin*	none			√						√	√		√	
S32	Enhanced PACOGEN	Manual Technique						√							
S33	*Johansen*	none			√						√				
S34	PROW	PICT, TestCover ACTS, CASA			√						√				
S36	Enhanced-FeatureIDE	ICPL, CASA, CHVATAL, Random					√								
S37	SPLVERIFI ER	none					√				√				
S38	FAKTUM	Manual					√				√				
S39	Customizable AETG	none													√
S40	TESALIA	Random													√
S41and S44	*Sánchez*	none		√											
S42 and S43	Enhanced ICPL	ICPL			√						√				

Note: Italicised tool/algorithm names are unavailable, hence name of the first author is used.

Acronyms:

AEF: Average Execution Frequency
APFD: Average Percentage of Faults Detected
CAS: Covering Array Size
FDC: Fault Detection Capability
FPC: Feature Pairwise Coverage
GD: Generational Distance
HV: Hypervolume
OET: Overall Execution Time
TCS: Test Config Similarity
TMP: Test Minimization Percentage
TWC: t-wise Coverage

Table 8. Evaluation metrics

Evaluation metrics	Count of usage	%
AEF	3	3.53
APFD	4	4.71
CAS	24	28.24
Cost	1	1.18
FDC	6	7.06
FPC	7	8.24
GD	1	1.18
HV	1	1.18
OET	24	28.24
TCS	1	1.18
TMP	3	3.53
TWC	4	4.71
Unknown	6	7.06

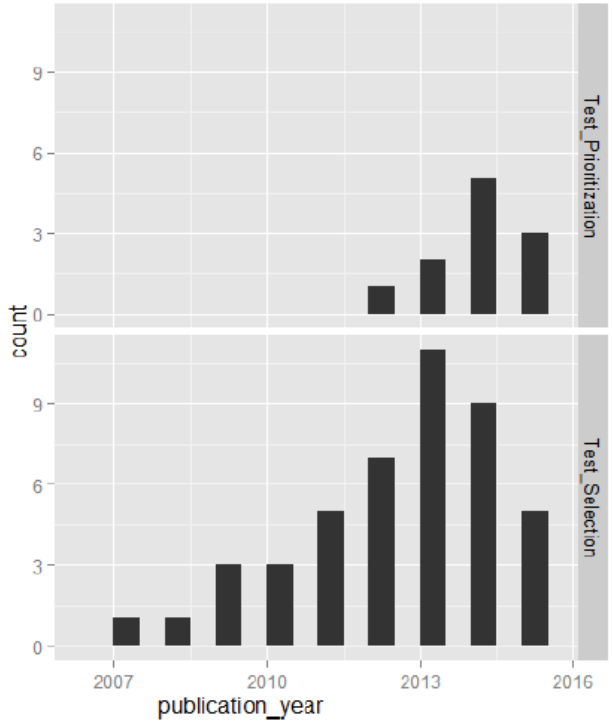

Fig. 3. Distribution of studies based on test selection or test prioritization

Two evaluation metrics that measure the number of faults are Average Percentage of Faults Detected (APFD) and Fault Detection Capability (FDC). APFD appeared in 4 papers, whereas FDC appeared in 6 papers.

Two measurement metrics, unique to multi-objective problems, that were employed to measure the quality of Pareto fronts are Hypervolume (HV) and Generational Distance (GD) and they appeared only in one paper. The remaining two evaluation metrics are Test Config Similarity (TCS) and Average Execution Frequency (AEF) which appeared in one and three primary studies, respectively.

In terms of the number of evaluation metric techniques employed, 20 primary studies use two evaluation metrics, 18 studies use one evaluation metrics, four primary studies use five evaluation metrics and one primary studies employed one evaluation metrics each.

Contribution of the Proposed Techniques

Many have proposed combinatorial interaction testing techniques that employ systematic selection, guided prioritization, or combination of both and this review managed to capture those papers, as per summarized in Fig. 3. Test selection has long been the driving motivation for SPL combinatorial testing, as early as the year 2007 with one primary study reported. The quantity starts to increase to five studies in 2011 and keeps increasing to 11 studies in 2013. On the other hand, test prioritization only appears under combinatorial testing for SPL in year 2012 with single publication. The study doubles in 2013 and in year 2014 the reported published studies peaked to five. The number gets reduced to three studies in 2015.

Test Configuration Selection

As previously mentioned, the obstacle faced in selecting test configuration in SPL is due to the huge number of possible test configuration that can be constructed even for medium size SPL systems. The problem was compounded by the difficulty in handling constraints, exhibit in the feature model, as the presence of constraints are un-avoidable. Constraint handling in test configuration selection first appeared in year 2007 by S23 (tool named mAETG). A lengthy discussion on constraint handling techniques was presented and they introduced the concept of forbidden tuples. mAETG was extended in S15 in 2008 and they exploited the current covering array generation with an open constraint handling technique called as Constraint Covering Array (CCA). The building block of mAETG was based on Simulated Annealing (SA). Similarly, in primary study S6, CASA has been fabricated using SA, a year later, whereby two improvements have been proposed in CASA, as (*i*) modified strategy for selecting sample size and (*ii*) changing the neighborhood of current solution. CASA has been extended in S19 with more thorough evaluation.

In 2010, primary study S39 proposed an adaptation of non-SPL CIT technique to tackle test configuration in SPL, coined as Customizable AETG. During the same year, primary study S7 proposed a systematic way to sample small sets of test cases, using “divide-andcompose” strategy. It splits t-wise combinations into solvable subsets. Then each subset is solved using constraint solver. S8 and S26 proposed a tool named as MoSo-PoLiTe, implementing technique that combines graph transformation, CIT and forward checking. It was based on the notion of applying transformation of feature models allows a simpler processing of SPL model

especially in a more complex SPL. MoSo-PoLiTe was further extended in S30, implemented Combinatorial Design (CD) approach that combines pairwise with model-based testing by transforming feature models into state charts. An industrial evaluation of combinatorial SPL testing using MoSo-PoLiTe was performed at Danfoss Power Electronics A/S and reported in S22.

In 2011, PACOGEN introduced in S29 and proposed a definition of global constraint and filtering algorithm to select only valid test configurations. It was further enhanced in S32 and validated on industrial setting of Video Conferencing SPL (VCSPL). Apart from that, Johansen *et al*. have published S33, in which a technique based on Chvátal and SPLAR has been proposed. It was empirically tested with numerous data sets and managed to generate covering array of t-wise with strength up to 4, of which many state-of-the-art techniques failed to achieve. Later in 2012, primary study S25 had proposed the mapping of feature models to state charts and the mapping was used as test model to generate test configuration. Meanwhile, S4 proposed a CVL-based Eclipse plugin that integrates SPL and Agile based system. They proposed a solution able to deal with compatibility issues among features in continuous integration phase.

Driven by the promising achievement in S33, Johansen *et al*. extended their work in S5. They published the tool named ICPL, capable of processing large feature models, better execution time and most importantly produced small covering array. They used the fact that a (t-1)-wise is always a subset of the t-wise, thereby creates lower strength recursively to build up a higher strength covering array. Extending the framework of ICPL, S43 introduced two reduction rules to eliminate some features from the FM to enhance CA generation process. Similarly, by extending ICPL, S42 proposed a filter to reduce set of features and suggested extension of feature models with three attributes, which is shared resources, communication channels utilized and feature priorities.

A year later, apart from MoSo-PoLite, another work reported in S20 (tool named MPFM), also dealt with industrial-based SPL, called as Site X. Their approach is quite similar to the notion of regression testing, where their main concern was to improve the process of testing an entire product during software evolution. Their main contribution was the introduction of separation of concerns (modularization) of feature models into multiple perspectives. Features from FM are grouped into multiple perspectives. Instead of using feature as parameter and true/false as values, MPFM use perspective as parameter and features of perspective as values. MPFM stands for Multi-Perspective Feature Models.

CitLab has been reported in S13 as a Combinatorial Interaction Model (CIM) that implements simplification of constraints thru elimination, which reduce the time required by the constraint checking process. Another transformation-based technique proposed in S16, which transform feature model into extended logic tree.

Having many testing goals in test configuration selection is sometimes unmanageable. The goals might go against each other. To overcome this issue, study S28 utilized a genetic algorithm to handle multiple conflicting objectives in test configuration selection for SPLs. They formulate the pairwise coverage, number of products and testing cost as three objective functions. Generally, they dealt with SPL testing by exploring the possibility of achieving multi-objective CIT optimization. Another work, S24, also employ multi objectives fitness function to optimize CIT based on genetic algorithm. It is extended as a tool named IPT in primary study S3, which proposed Component Family Model (CFM) that provides traces between test cases and feature models. IPT has been validated in an industrial setting of SATURN SPL.

Another approach proposed in S27 employs integer programming in test configuration selection to anticipate the trade-offs between maximizing test coverage and minimizing test suite size, based on the non-domination of any testing objectives. A more recent work in S14 proposed similar multi-objectives optimization based on the work of CASA but with improvement in seeding strategy. They suggested the seeding strategies to be based on three information; (i) test suites size, (ii) test suites that were generated using greedy algorithm and (iii) test suites that were generated using an existing single-objective pairwise testing approach.

A tool chain called as SPLVERIFIER was published in S37. It has the capability to select product for testing using either product-based, sample-based or family based model checking. They claimed that family-based model checking allows for better fault detection compared to the other two. Meanwhile, FAKTUM, published in S38 was the only work that attended the less studied area of SPL testing which is testing the data intensive SPL system. They proposed a divide-and combine strategy to tackle feature interaction problem using Generated Data Interactions.

Primary study S12 published a tool named LOOKUP in 2014, which they claimed employed an efficient algorithm based on validity checking using minimum invalid tuples (MITs). Recently, in 2015, TEMSA has been published in S17, which acts as a recommender tool that suggests an appropriate meta-heuristics algorithm based on the selected objective function. It supports three families of meta-heuristics algorithms, i.e., Evolutionary Algorithm (EA), Particle Swarm Optimization (PSO) and Cellular

Genetic Algorithm + Differential Evolution (CellDE). They defined five objective function that user can select, i.e., Test Minimization Percentage (TMP), Feature Pairwise Coverage (FPC), Fault Detection Capability (FDC), Overall Execution Time (OET) and Average Execution Frequency (AEF). Another work, referred here in S18, proposed a feature transition system that can be used to evaluate behavioral coverage of a particular test configuration. Their approach has been implemented in VIBeS.

Test Configuration Prioritization

The employment of test prioritization approach driven by Combinatorial Interaction Testing for SPL first appeared in year 2012. Primary study S21 (with tool named pICPL) focused on industrial setting of TOMRA SPL, where the main problem that they were dealing with was unrealistic effort in generating test configuration. Their most significant contributions were to generate covering array by weight-based prioritization of feature interactions. Weight is calculated based on the number of product instances (exists in the market), which results in fewer and more realistic set of product to be tested. This allows the same covering array to be generated on every execution.

Similarly, weight-based criteria had been exploited by MPFM (S20), TESALIA (S40), S41, S44 and PROW (S34) to achieve test configuration prioritization. The weight for features is determined by a given feature important ratings (S20). For TESALIA, weight is assigned by feature value and cost, in which cost is defined explicitly as the number of configuration that include a specific feature. Cost per product instance is calculated based on the number of concrete features, whereas value is calculated based on the market share of each feature. For S41, prioritization of test configuration is achieved by analyzing historical faults. Faults in Drupal system are captured in the project's issue tracking system. The issue tracking system of two Drupal versions were manually searched in order to extract faults in system evolution. Faults were mapped with features and the higher the number of faults, the higher the weight of a particular feature towards prioritization. In an extension to S41 (S44), apart from change driven weight assignment based on faults, they also proposed product complexity criteria, product dissimilarity criteria and non-functional properties of features to guide in the prioritization process. Recently, S34 introduced the notion of pair weight to mark pairs of features that are more significant to be tested, thereby imposing prioritization. However, weight is pre-assigned based on testers' knowledge, hence it prioritization is driven by human factor.

Meta-heuristic techniques have been proposed in S2 (PPGS), S28, S1 and S10 to further improve test configuration prioritization results. In S28, multiobjective optimization approach has been leveraged as their prioritization technique. The criteria for prioritization that they proposed were to maximize pairwise coverage and minimize number of products and cost. Another approach on prioritization that were based on genetic algorithm is cited here as S1 (Enhanced-SPLAR). The approach employs five criteria to assist in test configuration prioritization, i.e., (*i*) Cross-Tree-Constraints Ratio (CTCR), (*ii*) Coefficient of Connectivity-Density (CoC), (*iii*) Variability Coverage and Cyclomatic Complexity (VC&CC), (*iv*) Number of Reused Features and (*v*) Product Dissimilarity. As opposed to multi-goal in S28, S1 aimed at achieving single goal, which is to detect fault as early as possible.

A more recent study reported here (S10) take advantage of the simplified evolutionary algorithm, (1+1) EA, with single population size, no crossover operator and simple bit-wise mutation operator. S2 proposed parallel genetic algorithm in 2014. The criteria for prioritization proposed by them are based on nonfunctional criteria, which consist of estimation of performance, memory consumption and footprint. They also utilized product dissimilarity criteria, based on ranking, as opposed to S1 and S10, which employs Jaccard Distance algorithm. Product dissimilarity has also been applied by S36 using Hamming Distance algorithm. Prioritization goals and prioritization criteria for the relevant primary studies are summarized in Table 9.

Parallelization of the Process

Realizing on the set back of the scalability aspect in terms of covering array generation time, few primary studies are already moving one step further by accelerating the process with parallelization. Two categories of parallelization have been identified. The first one is based on cluster, proposed by three primary studies S2, S14 and S27. The second category is based on data parallel, employed by primary study S5 (ICPL). Parallelization based on clusters requires the availability of tens to hundreds of physical dedicated machine, which logically being managed by a cluster manager. Due to its more simpler implementation and deployment, cluster based parallelization is more favorable than data parallel, which require more customized codes and settings. It is interesting to highlight that all the three studies under cluster based parallelization only managed to achieve t-wise of strength 2, whereas the data parallel approach employed in ICPL managed to scale up to 3 twise strength. We also found that this parallel version of ICPL is the only tool that is capable of generating covering array for Linux data set, the largest data set reported in this review.

Table 9. Prioritization criteria and prioritization goals

Prioritization criteria							Prioritization goals					
Primary study	Product complexity	# of reused Features	Products dissimilarity	Non-functional	Weight-Based	Other	Early fault detection	Higher fault detected	Small covering array	Maximize weight Coverage	Maximize T-Wis Coverage	Other
S1	CTCR, CoC, VCCC	Yes	Jaccard distance			Random	Yes					
S2			Ranked-Based	Performance, memory consumption, footprint		Random	Yes		Yes			
S10			Jaccard distance								Yes	
S20					Feature importance ratings			Yes	Yes			
S21					Product instances					Yes		
S28						Random			Yes	Yes	Yes (Pairwise)	Minimize cost
S34					Knowledge based					Yes		
S36			Hamming distance				Yes					Maximize interaction coverage
S40					Feature value and cost					Yes		
S41					Historical faults		Yes					
S44	VC,CC		Jaccard distance	Size	Historical faults		Yes					

Discussion

Covering Array Generation Techniques

The influence and effectiveness of search-based techniques in SPL combinatorial testing is still under-explored, because there are still lack of work in employing metaheuristics algorithms in covering array generation as compared to greedy algorithms for *t*-wise testing of strength less than 4. On the other hand, there are some evidence that show the viability of both approach in generating CA for higher strength of t-wise. However, all works are dealing with only uniform strength of t-wise covering array. They set a fixed and single value of *t* during the covering array construction, hence exercising uniform combination of features. An alternative or perhaps a complementary strategy is to consider a varying number of *t*, which is called as *variable strength* of t-wise. This strategy is widely accepted in non-SPL testing. Interactions do not exists uniformly between inputs or parameters (Nie and Leung, 2011). Some inputs or parameters will have strong interactions with other parameters, while some others may have few or no interactions. Similarly in SPL, some group of features have more critical processes and requires much more features to support its operations as oppose to other less critical features. While lower t-wise strength might be sufficient in testing the less critical features, a higher t-wise strength could be needed to effectively test the more critical features. To the best of this review process, none of current state-of-the-art of CIT approaches in SPL deal with variable strength of combinatorial testing. This has to be further investigated as it could possibly improve the effectiveness of SPL testing.

Data Sets Size

The results on the type and size of data sets utilized by the selected primary studies suggests that not only lower strength of t-wise is being handled by majority of current works, as what being highlighted by Henard *et al*. (2014), but it is generally limited to small and medium sized SPL systems.

Reduction of Problem Space

Techniques reported in S7 and S38 suggested that the problem space is divisible into smaller problem by using divide and compose strategy. Solving the problem in a number of small scale problems could hinder the difficulties of testing in large scale SPL. In S7, they proposed a systematic way to sample a small set of test cases. Their strategy is to split the t-wise combinations into solvable subsets. The idea is to model the problem as a set of constraints and employs a constraint solver to find for solutions for subsets of identified constraints. On the other hand, S38 proposed a divide and combine strategy to tackle feature interaction problem using Generated Data Interactions. This idea was inspired by the intuition that faults may occurred from interactions of database features (e.g., Field values). As compared to others, this primary study is the only work attending the less studied area of SPL testing which is testing the data-intensive SPL system.

Evaluation Metrics

Regarding to the evaluation metrics, in general, relatively small number of primary studies are focusing on faults related metrics, which is only four percent reported for Average Percentage of Faults Detected (APFD) and seven percent for Faults Detection Capability (FDC). Almost one third of the studies were measuring based on Covering Array Size (CAS) and another one third on Overall Execution Time (OET). Although, both (CAS and OET) are beneficial in measuring the efficiency of the proposed technique, it is of equal important to measure the effectiveness, which could be evaluated using faults related metrics. Thus, we perceived more works should be conducted to further evaluate the effectiveness of CIT in testing SPL-based systems.

Test Selection and Prioritization

There have been quite a number of established works in non-SPL testing especially in the area of test case selection and test case prioritization. This has been motivated by the needs to have a more practical and economical testing process. It could be achieved by systematically selecting subset of test cases or running prioritized test cases based on particular testing objectives, which normally is to find faults as early as possible. Similarly, in SPL, the needs to have an efficient and effective testing process have been identified during the early years of SPL software paradigm adoption. Running all the test cases in an existing test suite can results in a large amount of effort or even become infeasible due to deadlines and cost constraints. It may take few days to complete the test configuration generation on an SPL using CIT even with low (2 or 3) t-wise covering array (Johansen *et al*., 2012a). To tackle this, Combinatorial Interaction Testing techniques that employ systematic selection, objective prioritization, or combination of both have been proposed by many and are still an active research area. Weight-based prioritization and product dissimilarity are suggested as most popular prioritization approaches. It is to our surprise that prioritization based on the number of reused features is less considered (one out of eleven) by researchers that adopted CIT in SPL. Feature is one of the most substantial elements in variability modeling of SPL and reusability is one of the must-have ingredients in SPL. Thus, we expect to see more works in prioritizing test configuration using number of reused features.

Parallelization

A couple of attempts have been seen to incorporate parallel processing to speed up the selection and prioritization process. However, we suggest this area demands further research, because of the lack of work especially on handling higher strength of *t*-wise and evaluation against large-scale data sets.

Threats to Validity

Several limitations have been identified during the review works. Proper treatments have been put in place to ensure the review to be as complete and comprehensive as possible. The suitability of the search terms could be questioned, however to the best of our knowledge, all the search terms especially "software product lines" and "combinatorial testing" are well established and universally accepted in its respective context. On the other hand, the accuracy of the search strings is also of our concern. Therefore, search strings were carefully constructed and suited to each search engine. Apart from Boolean operators, we also employed proximity operators (i.e., ONEAR, PRE) to get the best and precise search results. Regarding to the quality of source of primary studies, all primary studies were obtained from reliable and reputable sources, coming from various academic publication fora.

Related Works

This mapping study is focusing on finding evidences in the form of primary studies that are related to the application of Combinatorial Interaction Testing for SPL. An earlier mapping study (Engstrom and Runeson, 2011) highlighted that one of the main challenge in SPL testing is the large number of tests. Reuse of test assets by considering commonalities perceived as one way to enable test effort reduction, but da Mota Silveira Neto *et al*. (2011) highlighted that there was no general solution that deals with systematic reuse in SPL testing. Generally, based on current evidences, this review suggests that CIT is a plausible approach to minimize the redundancy of test assets.

A systematic review by Lamancha *et al*. (2013) reported that works have been done on SPL testing on different testing phases which includes unit testing, integration testing and functional testing. Multitudes of works have been reported on functional testing and variability testing by exploiting UML models and use cases. Other related systematic review performed by Machado *et al*. (2014) only focusing on general testing strategies for SPL. The author reported that testing strategies can be classified as either selecting products prior to testing, or conducting test on individual product. Despite of different scope of study, it shares a common aim with this mapping study, which is to collect and review all relevant primary studies towards test effort reduction.

An orchestrated survey by Khalsa and Labiche (2014) provides a comprehensive and lengthy discussion on available algorithms and tools based on Combinatorial Testing on non-SPL domains.

Recently, a mapping study published by Lopez-Herrejon *et al*. (2015) reported that numerous works have been done on Combinatorial Interaction Testing in SPL, by looking from two perspectives which is *what* phase and *how* phase of CIT. Similar to our work, they also presented state-of-the-art works based on the techniques employed in CIT. While their works focused on finding evidences based on different phases of SPL testing, our work investigate the level of *t*-wise strength in covering array generation towards test configuration selection and prioritization. Our work also differs from theirs such that we reports on the current trend in evaluation metrics employed in CIT testing of SPL. Very small number of works evaluates the effectiveness (such as APFD) of their approach against efficiency measures (such as CAS).

Conclusion

SPL testing demands new mechanism due to its nature of feature commonality and variability. Feature interaction problem of SPL received numerous attentions from testing community, hence showing that it is a significant issue to be addressed. Combinatorial Interaction Testing that employs greedy-based algorithms and meta-heuristics algorithms are identified as two prominent techniques to anticipate the feature interaction problem, thus it is interesting to investigate how well the feature interaction problem has been tackled so far.

Based on this mapping study, it is learnt that most studies are focusing on lower strength of *t*-wise combinatorial testing. Over ninety percent of the reviewed primary studies were engaging *t*-wise of strength two or three, further substantiating claims made by Henard *et al*. (2014). The main limitation that is causing this phenomenon is due to the expensive computation time required to deploy higher strength *t*-wise testing. The exploitation of Combinatorial Interaction Testing technique has long been leveraged on single software product development. Some empirical results for single software development problem suggested that higher strengths are important in detecting more faults. Therefore, based on the small quantity of studies on high strength *t*-wise, it is deemed necessary to further explore the possible correlation between higher strength of *t*-wise with higher faults detection. Apart from that, thus far, we have not discovered any works that exploit *variable strength* covering array in Combinatorial Interaction Testing of SPL. It is therefore our plan to devise a work towards that direction.

With respect to meta-heuristics techniques, so far, only classical evolutionary algorithms are appearing on the primary studies that were engaged in order to improve the covering array generation process. However, to our best knowledge, there are still no work that employs other evolutionary algorithms such as Fruit Flies Algorithm, Artificial Fish Swarm Algorithm, Firefly Algorithm, Cultural Algorithm and Estimation of Distribution Algorithms. While these algorithms demonstrate convincing achievements in other area of software testing, it might produce a plausible contribution towards a better Combinatorial Testing of SPL system.

Acknowledgement

The authors wish to thank anonymous reviewers for their valuable, insightful comments that improve the content of this review paper. We also express our great appreciation and thanks to MOHE and UTHM for sponsoring one of the authors in PhD research. The authors also wish to thank Universiti Putra Malaysia (UPM) which provided the facilities and appropriate environments for carrying out this study and to the Software Engineering research group of FSKTM/UPM for their invaluable comments and suggestions.

Funding Information

This research is funded by the Malaysia's Ministry of Higher Education (MOHE), Universiti Putra Malaysia (UPM) and Universiti Tun Hussein Onn Malaysia (UTHM).

Author's Contributions

Mohd Zanes Sahid: Designed the literature review plan and organized the study and contributed to the writing of the manuscript.

Abu Bakar Md Sultan: The main research supervisor, advised and supervised in the review process of this systematic mapping study.

Abdul Azim Abdul Ghani: Advised and supervised in the software engineering review process of this study.

Salmi Baharom: Advised and supervised in theoretical and technical aspect of software testing issues of the research.

Ethics

The corresponding author confirms that the other authors have read and approved the manuscript and there is no ethical issue involved. This paper is original and contains unpublished material.

References

Al-Hajjaji, M., T. Thüm, J. Meinicke, M. Lochau and G. Saake, 2014. Similarity-based prioritization in software product-line testing. Proceedings of the 18th International Software Product Line Conference, ACM, pp: 197-206. DOI: 10.1145/2648511.2648532

Apel, S., A.V. Rhein, P. Wendler, A. Größlinger and D. Beyer, 2013. Strategies for product-line verification: Case studies and experiments. Proceedings of the 35th International Conference on Software Engineering, May 18-26, IEEE Xplore Press, San Francisco, CA, USA, pp: 482-491. DOI: 10.1109/ICSE.2013.6606594

Atkinson, C. and D. Muthig, 2002. Component-Based Product-Line Engineering with the UML. Software Reuse: Methods, Techniques and Tools, Gacek, C. (Ed.), Springer, Berlin, ISBN-10: 3540460209, pp: 343-344.

Calder, M., M. Kolberg, E.H. Magill and S. Reiff-Marganiec, 2003. Feature interaction: A critical review and considered forecast. Comput. Networks: Int. J. Comput. Telecommun. Network., 41: 115-141. DOI: 10.1016/s1389-1286(02)00352-3

Calvagna, A., A. Gargantini and P. Vavassori 2013. Combinatorial testing for feature models using citlab. Proceedings of the 6th International Conference on Software Testing, Verification and Validation Workshops, Mar. 18-22, IEEE Xplore Press, pp: 338-347. DOI: 10.1109/ICSTW.2013.45

Clements, P. and L. Northrop, 2002. Software Product Lines: Practices and Patterns. 3rd Edn., Addison-Wesley, Boston, ISBN-10: 0201703327, pp: 563.

Cohen, M.B., M.B. Dwyer and J. Shi, 2007. Interaction testing of highly-configurable systems in the presence of constraints. Proceedings of the International Symposium on Software Testing and Analysis, Jul. 09-12, ACM, pp: 129-139. DOI: 10.1145/1273463.1273482

Cohen, M.B., M.B. Dwyer and J. Shi, 2008. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. IEEE Trans. Software Eng., 34: 633-650. DOI: 10.1109/TSE.2008.50

Czarnecki, K. and U.W. Eisenecker, 2000. Generative Programming: Methods, Tools and Applications. 1st Edn., Addison Wesley, Boston ISBN-10: 0201309777, pp: 832.

Czarnecki, K., P. Grünbacher, R. Rabiser, K. Schmid and A. Wąsowski, 2012. Cool features and tough decisions: A comparison of variability modeling approaches. Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems, Jan. 25-27, Leipzig, Germany, pp: 173-182. DOI: 10.1145/2110147.2110167

da Mota Silveira Neto, P.A., I.D. Carmo Machado, J.D. McGregor, E.S. de Almeida and S.R. de Lemos Meira, 2011. A systematic mapping study of software product lines testing. Inform. Software Technol., 53: 407-423. DOI: 10.1016/j.infsof.2010.12.003

Devroey, X., G. Perrouin, A. Legay, P.Y. Schobbens and P. Heymans, 2015. Covering SPL behaviour with sampled configurations: An initial assessment. Proceedings of the 9th International Workshop on Variability Modelling of Software-Intensive Systems, Jan. 21-23, Hildesheim, Germany, pp: 59-59. DOI: 10.1145/2701319.2701325

Do, T.B.N., T. Kitamura, V.T. Nguyen, G. Hatayama and S. Sakuragi *et al*., 2013. Constructing test cases for N-wise testing from tree-based test models. Proceedings of the 4th Symposium on Information and Communication Technology, Dec. 05-06, Danang, Viet Nam, pp: 275-284. DOI: 10.1145/2542050.2542074

Engstrom, E. and P. Runeson, 2011. Software product line testing-a systematic mapping study. Inform. Software Technol., 53: 2-13. DOI: 10.1016/j.infsof.2010.05.011

Galindo, J.A., H. Turner, D. Benavides and J. White, 2014. Testing variability-intensive systems using automated analysis: An application to Android. Software Quality J., 24: 1-41. DOI: 10.1007/s11219-014-9258-y

Garvin, B.J., M.B. Cohen and M.B. Dwyer, 2009. An improved meta-heuristic search for constrained interaction testing. Proceedings of the 1st International Symposium on Search Based Software Engineering, May 13-15, IEEE Xplore Press, pp: 13-22. DOI: 10.1109/SSBSE.2009.25

Garvin, B.J., M.B. Cohen and M.B. Dwyer, 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Eng., 16: 61-102. DOI: 10.1007/s10664-010-9135-7

Harman, M., Y. Jia, J. Krinke, W. B. Langdon, J. Petke and Y. Zhang 2014. Search based software engineering for software product line engineering: a survey and directions for future work. Proceedings of the 18th International Software Product Line Conference-Volume 1, Sept. 15-19, Florence, Italy, pp: 5-18. DOI: 10.1145/2648511.2648513

Haslinger, E.N., R.E. Lopez-Herrejon and A. Egyed, 2013. Using feature model knowledge to speed up the generation of covering arrays. Proceedings of the 7th International Workshop on Variability Modelling of Software-Intensive Systems, Jan. 23-25, Pisa, Italy, pp: 16-16. DOI: 10.1145/2430502.2430524

Henard, C., M. Papadakis, G. Perrouin, J. Klein and Y.L. Traon, 2013. Multi-objective test generation for software product lines. Proceedings of the 17th International Software Product Line Conference, Aug. 26-30, Tokyo, Japan, pp: 62-71. DOI: 10.1145/2491627.2491635

Henard, C., M. Papadakis, G. Perrouin, J. Klein and P. Heymans *et al*., 2014. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans. Software Eng., 40: 650-670. DOI: 10.1109/tse.2014.2327020

Hervieu, A., B. Baudry and A. Gotlieb, 2011a. PACOGEN: Automatic generation of pairwise test configurations from feature models. Proceedings of the IEEE 22nd International Symposium on Software Reliability Engineering, Nov. 29-Dec. 2, IEEE Xplore Press, pp: 120-129. DOI: 10.1109/issre.2011.31

Hervieu, A., B. Baudry and A. Gotlieb, 2011b. Pacogen: Automatic generation of pairwise test configurations from feature models. Proceedings of the 22nd International Symposium on Software Reliability Engineering, 29 Nov.-2 Dec., IEEE Xplore Press, pp: 120-129. DOI: 10.1109/ISSRE.2011.31

Johansen, M.F., Ø. Haugen and F. Fleurey, 2011b. Properties of realistic feature models make combinatorial testing of product lines feasible. Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems, Oct. 16-21, Wellington, New Zealand, pp: 638-652.

Johansen, M.F., Ø. Haugen and F. Fleurey, 2012a. An algorithm for generating t-wise covering arrays from large feature models. Proceedings of the 16th International Software Product Line Conference, Sept. 02-07, Salvador, Brazil, pp: 46-55. DOI: 10.1145/2362536.2362547

Johansen, M.F., Ø. Haugen and F. Fleurey, 2012b. Bow tie testing: A testing pattern for product lines. Proceedings of the 16th European Conference on Pattern Languages of Programs, Jul. 13-17, Irsee, Germany. DOI: 10.1145/2396716.2396725

Johansen, M.F., Ø. Haugen, F. Fleurey, A.G. Eldegard and T. Syversen, 2012d. Generating better partial covering arrays by modeling weights on subproduct lines, Springer.

Johansen, M.F., Ø. Haugen, F. Fleurey, E. Carlson and J. Endresen *et al*., 2012c. A Technique for Agile and Automatic Interaction Testing for Product Lines. Testing Software and Systems, Nielsen, B. and C. Weise, (Eds.), Springer, ISBN-10: 3642346901, pp: 39-54.

Johansen, M., Ø. Haugen and F. Fleurey, 2011a. Properties of realistic feature models make combinatorial testing of product lines feasible. Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems, Oct. 16-21, Wellington, New Zealand, pp: 638-652.

Khalsa, S.K. and Y. Labiche, 2014. An orchestrated survey of available algorithms and tools for combinatorial testing. Proceedings of the 25th International Symposium on Software Reliability Engineering, Nov. 3-6, IEEE Xplore Press, pp: 323-334. DOI: 10.1109/ISSRE.2014.15

Kim, C.H.P., D.S. Batory and S. Khurshid, 2011. Reducing combinatorics in testing product lines. Proceedings of the 10th International Conference on Aspect-Oriented Software Development, Mar. 21-25, Porto de Galinhas, Brazil, pp: 57-68. DOI: 10.1145/1960275.1960284

Kitchenham, B.A. and S. Charters, 2007. Guidelines for performing systematic literature reviews in software engineering. Software Engineering Group, Keele University.

Kowal, M., S. Schulze and I. Schaefer, 2013. Towards efficient SPL testing by variant reduction. Proceedings of the 4th International Workshop on Variability and Composition, Mar. 24-29, Fukuoka, Japan, pp: 1-6. DOI: 10.1145/2451617.2451619

Kuhn, D.R., R.N. Kacker and Y. Lei, 2010. SP 800-142. Practical Combinatorial Testing, National Institute of Standards and Technology.

Kuhn, R., R. Kacker, Y. Lei and J. Hunter, 2009. Combinatorial software testing. Computer, 42: 94-96. DOI: 10.1109/MC.2009.253

Lamancha, B.P., M. Polo and M. Piattini, 2013. Systematic Review on Software Product Line Testing. Software and Data Technologies, Springer, Heidelberg, ISBN-10: 3642361773, pp: 58-71.

Lamancha, B.P., M. Polo and M. Piattini, 2015. PROW: A pairwise algorithm with constraints, order and weight. J. Syst. Software, 99: 1-19. DOI: 10.1016/j.jss.2014.08.005

Lauenroth, K. and K. Pohl, 2005. Principles of Variability. Software Product Line Engineering: Foundations, Principles and Techniques, Pohl, K., G. Böckle, F.J. van der Linden, (Eds.), Springer Science and Business Media, Berlin, ISBN-10: 3540289011, pp: 57-88.

Lee, K., K. Kang and J. Lee, 2002. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. Software Reuse: Methods, Techniques and Tools, Gacek, C. (Ed.), Springer, pp: 62-77.

Lochau, M., S. Oster, U. Goltz and A. Schürr, 2012. Model-based pairwise testing for feature interaction coverage in software product line engineering. Software Quality J., 20: 567-604. DOI: 10.1007/s11219-011-9165-4

Lopez-Herrejon, R.E., F. Chicano, J. Ferrer, A. Egyed and E. Alba, 2013. Multi-objective optimal test suite computation for software product line pairwise testing. Proceedings of the 29th IEEE International Conference on Software Maintenance, Sept. 22-28, IEEE Xplore Press, pp: 404-407. DOI: 10.1109/ICSM.2013.105

Lopez-Herrejon, R.E., J. Ferrer, F. Chicano, A. Egyed and E. Alba, 2014a. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. Proceedings of the IEEE Congress on Evolutionary Computation, Jul. 6-11, IEEE Xplore Press, pp: 387-396. DOI: 10.1109/CEC.2014.6900473

Lopez-Herrejon, R.E., J. Javier Ferrer, F. Chicano, E.N. Haslinger and A. Egyed *et al*., 2014b. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. Proceedings of the Annual Conference on Genetic and Evolutionary Computation, Jul. 12-16, Vancouver, BC, Canada, pp: 1255-1262. DOI: 10.1145/2576768.2598305

Lopez-Herrejon, R.E., S. Fischer, R. Ramler and A. Egyed, 2015. A first systematic mapping study on combinatorial interaction testing for software product lines. Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops, Apr. 13-17, IEEE Xplore Press, pp: 1-10. DOI: 10.1109/ICSTW.2015.7107435

Machado, I.D.C., J.D. McGregor, Y.C. Cavalcanti and E.S. de Almeida, 2014. On strategies for testing software product lines: A systematic literature review. Inform. Software Technol., 56: 1183-1199. DOI: 10.1016/j.infsof.2014.04.002

Marijan, D., A. Gotlieb, S. Sen and A. Hervieu, 2013. Practical pairwise testing for software product lines. Proceedings of the 17th International Software Product Line Conference, Aug. 26-30, Tokyo, Japan, pp: 227-235. DOI: 10.1145/2491627.2491646

Nie, C. and H. Leung, 2011. A survey of combinatorial testing. ACM Comput. Surveys, 43: 1-29. DOI: 10.1145/1883612.1883618

Olimpiew, E.M. and H. Gomaa, 2009. Reusable Model-Based Testing. Formal Foundations of Reuse and Domain Engineering, Edwards, S.H. and G. Kulczycki (Eds.), Springer Science and Business Media, Berlin, ISBN-10: 3642042104, pp: 76-85.

Oster, S., F. Markert and P. Ritter, 2010. Automated Incremental Pairwise Testing of Software Product Lines. Software Product Lines: Going Beyond, Bosch, J. and J. Lee (Eds.), Springer, Berlin, ISBN-10: 3642155790, pp: 196-210.

Oster, S., I. Zorcic, F. Markert and M. Lochau, 2011b. MoSo-PoLiTe: Tool support for pairwise and model-based software product line testing. Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, Jan. 27-29, Namur, Belgium, pp: 79-82. DOI: 10.1145/1944892.1944901

Oster, S., M. Zink, M. Lochau and M. Grechanik, 2011a. Pairwise feature-interaction testing for SPLs: Potentials and limitations. Proceedings of the 15th International Software Product Line Conference, Aug. 22-26, Munich, Germany. DOI: 10.1145/2019136.2019143

Patel, S., P. Gupta and V. Shah, 2013. Feature interaction testing of variability intensive systems. Proceedings of the 4th International Workshop on Product Line Approaches in Software Engineering, May 20-20, IEEE Xplore Press, pp: 53-56. DOI: 10.1109/PLEASE.2013.6608666

Lamancha, B.P. and M.P. Usaola, 2010. Testing Product Generation in Software Product Lines Using Pairwise for Features Coverage. Testing Software and Systems, Petrenko, A., A. Simão and J. Maldonado, Springer, Berlin, ISBN-10: 3642165737, pp: 111-125.

Perrouin, G., S. Oster, S. Sen, J. Klein and B. Baudry et al., 2012. Pairwise testing for software product lines: Comparison of two approaches. Software Quality J., 20: 605-643. DOI: 10.1007/s11219-011-9160-9

Perrouin, G., S. Sen, J. Klein, B. Baudry and Y. le Traon, 2010. Automated and scalable t-wise test case generation strategies for software product lines. Proceedings of the 3rd International Conference on Software Testing, Verification and Validation, Apr. 6-10, IEEE Xplore Press, pp: 459-468. DOI: 10.1109/icst.2010.43

Petersen, K., R. Feldt, S. Mujtaba and M. Mattsson, 2008. Systematic mapping studies in software engineering. Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, Jun. 26-27, ACM, pp: 68-77. dl.acm.org/citation.cfm?id=2227123

Reis, S., A. Metzger and K. Pohl, 2007. Integration testing in software product line engineering: A model-based technique. Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering, Mar. 24-Apr. 1, Braga, Portugal, pp: 321-335. DOI: 10.1007/978-3-540-71289-3_25

Sánchez, A.B., S. Segura and A. Ruiz-Cortés, 2014a. A comparison of test case prioritization criteria for software product lines. Proceedings of the 7th International Conference on Software Testing, Verification and Validation, 31 Mar.-4 Apr., IEEE Xplore Press, pp: 41-50. DOI: 10.1109/ICST.2014.15

Sánchez, A.B., S. Segura and A. Ruiz-Cortés, 2014b. The Drupal framework: A case study to evaluate variability testing techniques. Proceedings of the 8th International Workshop on Variability Modelling of Software-Intensive Systems, Jan. 22-24, Sophia Antipolis, France. DOI: 10.1145/2556624.2556638

Sánchez, A.B., S. Segura, J.A. Parejo and A. Ruiz-Cortés, 2015. Variability testing in the wild: The Drupal case study. Software Systems Model., 1: 1-22. DOI: 10.1007/s10270-015-0459-z

Scheidemann, K.D., 2006. Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems. Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society, Aug. 21-24, IEEE Xplore Press, pp: 75-84. DOI: 10.1109/SPLINE.2006.1691579

Sen, S. and A. Gotlieb, 2013. Testing a Data-Intensive System with Generated Data Interactions. Advanced Information Systems Engineering, Salinesi, C., M.C. Norrie and O. Pastor, Springer Berlin Heidelberg, Berlin, ISBN-10: 364238708X, pp: 657-671.

Sisodia, R.S. and V. Channakeshava, 2009. Combinatorial approach for automated platform diversity testing. Proceedings of the 4th International Conference on Software Engineering Advances, Sept. 20-25, IEEE Xplore Press, pp: 134-139. DOI: 10.1109/ICSEA.2009.28

Sloane, N.J., 1993. Covering arrays and intersecting codes. J. Combinat. Designs, 1: 51-63. DOI: 10.1002/jcd.3180010106

Steffens, M., S. Oster, M. Lochau and T. Fogdal, 2012. Industrial evaluation of pairwise spl testing with moso-polite. Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems, Jan. 25-27, Leipzig, Germany, pp: 55-62. DOI: 10.1145/2110147.2110154

Thiel, S. and F. Peruzzi, 2000. Starting a product line approach for an envisioned market: Research and experience in an industrial environment. Proceedings of the 1st Conference on Software Product Lines: Experience and Research Directions, (ERD' 00), Kluwer Academic Publishers, Denver, Colorado, USA, pp: 495-512. DOI: 10.1007/978-1-4615-4339-8_26

Wang, S., S. Ali and A. Gotlieb, 2013. Minimizing test suites in software product lines using weight-based genetic algorithms. Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, Jul. 06-10, AMSTERDAM, Netherlands, pp: 1493-1500. DOI: 10.1145/2463372.2463545

Wang, S., S. Ali and A. Gotlieb, 2015. Cost-effective test suite minimization in product lines using search techniques. J. Syst. Software, 103: 370-391. DOI: 10.1016/j.jss.2014.08.024

Wang, S., S. Ali, A. Gotlieb and M. Liaaen, 2014. A systematic test case selection methodology for product lines: Results and insights from an industrial case study. Empirical Software Eng., 21: 1-37. DOI: 10.1007/s10664-014-9345-5

Weiss, D.M. and C.T.R. Lai, 1999. Software Product-Line Engineering: A Family-based Software Development Process. 1st Edn., Addison-Wesley, Reading, ISBN-10: 0201694387, pp: 426.

Yu, L., F. Duan, Y. Lei, R.N. Kacker and D.R. Kuhn, 2014. Combinatorial test generation for software product lines using minimum invalid tuples. Proceedings of the 15th International Symposium on High-Assurance Systems Engineering, Jan. 9-11, IEEE Xplore Press, pp: 65-72. DOI: 10.1109/HASE.2014.18