Empirical Software Engineering (2024) 29:146
https://doi.org/10.1007/s10664-024-10516-x

Software product line testing: a systematic literature review

Halimeh Agh[1] · Aidin Azamnouri[1] · Stefan Wagner[1,2]

Accepted: 19 June 2024

Abstract
A Software Product Line (SPL) is a software development paradigm in which a family of software products shares a set of core assets. Testing has a vital role in both single-system development and SPL development in identifying potential faults by examining the behavior of a product or products, but it is especially challenging in SPL. There have been many research contributions in the SPL testing field; therefore, assessing the current state of research and practice is necessary to understand the progress in testing practices and to identify the gap between required techniques and existing approaches. This paper aims to survey existing research on SPL testing to provide researchers and practitioners with up-to-date evidence and issues that enable further development of the field. To this end, we conducted a Systematic Literature Review (SLR) with seven research questions in which we identified and analyzed 118 studies dating from 2003 to 2022. The results indicate that the literature proposes many techniques for specific aspects (e.g., controlling cost/effort in SPL testing); however, other elements (e.g., regression testing and non-functional testing) still need to be covered by existing research. Furthermore, most approaches are evaluated by only one empirical method, most of which are academic evaluations. This may jeopardize the adoption of approaches in industry. The results of this study can help identify gaps in SPL testing since specific points of SPL Engineering still need to be addressed entirely.

Keywords Software product lines · Software testing · Software quality · Systematic literature review

Communicated by Sven Apel.

Halimeh Agh
halimeh.agh@iste.uni-stuttgart.de

Aidin Azamnouri
aidin.azamnouri@iste.uni-stuttgart.de

Stefan Wagner
stefan.wagner@tum.de

1 Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany

2 TUM School of Communication, Information and Technology, Technical University of Munich, Heilbronn, Germany

1 Introduction

Software Product Line (SPL) engineering has proven to be an efficient and effective strategy to decrease implementation costs, reduce time to market, and improve the quality of derived products (Denger and Kolb 2006; Northrop et al. 2007). SPLs and Configurable Systems (Alves Pereira et al. 2020) are two approaches used in software engineering to manage and create software with varying levels of customization and flexibility. While both SPLs and configurable systems share the goal of offering flexibility and customization, they differ in their core approach. SPLs primarily emphasize the systematic reuse of components, architectures, and design patterns across a range of related software products. In contrast, configurable systems are single software products designed to be adaptable, enabling users to configure them to meet their unique requirements. We decided to limit the scope on SPL to keep the review focused.

Testing is an essential part of SPL Engineering (SPLE) to identify potential faults (Pohl and Metzger 2006). This activity examines core assets shared among many products, product-specific parts, and the interaction among them (McGregor 2001). Therefore, SPL testing includes activities from the validation of initial requirements to the acceptance testing of a specific product by customers (Da Mota Silveira Neto et al. 2011).

As the adoption of the SPL approach by companies has grown (Weiss 2008), many researchers have made contributions in the SPL testing field to provide efficient and effective approaches that can satisfy specific needs of the industry (e.g., controlling the cost/effort of SPL testing). This resulted in many publications on different aspects of SPL testing. Therefore, analyzing research conducted in this field using well-known empirical methods is required to provide an overview of state-of-the-art testing practices and assess the effectiveness of the proposed approaches. To this end, Systematic Literature Reviews (SLR) and Systematic Mapping Studies (SMS) were conducted on SPL testing, but the most recent one dates back to 2014 (do Carmo Machado et al. 2014). While some recent research has focused on reviewing specific aspects of SPL testing, such as model-based testing of SPLs (Petry et al. 2020), test case prioritization for SPL (Kumar 2016), and combinatorial interaction testing for software product lines (Lopez-Herrejon et al. 2015), there has not been an SLR or SMS since 2014 that provides a comprehensive overview of the current state of SPL testing in a general context. Therefore, there is a need to update existing literature reviews (Mendes et al. 2020) to identify up-to-date evidence and issues that enable further development of the SPL testing field.

This paper presents an SLR to analyze interesting aspects of SPL testing that are formalized as research questions. An SLR is a rigorous and systematic method to identify, evaluate, and interpret all available research relevant to a particular research question, topic area, or phenomenon of interest (Cruzes and Dybä 2011). The specific aspects based on which we analyzed relevant studies are:

- Characteristics of the studies focused on SPL testing.
- Test levels executed throughout the SPL lifecycle.
- Creating test assets by considering commonalities and variabilities.
- Dealing with configuration-aware software testing.
- Preserving traceability between test assets and other artifacts.
- Testing non-functional requirements in an SPL.

- Controlling cost/effort of SPL testing.

The SLR process was conducted from June 2022 to the end of 2022. While some of the findings derived from this SLR align with the conclusions of previous SLRs, such as the identification of existing gaps in non-functional testing for SPLs and the necessity for more robust and user-friendly testing tools, our review uncovered specific insights and unaddressed gaps in this domain that were not fully explored in prior SLRs. These include:

1. Variability control, referring to the disciplined management and regulation of feature variations within SPLs, alongside modeling and tracing, presents persistent challenges that require attention throughout the testing process. Variability control involves implementing strategies, such as configuration and change management, to ensure consistency and predictability in the diverse configurations of products derived from the SPL.
2. Novel approaches are needed for regression test selection, prioritization, and minimization, along with architecture-based regression testing, to effectively manage regression testing in SPLs.
3. Promoting the adoption of SPL testing practices in industrial settings necessitates addressing practical challenges, such as offering guidance for industry-specific SPL testing, and conducting industrial evaluations.
4. Exploring the details of test levels across the SPL lifecycle and highlighting the consequences of neglecting a particular test level can offer valuable insights for practitioners.
5. Studies focusing on testing SPLs rarely address traceability explicitly. Considering feature variability and configuration management, more efficient methods for modeling and representing traceability relationships are required.

The remainder of this paper is organized as follows: Sect. 2 provides background information required to understand SPL and SPL testing concepts; Sect. 3 describes how the SLR methodology has been applied; the results of the SLR are reported in Sect. 4; potential threats to the validity of this study and the strategies employed to mitigate them are discussed in Sect. 5; Sect. 6 presents a summary of the research and examines the main findings; Sect. 7 provides a survey of the related research; Sect. 8 presents concluding remarks and further research.

2 Background

This section provides a concise background on the SPL development process, variability management, and testing approaches and levels as a basis for the remainder of this article.

2.1 SPL development process

SPL is a software development paradigm to achieve economies of scale and scope by analyzing product commonalities and variabilities. As this paradigm has specific benefits such as substantial cost savings, reduction of time to market, and high productivity, many organizations, including Philips, Nokia, Cummins, and Hewlett-Packard, have adopted it (Clements and Northrop 2002). In SPL, a set of core assets (e.g., reference architecture and reusable components) is first developed. Specific products are then built by configuring and composing the core assets in a prescribed way with product-specific features to satisfy particular market segments (Clements and Northrop 2002).

The SPL development process/lifecycle can be divided into two distinct phases: Domain Engineering and Application Engineering. According to Czarnecki and Eisenecker (2000, p. 20), Domain Engineering is "the activity of collecting, organizing, and storing experience in building systems or parts of systems in a particular domain in the form of reusable assets, as well as providing an adequate means for reusing these assets when building new systems." Application Engineering is focused on deriving specific products from the core assets created during Domain Engineering; in this phase, specifics of the products are added to common parts to satisfy the particular needs of a product (Clements and Northrop 2002). Of these two phases, Domain Engineering demands significant resources and time. If not managed effectively, it can lead to the failure of the entire SPL (Pohl et al. 2005, p. 9–10). Three common approaches are employed for constructing an SPL, and each of these approaches directly influences the implementation of Domain Engineering (Apel et al. 2013):

- Proactive approaches start with a comprehensive and thorough scoping of the domain to anticipate all requirements. Subsequently, all these requirements are implemented as assets, and SPL experts typically carry out this task.
- Extractive approaches follow an automated process, utilizing a set of existing product variants as input. The SPL is constructed by extracting features from these variants. Features are identified and retrieved through feature location techniques (AL-Msie'deen et al. 2013; Rubin and Chechik 2013).
- Reactive approaches follow an incremental process. They take as input an existing SPL version (SPL_i) and a set of new requirements about a new product. This process results in the creation of SPL_{i+1}, which can produce the new product.

2.2 Variability Management in SPL

In SPL engineering, variability mechanisms are fundamental for managing diversities across products. These mechanisms, as classified by Apel et al. (2013), include annotative mechanisms, transformative mechanisms (delta-oriented), and feature-oriented mechanisms. Annotative mechanisms involve marking or annotating code to denote variability points, while transformative mechanisms, such as delta-oriented programming, describe changes required to transform one product variant into another. Feature-oriented mechanisms organize variability around features and their interactions. These variability mechanisms can be applied across all stages of the software lifecycle.

A Feature Model is commonly used in Domain Engineering to present different combinations of features. A feature model is a formal representation and graphical notation that describes the variability and relationships among features in an SPL. Feature models typically consist of features (functionalities or characteristics), feature hierarchies (representing parent-child relationships between features), and constraints (rules governing the valid combinations of features) (Pohl et al. 2005). Due to the presence of numerous optional features, the configuration space in feature models may exponentially increase (reaching 2^n possible configurations, where *n* represents the number of optional features without further constraints) (Chen and Babar 2011). A specific product can be derived once a complete feature configuration is established.

Although proactive approaches emphasize systematic upfront planning, modeling variabilities with feature and configuration models, and high asset reusability, reactive methods can also use feature models to represent variabilities introduced by new requirements. Configuration files or mechanisms are often used in reactive approaches to specifying how variabilities are configured in reaction to new requirements (Ghanam et al. 2010). Furthermore, extractive approaches may employ feature models to represent and visualize variabilities discovered in existing products. Configuration scripts or files may be used to document and manage variabilities found in the codebase (Parra et al. 2012).

2.3 Testing approaches and levels

There exist diverse approaches to software testing, including (Luo 2001; Jorgensen 2013):

- **Manual testing**: Testers create and execute test cases manually to evaluate the behavior of a software application or system without using automated testing tools or scripts.
- **Automated Testing**: Specialized testing tools and scripts are used to automate the execution of test cases and the verification of software applications or systems.
- **Functional testing**: Focuses on verifying software functions according to specified requirements. This approach includes different levels of testing, including:

  – **Unit Testing** is conducted at the lowest level, focusing on the fundamental unit of software, referred to interchangeably as "unit," "module," or "component."
  – **Integration Testing** takes place when two or more tested units are integrated into a larger structure. This testing assesses the interactions between components and evaluates the quality of the overall structure when the properties cannot be determined solely from its individual components.
  – **System Testing** aims to validate the comprehensive quality of the entire system, covering end-to-end functionality. This type of testing typically aligns with the functional and requirement specifications of the system. Additionally, it assesses non-functional quality attributes like reliability, security, and maintainability.
  – **Acceptance Testing** occurs when the developers deliver the completed system to the customers or users. The primary goal of acceptance testing is to give confidence that the system functions correctly rather than to uncover errors.

- **Non-functional testing**: Focuses on evaluating the attributes of a software system that are not directly related to its functional behavior. Instead, non-functional testing assesses the system's performance, reliability, scalability, security, usability, and other qualities that impact the overall user experience and the system's ability to meet non-functional requirements.
- **Regression testing**: Focuses on verifying that recent changes or updates to a software application have not introduced new defects or negatively affected existing functionality.
- **Model-based testing**: Test cases are derived from models representing the software's expected behavior. Different models can be used to generate test cases systematically,

including graphical representations, mathematical models, or formal notations.

SPL testing is an essential activity in SPLE to identify potential faults (Pohl and Metzger 2006). Exhaustive testing in SPL is usually infeasible due to a combinatorial explosion in the number of products. Following Tevanlinna et al. (2004), Reuys et al. (2005), Käköla and Dueñas (2006), there are specific differences between single-system testing and SPL testing:

1) Testing is a part of both phases: Domain Engineering and Application Engineering. Domain testing is focused on testing domain artifacts (e.g., requirements, features, and source code); however, as domain artifacts include variability, completely testing the domain artifacts in domain testing is impossible. Application testing aims to detect remaining faults in a derived product mainly caused by unexpected interactions.
2) Test assets created in Domain Engineering (e.g., test cases, test scenarios, test results, and test data) are reused in Application Engineering to test instantiated products. To this end, test assets should be created by considering variability, which we call variant-rich test assets.

3 Systematic literature review methodology

To carry out this SLR, we followed guidelines for performing SLRs in software engineering (Kitchenham and Charters 2007). The steps followed in conducting this SLR are developing a review protocol, conducting the review, analyzing the results, reporting the results, and discussing the findings. The review protocol used in this SLR is explained in the following subsections. The protocol includes the formulation of research questions to achieve the objective (Sect. 3.1), identification of sources to extract the research papers, the search criteria and principles for selecting the relevant studies (Sect. 3.2), specifying a set of criteria to assess the quality of each study remained for data extraction (Sect. 3.3), and developing the template used for extracting data (Sect. 3.4).

3.1 Research questions

As previously stated, this study aims to investigate how the existing approaches deal with testing in SPL. To formulate research questions, we examined topics addressed by previous research on SPL testing (Pérez et al. 2009; Engström and Runeson 2011; Da Mota Silveira Neto et al. 2011; do Carmo Machado et al. 2014). Some of the research questions were completely reused from previous research – i.e., RQ1, RQ2, RQ3, RQ6, and RQ7 – and some of them were formulated by analyzing specific aspects that have not been investigated in detail in previous research – i.e., RQ4 and RQ5.

We reuse RQs to contrast and compare the newer research contributions with the results of previous SLRs. Yet, we identified two unique, interesting aspects: Because testing every potential configuration of an SPL is often impractical, it becomes essential to employ specific approaches for identifying valid and invalid configurations. We have examined the techniques utilized or proposed in RQ4 to address this issue. Maintaining traceability between test assets and other SPL artifacts offers substantial advantages, including enhanced reusability, impact analysis, and change management. Consequently, we designed RQ5 to investigate the techniques employed for preserving traceability. Answering these questions led to a detailed investigation of the identified studies to specify practical and research issues regarding SPL testing; therefore, the results of this study can support both industrial and academic activities. The research questions are as follows:

- **RQ1. How is the research on SPL testing characterized?** This question intends to discuss the bibliometrics of the primary studies and the evidence available to adopt the proposed approaches.
- **RQ2**. **What levels of tests are usually executed throughout the SPL lifecycle (i.e., Domain Engineering and Application Engineering)?** There are different levels of tests, and each level is associated with a specific development phase, including unit, integration, system, and acceptance tests (Ammann and Offutt 2008; Jaring et al. 2008). This question aims to specify different test levels usually executed throughout the SPL lifecycle.
- **RQ3**. **How are test assets created by considering commonalities and variabilities?** The large number of variation points and variants in an SPL increases the number of possible testing combinations. Creating test assets for all combinations of functionality is almost impossible in practice; therefore, test assets must be created by considering commonality and variability so that they can be reused as much as possible. Furthermore, an undetected error in common core assets of an SPL can be spread to all instances depending on those assets (Pohl and Metzger 2006); therefore, creating test assets by considering commonalities and variabilities and testing common aspects as early as possible is essential. Answering this question led to investigating how testing approaches handle commonality and variability throughout creating/executing test assets.
- **RQ4**. **How do SPL approaches deal with configuration-aware software testing?** Testing all functionality combinations in an SPL is impossible and unnecessary since some combinations are invalid based on the constraints defined between configuration parameters. This question is intended to specify ways/techniques to detect valid and invalid combinations of configuration parameters.
- **RQ5**. **How is the traceability between test assets and other artifacts of SPL preserved throughout the SPL lifecycle?** The reusability of test assets is essential to manage the complexity of SPL testing; preserving traceability between test assets and requirements/implementation can enhance the reusability of test assets. In this sense, this question is intended to identify specific ways/techniques to achieve traceability between test assets and other artifacts throughout the SPL lifecycle.
- **RQ6**. **How are Non-Functional Requirements (NFRs) tested in SPL?** NFRs such as security, reliability, and performance are very important for SPLs, and ignoring these requirements can lead to negative results (e.g., economic loss) (Nguyen 2009). Therefore, systematically testing NFRs by considering commonalities and variabilities is an important aspect of SPLE. This question is intended to investigate how tests of NFRs are performed in an SPL.
- **RQ7**. **What mechanisms have been used for controlling cost/effort of SPL testing?** As SPL testing is more expensive than single-system testing, identifying specific techniques to reduce effort can provide the reader with an initial list of techniques identified by analyzing the selected studies. The specified list can be enriched regarding new

publications about SPL testing.

3.2 Identification of relevant literature

The process of gathering and selecting primary studies has been performed in three stages: in the first stage, we investigated previously published literature reviews on SPL testing (Pérez et al. 2009; Engström and Runeson 2011; Da Mota Silveira Neto et al. 2011; do Carmo Machado et al. 2014) to identify the initial set of papers that have been published up to 2013. In the second stage, we updated the list of papers by searching for new papers published between 2013 and 2022; in this stage, we performed forward and backward snowballing (Webster and Watson 2002) to identify missing relevant papers. In the third stage, we applied inclusion and exclusion criteria to each potential primary study identified through stages one and two. Each of the three stages is explained in detail in the following subsections. We must note that we chose studies that could address at least one of the RQs while selecting primary studies. For instance, certain studies focusing on SPL verification were included because they could provide insights relevant to questions such as RQ4. An Excel file was created to be shared among the authors to document the various steps of the SLR process. This file[1] contains all the details about how we gathered and selected primary studies and how we extracted data from the chosen studies.

3.2.1 Analysis of existing reviews

By searching for existing SLRs or Systematic Mapping Studies (SMSs) on SPL testing, we found four SLRs (Engström and Runeson 2011; Da Mota Silveira Neto et al. 2011, Pérez et al. 2009; do Carmo Machado et al. 2014). Engström and Runeson (2011) conducted an SMS to identify useful approaches and needs for future research; in this study, 64 papers published up to 2008 were surveyed. Da Mota Silveira Neto et al. (2011) performed an SMS to investigate state-of-the-art testing practices in SPL testing; this study analyzed a set of 45 publications from 1993 to 2009. Pérez et al. (2009) conducted an SLR to identify experience reports and initiatives carried out in the SPL testing area; in this study, 23 primary studies published up to 2009 were analyzed. do Carmo Machado et al. (2014) conducted an SLR by analyzing 49 studies published up to 2013. As the four studies followed a systematic process to gather and select the primary studies, we are confident that they covered all the primary studies in the SPL testing field published up to 2013.
Using the list of primary studies in the four SLR/SMS, a set of 181 potentially relevant papers was identified, shown as stage 1.1 in Fig. 1. By reading the titles and abstracts of the publications, papers that addressed none of the research questions were excluded. Furthermore, duplicated papers were removed, i.e., those included

[1] Replication package available on https://zenodo.org/doi/https://doi.org/10.5281/zenodo.10018266.

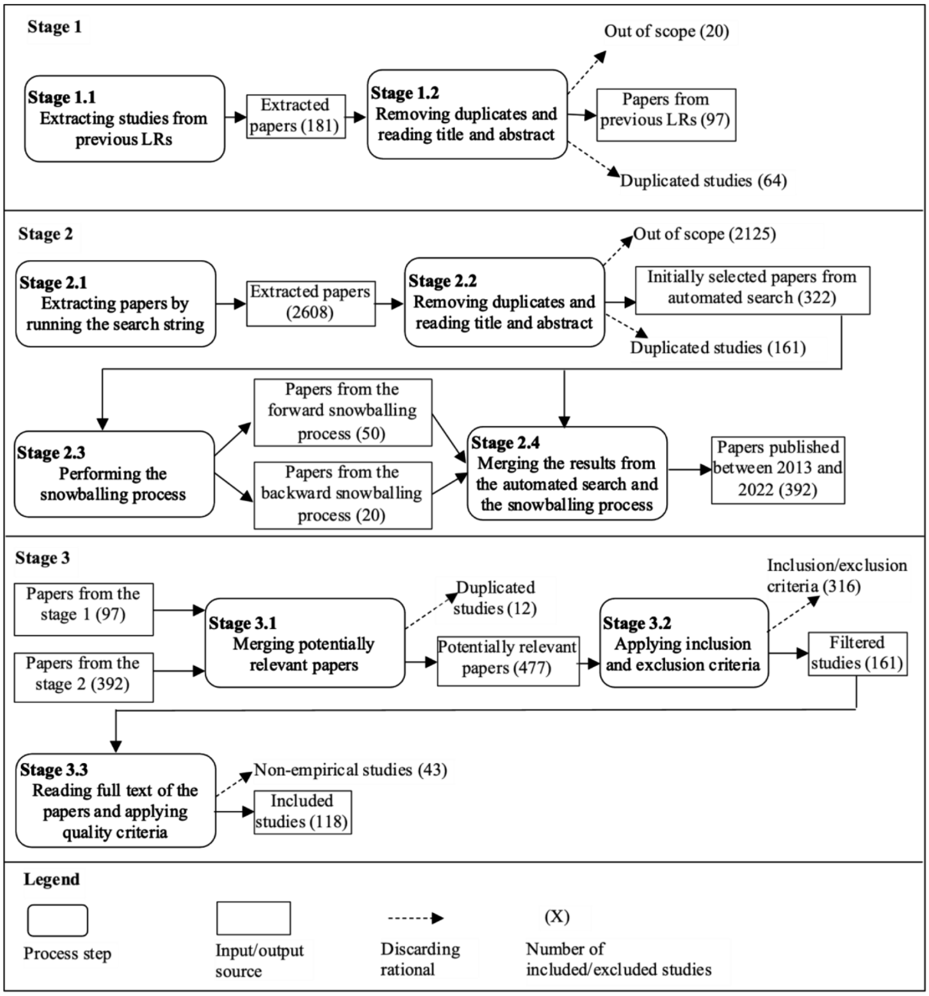
Stage 1
Stage 1.1 Extracting studies from previous LRs
Extracted papers (181)
Stage 1.2 Removing duplicates and reading title and abstract
Out of scope (20)
Papers from previous LRs (97)
Duplicated studies (64)
Stage 2
Stage 2.1 Extracting papers by running the search string
Extracted papers (2608)
Stage 2.2 Removing duplicates and reading title and abstract
Out of scope (2125)
Initially selected papers from automated search (322)
Duplicated studies (161)
Stage 2.3 Performing the snowballing process
Papers from the forward snowballing process (50)
Papers from the backward snowballing process (20)
Stage 2.4 Merging the results from the automated search and the snowballing process
Papers published between 2013 and 2022 (392)
Stage 3
Papers from the stage 1 (97)
Papers from the stage 2 (392)
Stage 3.1 Merging potentially relevant papers
Duplicated studies (12)
Potentially relevant papers (477)
Stage 3.2 Applying inclusion and exclusion criteria
Inclusion/exclusion criteria (316)
Filtered studies (161)
Stage 3.3 Reading full text of the papers and applying quality criteria
Non-empirical studies (43)
Included studies (118)
Legend
Process step
Input/output source
Discarding rational
(X)
Number of included/excluded studies

Fig. 1 The process of gathering and selecting primary studies

in more than one literature review. At the end of this stage, 97 studies were finally selected, shown as stage 1.2 in Fig. 1.

3.2.2 Gathering recent publications

In the second stage of the search process, we updated the list of primary studies by analyzing papers published between 2013 and 2022 using the following databases: IEEE Xplore, Scopus, ACM DL, Springer, and Wiley online library. To answer the stated research questions, we identified the keywords that had to be used in the search process. Variants of the terms "*software product line*", "*software product family*", and "*software testing*" were applied to compose the search query, as follows:

(Software Product Line OR Software Product Lines OR Software Product Family OR Software Product Families) AND (Test OR Testing).

To evaluate the search string, we first performed a limited manual search to see whether the results of that search were among the results obtained by running the search string. The search string was adapted based on the syntax requirements of each data source used. Table 13 in Appendix A shows the forms of search strings applied to different engines and the number of papers extracted from each data source.

We obtained a set of 2,608 papers by running the search string on the search engines, shown as stage 2.1 in Fig. 1. We excluded 161 papers as duplicates since they were retrieved from multiple search engines. Furthermore, by reading the titles and abstracts of the remaining papers, a set of 2,125 papers was identified as irrelevant since they considered testing from a single-system development perspective, not an SPL point of view. At the end of this step, we had 322 papers, shown as stage 2.2 in Fig. 1.

In the next step, we conducted both backward and forward snowballing by examining the reference lists of all the identified papers and exploring the papers that have cited these identified papers, respectively. Following this step, 70 additional papers (20 via backward snowballing and 50 via forward snowballing) were added to the previously identified set of papers, shown as stage 2.3 in Fig. 1. At the end of stage 2, we had a set of 392 new publications, shown in Fig. 1 as stage 2.4.

3.2.3 Primary study selection strategy

By merging the results of the two previous stages, a set of 477 papers was composed, shown as stage 3.1 in Fig. 1. Throughout the merging process, we identified 12 papers as duplicates because the year 2013 was considered in both the SLR conducted by do Carmo Machado et al. (2014) and in the automated search stage. We defined a set of inclusion and exclusion criteria to assess each potential primary study; the criteria are presented in Table 1. These criteria were applied to the titles and abstracts of the identified papers. The first author performed this stage. However, to reduce the researcher bias, the results of this stage were validated by the second and third authors of this paper.

At this stage, we initially applied inclusion criteria to select papers meeting all of the specified criteria for inclusion. Following this, we applied exclusion criteria to exclude papers that met one or more of the specified exclusion criteria. We included only papers evaluated via at least one empirical method, including Case study, Survey, Experiment, and Observational study (Wohlin et al. 2003; Sjoberg et al. 2007; Zhang et al. 2018). At the end of this stage, a set of 161 papers were selected to be subject to full-text reading, depicted in Fig. 1 as stage 3.2. The analysis results of the papers, conducted based on the inclusion and exclusion criteria, are accessible within the replication package.

3.3 Quality assessment

Quality assessment of candidate studies is recommended to be performed to ensure that studies are impartially assessed for quality (Kitchenham et al. 2016). To this end, we used a set of quality criteria to examine the studies, shown in Table 14 in Appendix B. These criteria were reused from the criteria proposed by Dybå and Dingsøyr (2008) and cover four main aspects related to quality, including:

Table 1 Inclusion and exclusion criteria

Type	Criteria
Inclusion criteria	Papers written in English
	Papers that present an approach for SPL testing and/or answer at least one of the RQs
	Papers that have been evaluated by at least one empirical evaluation method
Exclusion criteria	Secondary studies (these papers are presented in the Related Work section)
	Extended studies (only the most complete version of a study is kept and the others is excluded)
	Papers available only in the form of abstracts or PowerPoint presentations
	Technical reports and white papers
	Graduation projects, Master's theses and PhD dissertations
	Textbooks, both print and electronic
	Papers published in doctoral symposia
	Comparative studies, with no additional contribution
	Position or philosophical papers that only present an anecdotal evidence of SPL testing
	Papers available only in the form of abstracts or PowerPoint presentations
	Papers that do not directly address testing, instead they generally consider SPL

- **Reporting**: Reporting of the study's rationale, aims, and context.
- **Rigor**: Has a thorough and appropriate approach been applied to key research methods in the study?
- **Credibility**: Are the findings well-presented and meaningful?
- **Relevance**: How useful are the findings to the software industry and the research community?

We used a weighting approach to examine the candidate studies in which two optional answers with their respective score were given for each question: "Yes" = 1, and "No" = 0. Then, we assigned a quality assessment score to each study by summing up the scores given to all the questions; the total quality score for each study ranged from 0 (very poor) to 11 (very good). The two authors assessed the papers, and any discrepancies were resolved by holding sessions with all the authors.

The first three criteria shown in Table 14 in Appendix B were used as the minimum quality threshold of the review to exclude non-empirical research papers. To this end, if question 1, or both of questions 2 and 3, received a "0" response, we did not continue the quality assessment process, and the paper was excluded. The results of the quality assessment for each paper are available in the replication package. Consequently, 43 papers were excluded, and 118 were selected as primary studies, shown in Fig. 1 as stage 3.3. The list of primary studies is presented in Table 15 in Appendix C.

The analysis of the studies based on quality assessment criteria is explained in more detail in Appendix E. In summary, concerning Reporting, most of the studies performed well. While the context description could be better in some studies, approximately 82% have clear research objectives, and all studies are based on research. On average, the studies performed reasonably well in terms of Rigor. Researchers have justified the research design in almost 62% of studies to accomplish the research's goals. A base approach has been compared with the proposed approach in around 60% of studies, with the researchers attempting to prove that the selected controls reflect a defined population. Despite these promising findings, 32% of the studies fail in rigor. According to the credibility issue, around 95% of the studies discuss the results in relation to the research questions and highlight the study's limitations. Most studies, however, need to establish relationships between the researcher and participants and the data collection that addresses the research problem. Regarding Relevance, about 97% of studies explicitly discuss SPL testing and how it contributes to existing knowledge, identifies new areas for research, and explains how the results can be used. Nevertheless, practitioner-based guidelines are present in about 15% of cases, indicating that more practical guidance is needed to strengthen industry adoption of SPL testing.

3.4 Data extraction and analysis

Data was extracted from each of the 118 primary studies during this stage. To this end, we used a predefined extraction form that enabled us to record the full details of the studies and be specific in answering research questions. The extraction form is shown in Table 2. The first two authors conducted the process of reading and completing the extraction form; the data were extracted and stored in a spreadsheet after reading each paper and shared with all the authors. We followed the content structuring / theme analysis approach of Mayring (2014) to analyze the data. The types of extracted data from the extraction form already provided us with a list of themes and the corresponding extracted data for these themes. This step was deductive. In the next step, we inductively created categories in the themes to summarize them. All the authors held multiple sessions to discuss the intermediate results and resolve any potential discrepancies.

4 Results

In the following sections, the data extracted from the primary studies is used to answer the research questions. An overview of the primary studies is first provided in Sect. 4.1. Then, we answer each RQ via the extracted data.

4.1 Characteristics of the studies (RQ1)

This section discusses the bibliometrics of the primary studies, the evidence available to adopt the proposed approaches, and the results of the evaluations conducted based on the quality assessment criteria.

4.1.1 Bibliometrics

In this section, we analyze annual trends and distribution per venue type of the studies selected.

Table 2 Data extraction template

	Extracted item	Type of data
General information	Author(s)	Free text
	Year of publication	Free text
	Venue	Free text
	Publication type	Predefined list of venues, including Journal article, conference proceeding, symposiums, workshop, and book chapter
	Complete reference	Free text
Empirical evaluation method	The type of empirical evaluation method(s) used to validate the proposed approach	Predefined list of existing methods for empirical evaluation, including Case study, Survey, Experiment, and Observational study (Wohlin et al. 2003; Sjoberg et al. 2007; Zhang et al. 2018)
RQ1	Characteristics of the primary studies	Predefined list of characteristics including the bibliometrics of the studies and the evidence available to adopt the proposed approaches
RQ2	Level of tests usually executed throughout the SPL lifecycle	Predefined list of test levels, including unit, integration, system and acceptance tests (Ammann and Offutt 2008; Jaring et al. 2008)
RQ3	Mechanisms/techniques used to create test assets by considering commonality and variability	List of mechanisms/techniques gradually completed by studying primary studies
RQ4	Mechanisms/techniques used to configuration-aware software testing	List of mechanisms/techniques gradually completed by studying primary studies
RQ5	Mechanisms/techniques used to preserve the traceability between test assets and other artifacts of SPL	List of mechanisms/techniques gradually completed by studying primary studies
RQ6	Mechanisms/techniques used to test non-functional requirements in SPL	List of mechanisms/techniques gradually completed by studying primary studies
RQ7	Mechanisms/techniques used to control the cost/effort of SPL testing	List of mechanisms/techniques gradually completed by studying primary studies

Annual trend: The distribution of the primary studies according to publication year is shown in Fig. 2. No publication prior to 2003 focuses on SPL testing. However, after 2003, there was at least one paper per year, except for 2004. As seen in Fig. 2, the number of published papers in this field has generally increased over time (2003–2019). This indicates that the SPL Testing field has attracted the attention of many researchers in the last few years. Furthermore, it shows increasing attention to the use of empirical methods to assess the value of proposed approaches since we only included empirically evaluated studies in our review. As we excluded some of the papers based on the quality assessment criteria, there is no primary study published in 2004 that satisfies the minimum quality threshold of the review. Furthermore, the number of papers published in some years (e.g., 2013) was actually higher than the ones presented in Fig. 2; however, some of those papers were excluded throughout the assessment of quality criteria. It is worth mentioning that many studies might not be made available by search engines until the time the search was performed (August 2022), and thus, we did not consider these studies in this study. We have specified these studies in the replication package. The overall trend that the number of publications increases is quantified by all entries in DBLP for each year, as shown in Fig. 2 for comparison. As we see in this figure, the trend in SPL testing is well above in several years (2014, 2016, 2017, and 2019). However, this trend has been decreasing in recent years.

Distribution per venue: Most of the primary studies were published in conferences; of 65 conference papers, 17 papers (~26%) were published in SPLC[2], which is the most representative conference for the SPL engineering area. This indicates that SPLC is an important

[2] SPLC stands for Software Product Line Conference.

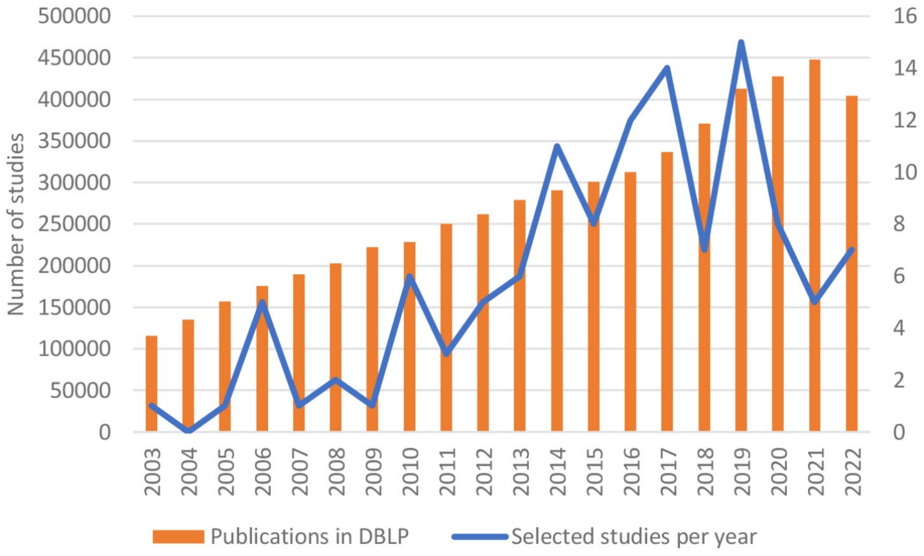

Fig. 2 Distribution of primary studies by year

venue for SPL research, and most primary studies in this field are presented in SPLC. Also, 31% of studies were published in journals, 7% in symposiums, and 5% in workshops.

4.1.2 Analyzing the evidence available to adopt the proposed approaches

As reported in the title or the text of the studies, case studies, experiments, and expert surveys are the specific methods that have been used for evaluating primary studies. Most of the primary studies were evaluated by conducting an experiment (~ 58%). It is worth mentioning that five studies applied more than one evaluation method, including case study and expert survey (Bucaioni et al. 2022), case study and experiment (Akbari et al. 2017; Fragal et al. 2019), experiment and expert survey (Hervieu et al. 2016), and case study, experiment, and expert survey (Wang et al. 2017). Table 3 shows the primary studies that have used each type of evaluation method.

Although the studies reported that their proposed approaches were evaluated by using the mentioned empirical methods, we need to analyze the strength of the evidence available to adopt the proposed approaches. The results of this analysis can help researchers to find new topics for empirical studies, and practitioners to assess the maturity of a proposed approach. Kitchenham and Charters (2007) classified the study design into five levels, based on the evidence presented in medical research.

Alves et al. (2010) revised the classification to be applicable in their study; the revised classification is fully applicable in our review. The following hierarchy is used in our study (from weakest to strongest):

1. No evidence.
2. Evidence obtained from demonstration or working out toy examples.
3. Evidence obtained from expert opinions or observations.
4. Evidence obtained from academic studies, e.g., controlled lab experiments.
5. Evidence obtained from industrial studies, e.g., causal case studies.
6. Evidence obtained from industrial practice.

Based on the evidence evaluation scheme explained, the results of the evaluation on how much evidence is available to adopt the proposed approaches are presented in Table 16 in Appendix D. All the studies have been evaluated by one kind of evaluation method. Academic studies (Lev4) are the most used evaluation method (60%), where open-source repositories are usually utilized to assess the proposed approaches. Following is Demonstration (Lev2) (~ 17%). Only a small number of studies have been evaluated by using industrial systems or real data sets (~ 16%) (Industrial studies, Lev5), or by applying the proposed methods to industrial settings and by involving industrial professionals (~ 13%) (Industrial practice, Lev6). This analysis shows an overall low level of evidence in the SPL testing field that is in line with the results of the SLR conducted by do Carmo Machado et al. (2014).

Table 3 Distribution of primary studies by the type of evaluation method

Evaluation method	Studies
Experiment	S1-S64, S111, S112, S114, S115, S117, S118
Case study	S33, S48, S64-S110, S113, S116
Expert survey	S61, S64, S85

4.2 Test levels executed throughout the SPL lifecycle (RQ2)

We divided SPL testing according to the two common phases of SPLE: Domain Engineering and Application Engineering. Based on the analysis of the studies, there are two types of testing activities that are performed during Domain Engineering: (1) developing test assets so they can be instantiated in Application Engineering, (2) applying tests to assets produced during Domain Engineering to detect faults in common core assets as soon as possible. By analyzing studies that are focused on the second activity, we identified two levels of tests usually performed in Domain Engineering; distribution of studies based on the test levels is shown in Table 4:

- **Unit testing**: Out of 118 studies, three studies are only focused on this level of testing (Jaring et al. 2008; Kim et al. 2011, 2012). Jaring et al. (2008) classified test levels based on the binding time of variabilities. Based on this study, unit tests are performed before variant binding; therefore, we included this study in this classification since Application Engineering is the phase in which variabilities are bounded to derive a specific product. Kim et al. (2011) and Kim et al. (2012) proposed specific methods in which analysis on the code level is performed to generate test suits for testing common parts of an SPL in Domain Engineering.
- **Integration testing**: Execution of integration tests in Domain Engineering are examined in the studies by Reis et al. (2007), Neto et al. (2010) and Akbari et al. (2017). Reis et al. (2007) proposed a model-based, automated technique for integration testing in Domain Engineering. In the proposed technique, integration test case scenarios are generated to support the test of interactions between the components of an integrated sub-system; placeholders are also created for necessary variable parts and all components that are not part of the integrated sub-system. Neto et al. (2010) presented a regression testing approach for SPL architectures to maintain the correctness and reliability of the architecture after modifications; as the main purpose of the approach is to verify the integration among modules and components that compose the SPL architecture, we included this study in this classification. Akbari et al. (2017) proposed a method for prioritized selection and execution of integration test cases in both Domain Engineering and Application Engineering.

Table 4 Distribution of primary studies by the testing level

Phase		Studies
Domain Engineering	Unit testing	S6, S16, S66
	Integration testing	S2, S10, S48
Application Engineering	Unit testing	S23, S54, S62, S99, S106, S113
	Integration testing	S3, S9, S11, S13, S23, S33, S38, S40, S43, S46, S47, S48, S58, S62, S65, S66, S71, S74, S87, S88, S90, S93, S94, S96, S99, S110, S113
	System/ Acceptance testing	S1, S20, S32, S43, S46, S47, S49, S65, S66, S67, S68, S69, S70, S71, S72, S74, S79, S80, S81, S82, S83, S84, S85, S86, S89, S90, S102, S112

Specific testing activities that are conducted in Application Engineering are: Creating specific product test assets by selecting and instantiating domain test assets, designing additional product-specific tests, and executing tests (Da Mota Silveira Neto et al. 2011). It is worth mentioning that some of the studies are focused on reducing the number of products that need to be tested by using specific techniques like pairwise testing (e.g., Matnei et al. 2016). In addition, some studies are focused on product prioritization to enhance the efficiency of SPL testing (e.g., Parejo et al. 2016). Once a set of configurations/products are selected/prioritized for testing, their behavior needs to be tested using a specific mechanism, e.g. executable unit tests (Parejo et al. 2016). Studies that are focused only on the first step (selecting/prioritizing configurations) do not usually consider a specific level of test. The testing levels usually performed in Application Engineering, as shown in Table 4, are as follows:

- **Unit testing**: Some of the studies considered executing unit tests in Application Engineering (Bürdek et al. 2015; Li et al. 2018; Souto and d'Amorim 2018; Jung et al. 2019, 2020; Lochau et al. 2014). Bürdek et al. (2015) proposed a white-box test-suit derivation mechanism for SPLs, specifically for unit testing, in which test specifications are extended with a presence condition. A presence condition constrains the set of configurations for which a specific test case is valid; this information is used for testing configurations in Application Engineering. Li et al. (2018) investigated test cases generated for one product that are reused for another product of the SPL by applying two categories of structure-based criteria, control-flow and data-flow. Souto and d'Amorim (2018), Jung et al. (2019) and Jung et al. (2020) identify unit test cases to be selected for regression testing.
- **Integration testing**: As shown in Table 4, this level of testing has been considered in a greater number of studies (27 studies). Some studies have not explicitly mentioned this level of testing; however, they mentioned that the untested parts of the framework are tested during Application Engineering (Scheidemann 2006; Al-Dallal and Sorenson 2008; Jaring et al. 2008). Some of the studies consider the selection of integration test cases during Application Engineering (e.g., Jung et al. 2019).
- **System /Acceptance testing**: This level of testing has also been considered in a greater number of studies (28 studies), as shown in Table 4. In most studies, test models designed throughout Domain Engineering are instantiated to derive specific system test cases (e.g., Olimpiew and Gomaa 2009). Arrieta et al. (2015) split the lifecycle of cyber-physical systems product lines into three phases: Domain Engineering, Application Engineering, and Simulation phases. Execution of system test cases are performed in the simulation phase; however, as we classified SPL lifecycle into Domain Engineering and Application Engineering, we included this study in this category.

4.3 Creating test assets by considering commonalities and variabilities (RQ3)

Creating test assets by considering commonality and variability to enhance their reusability and to reduce the probability of undetected errors in common core assets by testing them as early as possible is essential in SPL testing. Out of 118 papers, 25 primary studies (~21%) provide contributions to handle variability in a range of different manners. We conducted an exploratory analysis to identify shared characteristics among the approaches and subsequently categorized them. We identified three categories of approaches, including model-, specification-, and requirements-based approaches. The distribution of studies based on these categories is shown in Table 5.

- **Model-based approaches**: In model-based approaches, a set of techniques is used to design and execute tests for SPLs by leveraging formal or semi-formal models of the SPL's variability. In the examined studies, the subsequent methods are employed to incorporate variability into test models:

  - **Adaptation of UML models or integrating them with the feature model to produce test models including variability**: In studies (Reuys et al. 2005, 2006; Reis et al. 2007; Olimpiew and Gomaa 2009), activity diagrams are extended using specific mechanisms (e.g., stereotyping specific elements) to contain variabilities and then used as test models to create domain test case scenarios. Ebert et al. (2019) developed a common platform in Domain Engineering that contains all elements required for producing products. This study uses the SMArDT methodology (Drave et al. 2019) to elaborate each functionality defined in the platform via an extended version of activity diagram; generic test cases are then created for each functionality based on the SMArDT methodology. Reis et al. (2006) propose the ScenTED-PT technique in which the requirements and the architecture of the system are specified by UML models supplemented with performance requirements; then, they create a test model from which performance test case scenarios are derived.

    Lochau et al. (2012a) and Lackner et al. (2014) proposed to use the statechart modeling approach as a basis for capturing commonalities and variabilities of product implementations in an SPL; a 150% statechart model and the feature model is integrated to produce a reusable test model. The 150% statechart model is a model that contains the behavioral specification fragments of every feature without considering constraints between features, and the 100% statechart model is a specific instantiation of the 150% model by considering the dependencies and constraints (Lochau et al. 2012a).

  - **Using/defining different modeling notations to capture variabilities and using them to produce test assets**: In this category of model-based approaches, specific modeling notations have been used or defined to create variant-rich test models. Tuglular et al. (2019) introduced Featured Event Sequence Graphs (FESGs) to explicitly capture behavioral variability in SPLs. Gebizli and Sözer (2016) used hierarchical Markov chains to model system usage; as this model captures all

Table 5 Distribution of primary studies to answer RQ3

Approaches	Studies
Model-based	S2, S32, S33, S46, S47, S68, S70, S71, S72, S74, S76, S81, S84, S85, S90, S94, S95, S113
Specification-based	S12, S69
Requirement-based	S1, S49, S67, S80, S102

possible usage scenarios for a family of systems, it is considered as a reference test model. Bucaioni et al. (2022) define specific metamodels and languages to capture test variabilities, including SPL metamodel (SPLmm), Products metamodel (Pmm), Weaving metamodel (Wmm) to link features and signals in Pmm to those in SPLmm, Test case DSL (TcDSL), and Test Script generation Transformation (TsT). Fragal et al. (2019) use Featured Finite State Machines (FFSMs) to represent the abstract behavior of an SPL; in this study, the HSI method (Luo et al. 1995) has been extended to generate a single configurable test suite for an SPL. Luthmann et al. (2019a) extended the concept of Timed Automata (TA) by feature constraints and configurable parameters to facilitate efficient verification of real-time properties for SPLs. Lochau et al. 2012b), Lachmann et al. (2016), and Lity et al. (2019) apply the principles of delta modeling (Schaefer et al. 2010) to state machine test models to explicitly capture behavioral commonality and variability between product variants and then their test assets. In delta-oriented testing techniques, a product is considered as a base product and delta modules specify changes that should be applied to the base product to produce new ones (Schaefer et al. 2010). Beohar and Mousavi (2016) introduce the concept of Input-Output Featured Transition Systems (IOFTSs); IOFTSs are labeled transition systems with logical constraints on the presence or absence of features and are used as test models. In the work by Lochau et al. (2014), they introduced delta-oriented architecture test modeling as a means to systematically reuse common component and integration test elements across various system variants. They employed delta-oriented test artifact reuse and regression test planning to facilitate the systematic evolution of variable test elements among incrementally tested versions and/or variants of a software system.

- **Specification-based approaches**: In these approaches, specific links are defined between different configurations of an SPL and, therefore, between test cases designed for both shared and variable components of the products. Mishra (2006) uses the process algebraic specification language CSP-CASL (Roggenbach 2006) to formally specify the system; then, enhancement relationships are established between the specifications of products. In this way, test cases generated for the common parts are reused between products, and new test cases are generated for the differences in the specification. Uzuncaova et al. (2010) describe properties of features as first-order logic formulas in Alloy (Jackson 2012); by considering a product as a base, test cases are generated for the base product using Alloy Analyzer. For each new product, the test cases from previous products are reused/refined based on the differences in the specifications.
- **Requirement-based approaches**: In these approaches, variability is considered as early as possible so that it can be used to design test cases. In several primary studies, use case modeling is the approach used for representing requirements (Nebut et al. 2006; Araújo et al. 2017; Hajri et al. 2020). Nebut et al. (2006) enhance use cases with parameters and contracts used for presenting variability at the level of requirements; test-related artifacts (e.g., test objectives, test scenarios, and behavioral test patterns) are produced based on the enhanced use cases. Araújo et al. (2017) express use case specifications in a controlled natural language by considering variabilities; the specifications are then used for generating test procedures and their input and output. Hajri et al. (2020) propose to use the Product line Use case modeling Method (PUM) that supports

variability modeling in use case diagrams; by using the requirement traceability mechanism, test cases for a new product are generated by reusing/adapting existing test cases or by defining new test cases.

Kang et al. (2015) propose a method called Systematic Software Product Line Test - Data (SSPLT-D) in which a set of platform test requirements are first defined throughout Domain Engineering and then platform test scenarios, platform test cases, and platform test data are created based on test requirements. Nebut et al. (2003) propose to derive a set of behavioral test patterns from the requirement model and then use them to produce product-specific test cases.

4.4 Dealing with configuration-aware software testing (RQ4)

Dealing with configuration-aware software testing, i.e., detecting valid and invalid combinations of configuration parameters, is paramount in SPL approaches because testing all combinations of SPL functionalities would be impossible and unnecessary. In our investigation, 41 out of 118 papers (~35%) have addressed this. These papers have employed three methods to distinguish between valid and invalid configurations; distribution of studies based on these methods is shown in Table 6:

- **Using/proposing specific approaches/algorithms/tools to produce valid configurations**: Some studies utilize constraint programming, which is used for solving and modeling constraint satisfaction problems, to generate configurations that satisfy all cross-tree constraints imposed by the feature model (Hervieu et al. 2011; Marijan et al. 2013). In the same way, Kim et al. (2013) and Akbari et al. (2017) propose a constraint handling approach to produce valid configurations; as an example, an algorithm called SPLat is proposed in study (Kim et al. 2013) that dynamically prunes irrelevant configurations by handling constraints.

  Using formal methods to check cross-tree constraints defined in feature models to check the relations between features is another way to find and produce valid configurations (Lackner et al. 2014; Lopez-Herrejon et al. 2014; Beohar and Mousavi 2016; Parejo et al. 2016; Ferrer et al. 2017, 2021; Akimoto et al. 2019; Arrieta et al. 2019; Jakubovski Filho et al. 2019; Luthmann et al. 2019b; Ibias et al. 2022). For example, Lackner et al. (2014) transform a feature model into propositional formulas so that any variable assignment that satisfies the formula is a valid configuration for the product line.

Table 6 Distribution of primary studies to answer RQ4

Approaches	Studies
Specific methods/algorithms	S4, S7, S14, S15, S17, S25, S29, S32, S33, S34, S39, S41, S45, S46, S48, S51, S52, S53, S56, S57, S58, S61, S74, S77, S79, S91, S95, S100, S101, S103, S106, S108, S112, S117, S118
Runtime analysis	S71, S111, S113, S114
Manual analysis	S72, S73

Several studies suggest the utilization of sampling algorithms and techniques to generate valid configurations (Oster et al. 2010; Lochau et al. 2012a; Patel et al. 2013; Yu et al. 2014; Al-Hajjaji et al. 2016, 2019; Lee and Hwang 2019). Combinatorial Interaction Testing (CIT) is among the commonly used sampling algorithms to exclude invalid interactions between features; in CIT, design-time decisions for variability are considered to exclude invalid interactions between features. For example, Oster et al. (2010) and Lochau et al. (2012a) propose a pairwise algorithm in which dependencies and constraints between each pair of features are considered to generate all possible products that cover all valid pairs of features and their potential interactions. In a study conducted by Saini et al. (2022), they introduced a distance-based method for recognizing invalid configurations. This approach involves an initial phase where specific CIT algorithms are employed to generate real configurations. Following that, desired configurations are created, considering the availability of features in the configurations. The approach distinguishes valid from invalid configurations by applying a comparison technique to assess the differences between the actual and desired configurations.
Additionally, several studies proposed tool support for their specific approaches. They used SAT solvers to generate configurations to satisfy the feature model constraints which, in turn, reduces the configuration space to be tested (Henard et al. 2013, 2014a, b; Galindo et al. 2016; Hervieu et al. 2016; Souto and d'Amorim 2016; Fragal et al. 2019; Luthmann et al. 2019a; Krieter et al. 2020; Xiang et al. 2022). Using or implementing a tool or toolkit to produce valid configurations has been proposed by Ensan et al. (2012), Al-Hajjaji et al. (2016), Arrieta et al. (2016), Al-Hajjaji et al. (2019) and Arrieta et al. (2019). For example, FeatureIDE has been used in studies by Al-Hajjaji et al. (2016), Arrieta et al. (2016), Al-Hajjaji et al. (2019), and Arrieta et al. (2019); this tool can generate valid configurations manually and automatically.

- **Runtime analysis**: An alternative category of methods employs runtime analysis to differentiate intended from unintended interactions. In these methodologies, rather than relying on pre-established specifications to detect interactions, they examine runtime data to distinguish valid and invalid interactions (Reuys et al. 2006; Lochau et al. 2014; Rocha et al. 2020; Vidal Silva et al. 2020). As an example, in a study by Rocha et al. (2020), they introduced an iterative technique called VarXplorer to inspect interactions as they emerge. When provided with a test case consisting of system inputs, VarXplorer generates a Feature Interaction Graph (FIG), which is a concise representation of all pairwise interactions among features. This FIG offers a visual depiction of the features that interact, the contextual data, and the relationships between features, including cases where one feature suppresses another. By employing an iterative approach to interaction detection, developers and testers can thoroughly analyze the FIG derived from all the test cases within a test suite.

It is worth mentioning that some studies only stated that the feature model is manually analyzed to consider feature dependencies and feature grouping constraints (Olimpiew and Gomaa 2009; Cabral et al. 2010).

4.5 Preserving traceability between test assets and other artifacts (RQ5)

One of the essential factors in SPL testing is the preservation of the traceability between test assets and other artifacts throughout the SPL lifecycle. This is due to enhancing the reusability of test assets for managing the SPL testing complexity. However, in this regard, a few papers take preserving traceability into account, only 14 out of 118 (∼ 12%). We categorized these papers according to the type of the artifacts linked to test assets; distribution of studies based on this classification is shown in Table 7:

- **Preserving traceability between requirements and test assets**: In the majority of the studies, traceability is established between requirements, often represented using UML models (primarily use cases), and various test assets. These papers have utilized various methods, encompassing the gradually refinement of UML models into test models, direct mapping of requirements to test assets, annotation-based traceability, and the application of specific tools for automated tracing.

  Reuys et al. (2005), Nebut et al. (2006), Reis et al. (2007) and Olimpiew and Gomaa (2009) use UML models to preserve the traceability between requirements and test case scenarios. In the same way, Reuys et al. (2006) enabled the traceability between different artifacts (use cases, use case scenarios, architecture scenarios, and test case scenarios) by refining use case scenarios into test case scenarios.
  The manual definition of links between use cases and system test cases was mentioned by Hajri et al. (2020). Lackner et al. (2014), Gebizli and Sözer (2016) and Wang et al. (2017) created mapping relationships between variabilities modeled via the feature model and the test model to preserve traceability between requirements and test assets. Bucaioni et al. (2022) employed a metamodel to create a link between the product models and the SPL model. In this approach, the shared functionalities of the SPL are represented through a class diagram, and test cases are generated explicitly for these shared functionalities.
  Adding annotations to test assets to specify their relationship with other artifacts is the approach proposed by Marijan et al. (2017); in this approach, test cases were manually annotated using tags and related to one or more test requirements; this traceability information is then used to assess the quality of test cases with respect to the requirements coverage.
  In some studies, specific tools are used for automated tracing (Reis et al. 2006; Lochau et al. 2012a). Reis et al. (2006) use a tool named Mercury TestDirector to preserve the traceability between requirements specification, domain performance test case scenarios, and application performance test case scenarios. Lochau et al. (2012a) employed Rhapsody ATG to enable traceability between requirement models and test artifacts in an automated manner.

Table 7 Distribution of primary studies to answer RQ5

Approaches	Studies
Traceability between Requirements and Test Assets	S1, S2, S32, S64, S68, S70, S71, S72, S74, S84, S85, S102, S110
Traceability between Configurations and Test Assets	S69

- **Preserving traceability between configurations and test assets**: The solution proposed by Mishra (2006) is the definition of enhancement relationships between specifications of systems (different configurations of the SPL) and, therefore, their test cases.

It is also worth mentioning that some studies have emphasized the importance of preserving traceability between test assets and other artifacts, but they provide no mechanism in this regard (Kang et al. 2015; Aduni Sulaiman et al. 2019).

4.6 Testing non-functional requirements in SPL (RQ6)

In addition to functional requirements, there are non-functional requirements which should be tested in SPL, but only 3 out of 118 studies consider them. Various categories of NFRs have been addressed in these studies, including load testing and performance profiling (Reis et al. 2006), NFRs at the hardware-in-the-loop level (Arrieta et al. 2016), and real-time properties (Luthmann et al. 2019a).

Reis et al. (2006) propose a technique which concentrates on load testing and performance profiling. They employ the Object Management Group's UML Profile (Fomel 2002) to model performance aspects. Testing NFRs as a critical aspect of cyber-physical systems is investigated at the hardware-in-the-loop level by Arrieta et al. (2016); these requirements (e.g., the usage of memory and CPU) are modeled via the feature model and their coverage is considered by using selected test cases and the simulation process. Luthmann et al. (2019a) present configurable parametric timed automata to extend the expressiveness of featured timed automata to enable efficient family-based verification of real-time properties (e.g., synchronization and execution time behaviors); the proposed modeling formalism aims to represent the behavioral variability of time-critical product lines and consider the minimum/maximum delay coverage.

4.7 Controlling cost/effort of SPL testing (RQ7)

As the cost/effort of SPL testing remains a significant concern within SPLE, numerous studies have proposed various techniques to address this issue. However, the lack of a standardized classification for these techniques has made it challenging to analyze them effectively. One notable exception is the extensive research conducted on product sampling techniques, which has been categorized into specific sub-techniques, including automatic selection, semi-automatic selection, and coverage (Varshosaz et al. 2018). In our analysis, we utilized these established categories to organize the diverse range of techniques proposed in the literature.

While reviewing the papers, we identified other approaches that offer potential solutions to managing the cost and effort associated with SPL testing. These approaches were categorized based on their primary contributions and grouped into distinct categories. Some of the identified approaches focus on the reuse of test assets, either from a core asset base or from previously tested products. Others provide varying degrees of automation, ranging from the implementation or utilization of specialized tools to the automation of specific testing processes, such as specification-based approaches.

Additionally, a subset of studies explored strategies for prioritizing the execution order of SPL configurations or products and the associated test cases. Another category of research aimed to minimize the size of the test suite required for testing a particular product, thereby reducing overall testing effort.

It is important to note that these techniques can often be combined. For example, test prioritization and minimization techniques can be used with sampling techniques to further optimize the cost and effort associated with SPL testing. Furthermore, the list of techniques can be enriched concerning new publications regarding SPL testing. In the rest of this section, the details of these five techniques are provided:

- **Reusing test assets**: Based on the analysis of studies, test assets (e.g., test cases and test results) are reused in two ways, including:
  - **Reusing test assets from a core asset base**: In some studies, domain test scenarios containing variabilities are created in Domain Engineering; some of these scenarios are reused, and some of them are adapted based on the application requirements (Nebut et al. 2003; Reuys et al. 2005, 2006; Reis et al. 2006). Some other studies are focused on reusing test cases by selecting them from a repository based on the application requirements (Arrieta et al. 2016; Wang et al. 2017; Lima et al. 2020) or by binding variabilities defined in abstract test cases based on specific criteria (e.g., coverage criteria) (Al-Dallal and Sorenson 2008; Olimpiew and Gomaa 2009; Lackner et al. 2014; Bürdek et al. 2015; Kang et al. 2015; Ebert et al. 2019; Fragal et al. 2019; Luthmann et al. 2019a).
  - **Reusing test assets between products**: In some studies, test assets are reused between products by analyzing differences between the current product and previously tested products (Mishra 2006; Uzuncaova et al. 2010; Neto et al. 2010; Lochau et al. 2012b, 2014; Xu et al. 2013; Lachmann et al. 2015, 2016; Beohar and Mousavi 2016; Fragal et al. 2017; Li et al. 2018; Ebert et al. 2019; Lity et al. 2019; Luthmann et al. 2019a; Tuglular et al. 2019; Hajri et al. 2020). The technique usually used in these studies is the delta-oriented testing technique, based on regression testing principles and delta modeling concepts. By considering delta modules, test cases and test results from previously tested products can be reused and adapted for the new product.

- **Providing a specific level of automation**: We found two ways by which the studies provide a particular level of automation:
  - **Implementing/using a specific tool(s)**: In 49 studies, authors claimed that their proposed approach is automatically performed using specific tools. However, the majority of these studies fail to provide any details regarding the specific tools employed for this purpose (e.g., Reis et al. 2006; Olimpiew and Gomaa 2009; Calvagna et al. 2013; Li et al. 2018; Safdar et al. 2021). Table 8 shows that only 19 of these studies have provided online access to their tools. It is worth noting that most of these tools are in the form of research prototypes. Instead of developing a novel tool tailored to their proposed approach, some studies utilize a set of pre-existing tools at various stages of their approach. For instance, in the case of Parejo et al. (2016), the Combinatorial tool and Feature Model Testing System (FMTS),

Table 8 The list of tools implemented in the studies (last accessed October 2023)

Study	Name of the tool implemented	Link	License	Year of publication	Con-tinued
S3	JFramework Re-Tester	NA*	NA	2008	NA
S4	PACOGEN	http://people.rennes.inria.fr/Arnaud.Gotlieb/resources/Pacogen/Pacogen.html	CeCI-LL-C	2011	NA
S17	SPLat	NA	NA	2013	NA
S18	VANE	https://github.com/ViViD-DiverSE/VANE	None**	2014	No***
	SRTST	https://github.com/psjung/SRTST_experiments	None	2019	No
S25	LOOKUP	NA	NA	2014	NA
S32	SPLTestbench	NA	NA	2014	NA
S33	ConFTGen	https://github.com/vhfragal/ConFTGen-tool	None	2017	No
S38	FORMAT	https://github.com/brcoztn/AgileSWDevelopment	None	2017	No
S41	FeatureIDE (extended)	https://featureide.github.io/	L-GPL	2020	NA
S46	CoPTA	https://www.es.tu-darmstadt.de/es/team/lars-luthmann/copta-analysis	NA	2019	No
S47	IMoTEP	http://www.dfg-spp1593.de/imotep/	NA	2019	NA
S49	CARNAUbA	https://sites.google.com/site/use2testswithcnl/	NA	2017	No
S51	FeatureIDE (extended)	https://github.com/FeatureIDE/FeatureIDE/tree/develop/plugins/de.ovgu.featureide.fm.core	L-GPL	2016	NA
S54	CPA/TIGER	https://www.sosy-lab.org/research/cpa-tiger/	NA	2015	No
S55	Flower/C	NA	NA	2016	NA
S61	PACOGEN (extended)	http://people.rennes.inria.fr/Arnaud.Gotlieb/resources/Pacogen/Pacogen.html	CeCI-LL-C	2016	NA
S62	CPLTE	NA	NA	2020	NA
S63	FMTS	NA	NA	2017	NA
S64	IPT	NA	NA	2017	NA
S68	ScenTED-DTCD	NA	NA	2005	NA
S78	CIA Tool Suite	https://cloud.tu-braunschweig.de/s/fl-fiEgKYFwPAnciei651aGBg5A	NA	2020	No
S83	SPL-AT-Gherkin Feature File Generator	https://github.com/esg4aspl	MIT	2020	No
S96	DELTARX	NA	NA	2015	NA
S101	CoPTA (extended)	https://www.es.tu-darmstadt.de/es/team/lars-luthmann/copta-analysis	NA	2019	No
S102	PUMConf	https://sntsvv.github.io/PUMconf/	None	2016	No
S104	TEMSA	NA	NA	2015	NA
S108	FeatureIDE (extended)	https://featureide.github.io/	L-GPL	2019	NA
S109	VIBeS	https://projects.info.unamur.be/vibes/	NA	2017	NA

Table 8 (continued)

Study	Name of the tool implemented	Link	License	Year of publication	Con-tinued
S110	TITAN	NA	NA	2017	NA
S115	FeatureIDE (extended)	https://featureide.github.io/	L-GPL	2022	NA

* Unidentified; no information was found

** The license has been designated as “none” on GitHub

*** Discontinued

Table 9 Distribution of primary studies based on the techniques used for product sampling

		Studies
Automatic selection	Greedy	S24, S51, S65, S73, S101, S105
	Meta-heuristic search	S14, S15, S21, S25, S27, S28, S35, S37, S42, S45, S50, S52, S53, S55, S59, S60
Semi-automatic selection		S19, S58, S61, S100, S117
Coverage		S4, S5, S6, S7, S8, S18, S29, S31, S32, S39, S41, S44, S56, S72, S73, S74, S75, S77, S87, S97, S115, S116

as introduced by Ferreira et al. (2013), were employed to derive pairs and calculate solution fitness, respectively.

– **Using specific techniques** that help automate the testing process. Specification-based testing was used in some studies (e.g., Mishra 2006) as an appropriate step in automating the testing process because of its precise nature in describing the desired properties of the system under test by using a formal language. Model-based testing is another approach that helps automate the testing process. For example, Bucaioni et al. (2022) introduced a model-based approach in which test scripts are generated from shared SPL features by model transformation.

- **Handling the selection of products to test**: Testing all possible combinations of features is almost impossible in terms of resources and execution time (Cohen et al. 2006). Specific approaches have been proposed to determine a minimal set of configurations so that the correctness of the entire family can be inferred by successful verification of this set. Through our examination of the studies, we have identified diverse techniques for choosing a subset of products. These techniques have been categorized according to the provided categories for product sampling in study (Varshosaz et al. 2018). Distribution of studies based on these techniques are shown in Table 9:

  – **Automatic selection**: There are two general types of automatic selection techniques, including Greedy and Meta-heuristic search:

    – **Greedy**: Greedy algorithms (Vazirani 2001) are focused on finding an optimal solution by an iterative approach. In the context of SPLs, the optimal solution is the configuration most close to the optimum. Specific measures are used to determine a configuration as an optimum solution in each iteration (e.g., requirements/feature coverage).

- **Meta-heuristic search**: In this category, the problem of identifying a subset of products is considered as an optimization problem. Meta-heuristic algorithms are designed to target this problem by employing computational search within the configuration space to find an optimal subset of products (Varshosaz et al. 2018). Some studies have applied Evolutionary Algorithm, Random Search, and Genetic Algorithm by using an aggregation function of different objectives such as cost, number of products, number of revealed faults, pairwise coverage, and mutation score (e.g., Ensan et al. 2012). Some other studies propose to use multi-objective algorithms (e.g., Matnei et al. 2016). Hyper-heuristics are another category of approaches that have been explored in some studies to solve the problem of product sampling (e.g., Strickler et al. 2016). A hyper-heuristic is a methodology that can help automate configuration of heuristic algorithms and determine low-level heuristics (Jakubovski Filho et al. 2018). To consider user preferences throughout the selection of products as well as to make use of benefits of hyper-heuristic approaches, a preference-based hyper-heuristic approach has been proposed by Jakubovski Filho et al. (2018); this approach is an example of algorithms proposed in the field called Preference and Search-Based Software Engineering (PSBSE) (Ferreira et al. 2017b).

- **Semi-automatic selection**: In semi-automated selection, various factors are considered, including the desired number of generated products, the allocated sampling time, and the level of coverage, such as coverage of feature interactions. Moreover, the complete sample set or an initial set produced by other sampling techniques may serve as a starting point for the sampling process (Varshosaz et al. 2018). As an example, Reuling et al. (2015) propose a framework for fault-based (re-)generation of configuration samples based on feature-diagram mutation. The underlying rationale for this approach is rooted in the recognition that subsets of products generated by CIT approaches can often contain numerous redundant or less significant feature combinations. Furthermore, these approaches may overlook crucial or error-prone combinations beyond t-wise, primarily due to their black-box nature, which typically lacks consideration of domain-specific knowledge, including the fault history associated with feature combinations. The authors argue that the integration of their proposed approach with pairwise CIT sampling can potentially enhance the efficiency and effectiveness of SPL testing.
- **Coverage**: Coverage criteria are frequently employed to ensure the quality of product sampling. One commonly utilized criterion is the coverage of feature interactions (Varshosaz et al. 2018). CIT techniques are focused on the interactions between different features or configuration options, as these interactions often lead to defects in software systems. These techniques are classified as greedy by Cohen et al. (2007) since they are focused on selecting a subset of configurations where each configuration covers as many uncovered combinations as possible. However, it is categorized separately in some other studies (e.g., Cmyrev and Reissing 2014). We also prefer to separate this category of techniques from greedy algorithms since they are specially

focused on covering feature interactions. The studies that provide details of either a process or an algorithm for CIT are shown in Table 9.

The most popular kind of CIT is pairwise testing (2-wise), a specialized notion of t-wise coverage; in t-wise testing, configurations are selected in a way that guarantees that all combinations of t features are tested. Kuhn et al. (2004) showed that 80% of bugs can be revealed by investigating interaction between two variables. Furthermore, for solving problems of large complexity, pairwise has proven to be most effective since finding inconsistencies in a model including only two features might be easier than investigating all combinations of features at once (do Carmo Machado et al. 2014). However, Steffens et al. (2012) revealed that the interaction of three or more features usually occurs in the SPL testing field; therefore, considering the combination of high-strength can have an important role in revealing faults. To this end, some studies claimed that their proposed approach for t-wise coverage can work with any value of t (e.g., Krieter et al. 2020). However, high-strength ($t>3$) feature interaction can lead to a large number of valid configurations and therefore complicate the problem of t-wise coverage (Qian et al. 2018). Therefore, selecting a specific value for t is usually a trade-off between cost and efficiency to reveal faults.

- **Prioritizing configurations/test cases**: Test case prioritization is focused on defining the execution order of test cases that attempts to increase their effectiveness at meeting some performance goals (Li et al. 2007; Catal and Mishra 2012). By investigating studies, we found two categories of studies in this regard:

  – Several studies propose approaches for prioritizing SPL configurations/products to be tested; these approaches are usually used as a complement for product selection/sampling techniques. In some of these studies, one or more objectives are defined for configuration prioritization (e.g., high failure rate and high overall requirement coverage) (Scheidemann 2006; Sánchez et al. 2014; Wang et al. 2014; Galindo et al. 2016; Parejo et al. 2016; Akimoto et al. 2019; Hierons et al. 2020; Pett et al. 2020; Ferrer et al. 2021); results of the evaluations conducted by Parejo et al. (2016) indicate that multi-objective prioritization typically leads to faster fault detection than mono-objective prioritization. In another category of studies, similarity between configurations with respect to feature selections is considered as a criterion for product prioritization (similarity-based prioritization) (Arrieta et al. 2015; Al-Hajjaji et al. 2017a, 2019). In these approaches, configurations are prioritized based on the dissimilarity between them so that the configuration that has the lowest value of similarity compared to previously selected configurations in terms of feature selections is chosen. Al-Hajjaji et al. (2017b) propose a delta-oriented product prioritization method as similarity-based prioritization techniques do not consider all actual differences between products; in this approach, instead of comparing products to select features, delta-modeling artifacts (Clarke et al. 2010) are used to prioritize products.
  – Some studies are focused on prioritizing test cases for products. Lima et al. (2020) propose a learning-based approach is proposed to prioritize test cases in the Continuous Integration (CI) cycles of Highly Configurable Systems (HCI). Arrieta et

al. (2015), Marijan et al. (2017), Markiegi et al. (2017), Arrieta et al. (2019) and Hajri et al. (2020) use specific criteria to prioritize the test cases (e.g., Fault detection capability, Test execution time, or Test case appearance frequency). In another category of studies, similarity-based approaches are proposed to prioritize test cases (e.g., Devroey et al. 2017; Lachmann et al. 2015; Lachmann et al. 2016). As an example, Devroey et al. (2017) propose an algorithm to generate and sort dissimilar tests to achieve good fault finding; to this end, a distance function is calculated based on the actions executed by the test case. Furthermore, to provide a good coverage of a large number of products, prioritizing test cases is also performed based on the products that may execute a test case.

- **Minimizing test suite**: This technique is focused on minimizing the test suite size for testing a product, while preserving fault detection capability and testing coverage of the original test suite. Al-Dallal and Sorenson (2008), Stricker et al. (2010), Kim et al. (2012) and Beohar and Mousavi (2016) discuss approaches in which test cases already covered during Domain Engineering or test cases related to common parts that have already been executed in previous products are ignored. Other studies propose specific approaches to reduce redundant test executions for SPL regression testing by pruning tests that are not impacted by changes (Lachmann et al. 2016; Jung et al. 2019, 2020, 2022; Souto and d'Amorim 2018).

  There are studies focused on improving test generation process to produce minimal set of test cases while achieving specific objectives (e.g., coverage and cost/time) (Patel et al. 2013; Wang et al. 2015; Gebizli and Sözer 2016; Akbari et al. 2017; Marijan et al. 2017; Aduni Sulaiman et al. 2019; Markiegi et al. 2019; Rocha et al. 2020). As an example, Akbari et al. (2017) propose a method in which features in feature model are prioritized based on the domain engineer's decisions and the constraints that exist between features; integration test cases are then produced by considering specified priorities. Furthermore, there are approaches that are not directly focused on test suit minimization; however, they help reduce redundant execution of tests for unnecessary configurations (Kim et al. 2013; Souto and d'Amorim 2018). These approaches are focused on removing the valid configurations that are unnecessary for the execution of each test.

  The distribution of studies based on the identified techniques is presented in Table 10. As observed, the majority of the studies (~62%) are focused on proposing a specific level of automation. However, many of these studies do not offer details regarding the specific tools utilized for this purpose. The second most researched category of approaches pertains to handling the selection of products to test (~39%). Following this are techniques involving reusing test assets (~25%), prioritizing configurations/test cases (~18%), and minimizing test suite size (~15%).

Table 10 Distribution of primary studies to answer RQ7

Approaches	Studies
Reusing test assets	S3, S10, S12, S13, S22, S32, S33, S40, S46, S47, S54, S64, S67-S72, S76, S79, S80, S81, S90, S94, S95, S96, S99, S102, S113
Providing a specific level of automation	S1, S3-S8, S11, S12, S14, S17, S18, S19, S21, S23, S25, S26, S27, S32, S33, S37, S38, S41, S42, S45, S46, S47, S49, S50, S51, S53, S54, S55, S57, S59, S61, S62, S63, S64, S67, S68, S69, S70, S72, S74, S75, S76, S77, S78, S82, S83, S85, S86, S87, S89, S93-S104, S108, S109, S110, S112, S113, S115
Handling the selection of products to test	S4-S8, S14, S15, S18, S19, S21, S24, S25, S27, S28, S29, S31, S32, S35, S37, S39, S41, S42, S44, S45, S50, S51, S52, S53, S55, S56, S58, S59, S60, S61, S65, S72, S73, S74, S75, S77, S87, S97, S100, S101, S105, S115, S116, S117
Prioritizing configurations/ test cases	S20, S26, S30, S34, S36, S39, S40, S42, S57, S65, S78, S94, S96, S98, S100, S102, S103, S107, S108, S109, S110
Minimizing test suite	S3, S11, S16, S17, S23, S43, S48, S62, S84, S88, S92, S94, S95, S104, S106, S110, S111, S112

5 Threats to validity

In this section, we discuss the main threats associated with the validation of this study, classified according to the categorization proposed by Ampatzoglou et al. (2019). These particular threats are categorized into three categories: study selection validity, data validity, and research validity.

5.1 Study selection validity

One of the main threats to any secondary study is its inability to guarantee the inclusion of all relevant articles in the field. To mitigate this threat, a meeting involving all researchers was conducted to discuss and refine the search scope and keywords. Then, we evaluated the validity of the search string by conducting a limited manual search to see whether the results of that manual search show up in the results obtained by running the search string.

To ensure the comprehensive identification of all relevant studies in our search process, we rigorously followed the guidelines provided by Kitchenham and Charters (2007). We conducted a bibliographic search of published literature reviews in the SPL testing field. We updated the list of studies by applying a search string to multiple digital libraries and performed the backward and forward snowballing process. Therefore, we are confident that we have provided good coverage of studies in the SPL testing field.

During the primary study selection process, to minimize potential bias in applying inclusion/exclusion criteria, these criteria were clearly defined and regularly updated in our protocol. The first author applied inclusion and exclusion criteria. However, to reduce the researcher bias, the results of this stage were validated by the second and third authors of this paper.

Regarding quality assessment, we used a set of quality criteria to examine the studies. These criteria were reused from those proposed by Dybå and Dingsøyr (2008). Two researchers participated in the application of quality assessment criteria. We also conducted regular meetings to address and resolve any conflicts that arose during the process effectively.

5.2 Data validity

One of the main threats regarding data validity is data extraction bias. Subjective bias during the data extraction process has the potential to lead to an inconsistent interpretation of the extracted data by researchers. To mitigate this risk, two researchers collaborate during the data extraction phase, conducting resolution sessions to address any emerging ambiguities. Nevertheless, due to certain studies needing more explicit details on specific aspects of SPL testing, such as test levels, we had to make subjective interpretations based on information scattered throughout these studies.

Subjective bias may also lead to the misclassification of data in response to RQ3–RQ7. Since no predefined categories were available, we adopted an exploratory approach, scrutinizing the extracted data and identifying pertinent categories. To mitigate this potential issue, we introduced a structured data extraction form, conducted quality assessments on the chosen studies, and maintained ongoing discussions to ensure consistency in the data extraction process and category definitions. However, it is essential to acknowledge the potential influence of researcher bias on data extraction and presentation within this study.

5.3 Research validity

Research validity encompasses threats identified at all stages of our SLR.

We extensively searched secondary studies, as detailed in Sect. 3.2. This approach enabled us to identify research gaps, consider the scope and definition of RQs, and gain insights into the current state-of-the-art within the domain of SPL testing.

In our exploration of potential threats to the repeatability of this SLR, we acknowledge the complexity inherent in replicating research. Specifically, we highlight the concern that other researchers may not repeat the SLR with precisely the same results. To mitigate this threat, we provided the details of the SLR methodology so that other researchers can replicate the study; furthermore, we have made all the data collected during the SLR process available online. However, as subjectivity in the studies analysis is one major issue in conducting a literature review, we cannot guarantee that researchers can achieve exactly the same results.

One serious threat to the validity of the SLR is the inability to generalize the study's results to other scenarios and application domains. We included only the studies empirically evaluated in our analysis to handle this threat. However, as most evaluations do not refer to real-world practice, the results and classifications presented in this study may not fully apply to practical settings. Moreover, our SLR intentionally focused exclusively on SPLs. This deliberate choice was made to answer specific questions tailored for SPL testing. While this focus enhances the depth of our insights into SPL testing practices, it inevitably limits the applicability of our findings to the broader context of configurable systems. The decision not to include configurable systems was strategic, considering the extensive body of literature on configurable system testing, which would have required substantial additional time and effort for comprehensive analysis.

6 Discussion

In this study, we presented a systematic review of testing approaches proposed in the SPLE field. We have investigated seven RQs:

- **RQ1: How is the research on SPL testing characterized?**

  The analysis indicates that the SPL testing field has attracted significant attention from researchers in recent years, with an increase in empirically evaluated studies. Although the overall number of publications has grown, recent years have seen a decline. Most primary studies are published in conferences, with case studies, experiments, and expert surveys being the common evaluation methods. However, the strength of evidence supporting the proposed approaches varies, with academic studies (60%) being the most common, followed by demonstrations (17%). Only a small number of studies involve industrial systems or real data sets (16%) or industrial practice (13%), indicating an overall low level of evidence in the field.

- **RQ2**. **What levels of tests are usually executed throughout the SPL lifecycle (i.e., Domain Engineering and Application Engineering)?**

  In Domain Engineering, testing activities include developing test assets for later use and testing assets to detect faults early. In Application Engineering, activities involve creating specific product test assets, designing additional product-specific tests, and executing tests. Some studies focus on reducing the number of products tested or prioritizing products to enhance testing efficiency. The distribution of studies based on test levels shows that in Application Engineering, integration testing and system/acceptance testing are the most commonly reported levels. In contrast, unit testing is less frequently reported in both phases. This indicates a strong focus on higher levels of testing in the SPL testing field, particularly in the Application Engineering phase.

- **RQ3**. **How are test assets created by considering commonalities and variabilities?**

  Creating test assets to address commonality and variability in SPL testing is crucial for enhancing reusability and minimizing faults in core assets. Our analysis categorized these approaches into three groups: model-based, specification-based, and requirement-based.
  Model-based approaches utilize formal or semi-formal models of SPL variability to design and execute tests. Specification-based approaches define specific links between different SPL configurations and test cases. Requirement-based approaches prioritize considering variability early in test case design. The distribution of studies across these categories indicates that model-based techniques are the most commonly used in the examined studies.

- **RQ4**. **How do SPL approaches deal with configuration-aware software testing?**

Dealing with configuration-aware software testing, particularly in distinguishing valid and invalid combinations of configuration parameters, is crucial in SPL approaches. Testing all possible combinations of SPL functionalities is not only impractical but also unnecessary. The studies have employed three main methods to distinguish between valid and invalid configurations: Using/proposing specific approaches, algorithms, or tools, runtime analysis, and manual analysis. The distribution of studies across these methods indicates that the majority of the studies have either proposed specific methods or algorithms or have utilized already available tools.

- **RQ5. How is the traceability between test assets and other artifacts of SPL preserved throughout the SPL lifecycle?**

  Preservation of traceability between test assets and other artifacts is a crucial factor in SPL testing as it enhances the reusability of test assets and manages the complexity of SPL testing. However, only a few papers consider preserving traceability throughout the SPL lifecycle. The papers are categorized based on the types of artifacts associated with test assets, focusing on preserving traceability between requirements and test assets as well as between configurations and test assets. The distribution of primary studies addressing this aspect highlights that most of the studies focus on preserving traceability between requirements and test assets.

- **RQ6. How are Non-Functional Requirements (NFRs) tested in SPL?**

  Testing NFRs in SPLs has been rarely examined by researchers, with only three studies addressing this aspect. These studies cover various categories of NFRs, such as load testing, performance profiling, NFRs at the hardware-in-the-loop level, and real-time properties.

- **RQ7. What mechanisms have been used for controlling cost/effort of SPL testing?**

  Various techniques have been proposed to manage the cost and effort associated with SPL testing. However, the lack of a standardized classification for these techniques has made their analysis challenging. Notably, research on product sampling techniques has been extensively categorized into sub-techniques such as automatic selection, semi-automatic selection, and coverage. Beyond sampling techniques, other approaches have emerged, categorized based on their primary contributions, including reusing test assets, providing different levels of automation, handling product selection for testing, prioritizing configurations/test cases, and minimizing the test suite size.
  These techniques are often combinable, as seen in the use of test prioritization and minimization techniques alongside sampling techniques to optimize testing cost and effort further. Moreover, the list of techniques continues to evolve with new publications on SPL testing. The distribution of studies reveals that the majority focus on proposing a specific level of automation ($\sim$62%). However, many studies lack details on the specific tools used for this purpose. The second most researched category involves handling the selection of products to test ($\sim$39%). Additionally, techniques related to reusing test

assets (~ 25%), prioritizing configurations/test cases (~ 18%), and minimizing test suite size (~ 15%) are also explored.

We only included studies empirically evaluated in our analysis. In this discussion, we emphasize the maturity of evaluations conducted in these studies, highlight the contributions of the studies in addressing the research questions, present the main findings, and propose research directions to address identified gaps. It is important to note that our SLR intentionally focused exclusively on SPLs. We deliberately excluded the broader context of configurable systems from our analysis to have a clear focus for our article. Therefore, all the findings and research gaps reported in this section are based on our analysis within the SPL testing area. We acknowledge that this might lead to missing synergies with contributions from the broader field of configurable systems. Still, we hope this SLR can be the basis for exploring these aspects in future work.

6.1 Overview of evaluation maturity and studies' contributions

Proposed approaches have been evaluated using three types of evaluation methods, including case studies, experiments, and expert surveys. However, there is variation in the scope and type of SPLs employed in these evaluations. Different types of SPLs have been employed in the evaluations, representing diverse application domains, such as embedded systems (e.g., automotive and medical systems), web-based systems, banking systems, and smartphone and vendor machine SPLs. We categorized the scope of applications employed in the evaluations into three main groups: Industrial systems with real data sets, SPLs sourced from online repositories (e.g., SPLOT repository) or extracted from existing sources, and the development of a demonstrator. It is important to note that some studies utilized more than one category of applications, for instance, both industrial SPLs and SPLs available online. Approximately 60% of the studies (71 studies) conducted evaluations using SPLs available online or derived from prior research. Around 17% (20 studies) involved the development of a demonstrator for assessing the proposed approach. Only 29% (34 studies) utilized an industrial-scale SPL (Industrial study or Industrial practice) for evaluating their approach. This issue may jeopardize the adoption of the proposed approaches in industry; therefore, proposed approaches for SPL testing need to improve from their evaluation perspective.

Discussing threats to validity is crucial in research since it helps researchers and readers understand the limitations and potential challenges associated with the study. However, an analysis of the included studies reveals that only 32 primary studies (~ 27%) extensively discussed threats to validity. In approximately 42 studies (~ 36%), the examination of threats to validity was brief. Notably, 44 studies (~ 37%) entirely neglected to address this crucial aspect.

Another aspect that is worth analyzing is the distribution of the studies based on their contribution to the research questions. Figure 3 represents the frequencies of studies according to the research questions addressed by them. It should be mentioned that some studies covered more than one topic; therefore, the total amount shown in Fig. 3 exceeds the total number of studies selected for final analysis. As seen in Fig. 3, most studies address the questions RQ7 (Controlling cost/effort of SPL testing) and RQ2 (Test levels in SPL testing). Moreover, there is notable research interest in the area of configuration-aware testing (RQ4), followed by a substantial focus on variability-aware creation of test assets (RQ3). However, some aspects of SPL testing have rarely been considered and, therefore, need

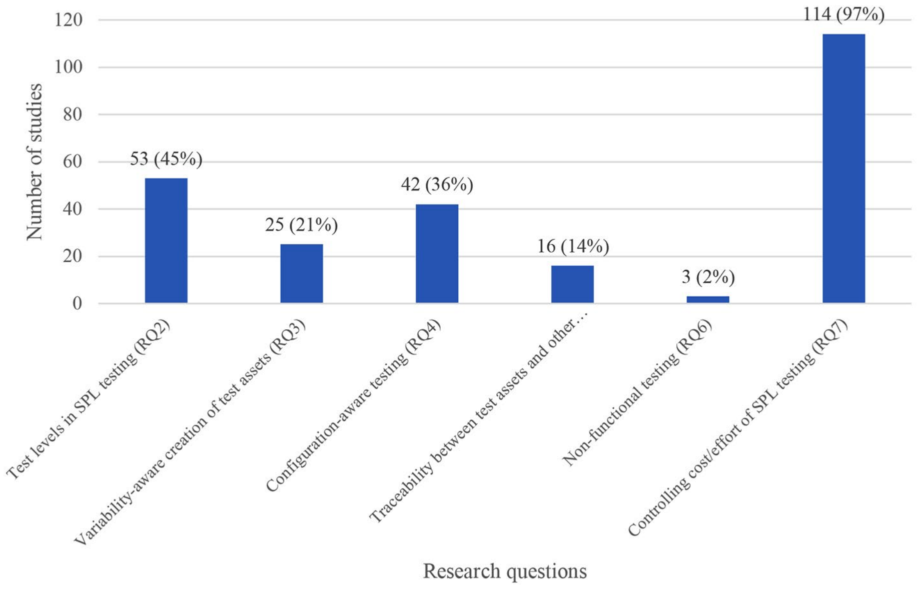

Fig. 3 Distribution of studies by the contribution to the research questions

new solutions, including RQ5 (Traceability between test assets and other artifacts) and RQ6 (Non-functional testing).

6.2 Main findings

We analyzed the data based on the content structuring/theme analysis approach of Mayring (2014). Initially, the data extracted from the extraction form provided us with a list of key challenges and sub-themes. In the next step, we inductively created categories within the themes to summarize them (analytical themes). The results of this analysis are shown in Table 11. In the rest of this section, we present various gaps and concerns that necessitate further exploration and attention from both researchers and practitioners:

- **Variability management**: Effective variability management in SPLs is crucial, yet it introduces complexities that can pose challenges to testing (Sect. 4.3). One facet that needs further exploration is the challenges associated with variability control. It demands a more in-depth investigation to identify and analyze challenges arising from the diverse features and configurations inherent in SPLs. These challenges encompass the complexities introduced by numerous potential combinations and the possibility of unforeseen interactions among variable elements. While this aspect has been previously examined, the key concern lies in the applicability of the proposed solutions and approaches in real-world scenarios. For example, one of the most investigated solutions involves selecting a subset of products for testing. However, the potential for unseen interactions between features in new products to result in faults raises doubts. Furthermore, many of the proposed approaches have only been evaluated at a proof-of-concept level, necessitating a more in-depth investigation into their suitability for industrial SPL

Table 11 The results of the analysis based on qualitative content analysis

Key Challenges	Sub-themes/ challenges (Descriptive themes)	Analytical themes (Insights)
Testing types	Non-functional testing	– Diverse nature of non-functional requirements (e.g., S70) – Impact on testing strategies (e.g., S79) – Resource intensiveness (e.g., S79) – Integration with functional testing (e.g., S46)
	Regression testing	– Impact of SPL evolution on regression testing (e.g., S10, S78) – Efficiency gains through automated regression testing (e.g., S23) – Traceability challenges in regression testing (e.g., S23) – Selective regression testing strategies (e.g., S10, S13, S78)
Tool support and empirical evaluations	Tool support	– Effectiveness and efficiency of SPL testing tools – Adaptability to evolving SPL configurations (e.g., S3) – User experience and usability of testing tools
	Industrial evaluations	– Industrial evaluations for the proposed approaches – Providing guidance for industry-specific SPL testing
SPL lifecycle and testing	Test levels throughout SPL lifecycle	– Integration of test levels across SPL phases (S20) – Impact of product line variabilities on test levels (S1, S23) – Adaptation of test Levels to changing requirements (e.g., S1, S10)
	Preserving traceability	– Challenges in traceability across configurations in SPL testing (e.g., S69) – Automated traceability tools in SPL testing (e.g., S1)
	Variability management	– Variability control challenges (e.g., S74) – Modeling variability for testing (e.g., S71) – Improving model-based approaches (e.g., S1, S74)

applications.

Another crucial aspect involves examining variability modeling. This includes an analysis of the current state of variability modeling in SPL testing and an exploration of opportunities to enhance modeling techniques to address testing challenges. While model-based approaches, commonly used to create variant-rich test assets, have shown promise in SPL testing, there is still room for improvement in automating the generation of test cases and ensuring comprehensive coverage based on variability models. Utilizing model-based approaches can automate the process of transforming high-level test assets (e.g., test scenarios) and generating low-level test assets (e.g., test cases and test data).

- **Non-functional testing**: Despite the fact that functional testing of SPLs has been extensively investigated, non-functional testing aspects need greater focus and specific methodologies (Sect. 4.6). This particular gap has already been acknowledged in previous literature reviews. Non-functional requirements encompass diverse dimensions, including but not limited to performance, security, usability, and scalability. While some studies have explored aspects such as real-time behaviors and performance, there remains a need for further research to comprehensively address diverse facets within this domain. Moreover, the inherent nature of non-functional requirements significantly shapes testing strategies. Considering their distinct characteristics and evaluation criteria, it is crucial to investigate how distinct testing approaches are essential for various aspects like performance testing, security testing, and usability testing.

  Non-functional testing, particularly in critical areas such as performance and security, poses challenges due to its resource-intensive nature. Investigating the challenges associated with acquiring and allocating resources for thorough non-functional testing throughout the SPL lifecycle is crucial for effective quality assurance.
  The complexities of seamlessly integrating non-functional testing with functional testing necessitate further exploration. Examining how the interplay between these two testing dimensions influences the overall quality assurance process will contribute valuable insights to the field.

- **Tool support**: Given the substantial testing effort required for SPLs, the availability of tools specifically designed for SPL testing is crucial (Sect. 4.7). The analysis of the studies with respect to automation provided by the tools indicates that most of the tool implementations are proof-of-concept prototypes developed for validating the proposed approach. Therefore, developing more robust and user-friendly tools can significantly help practitioners in their testing efforts. This particular challenge has previously been discussed in prior literature reviews.

  Some specific areas need further exploration. Evaluating the effectiveness and efficiency of existing SPL testing tools explores capabilities, limitations, and areas for improvement in tools designed for various testing activities within the SPL lifecycle. Analyzing how well testing tools adapt to changes in SPL configurations includes investigating their ability to accommodate evolving feature sets, configurations, and architectural variations, ensuring continued effectiveness. Assessing the user experience and usability of SPL testing tools explores how user-friendly and accessible tools are for practitioners involved in SPL testing, considering factors such as ease of use, learning curve, and user satisfaction.

- **Regression testing**: Effectively handling regression testing in SPLs, where modifications to one product can affect others, presents an intricate challenge (Sect. 4.7). Regression test selection/prioritization/minimization and architecture-based regression testing are potential points for future research. Test case selection is focused on choosing a set of relevant test cases to test the modified version of the system, and the aim of test minimization is to remove the redundant/irrelevant test cases from the existing test suit. Test case prioritization aims at ordering and ranking test cases based on specific criteria such as importance and likelihood of failure. All these techniques aim to reduce the cost/effort of SPL testing after applying any change to products or the SPL architecture.

  An important aspect is analyzing how changes and evolutions in the SPL architecture impact regression testing strategies. This investigation includes understanding the challenges of maintaining test suites across evolving SPL configurations and the need for adaptive regression testing approaches.
  Additionally, exploring the benefits and challenges of implementing automated regression testing within the SPL context is crucial. This requires an analysis of efficiency gains, potential pitfalls, and strategies to optimize the effectiveness of automated regression testing in SPL scenarios.
  Moreover, investigating challenges related to maintaining traceability between evolving codebase versions and regression test suites is critical. This requires exploring strategies to preserve traceability links, ensuring that regression testing aligns with the dynamic nature of SPL development.

- **Industrial evaluations**: Encouraging the adoption of SPL testing practices in industrial settings requires addressing practical challenges (Sect. 3.3 and 4.1). This includes offering guidance tailored for industry-specific SPL testing and conducting industrial evaluations.

  To enhance the industry adoption of SPL approaches, offering practical insights and recommendations is essential. This involves providing tailored guidance to help organizations navigate the unique challenges and requirements of adopting SPL testing methods in their specific industry domains. Additionally, there is a need to move beyond proof-of-concept evaluations and conduct practical assessments to verify the feasibility, scalability, and effectiveness of proposed SPL testing methods in diverse industrial contexts.

- **Test levels throughout the SPL lifecycle**: Exploring the details of a test level throughout the SPL lifecycle and illustrating the challenges associated with neglecting a particular test level would provide valuable insights for practitioners (Sect. 4.2). Two levels of tests are commonly executed throughout Domain Engineering: Unit testing and Integration testing. Although testing common core assets of an SPL is vital to detect faults as soon as possible, a few studies have considered the execution of tests in domain engineering. Therefore, it would be useful to conduct further investigations regarding how to execute a specific level of test in Domain Engineering and the consequences of not performing it. In Application Engineering, three levels of tests are usually executed: Unit testing, Integration testing, and System/acceptance testing. The two last levels have been investigated in most of the studies. It is worth mentioning that Unit testing has been investigated as a level of test in Application Engineering in a few studies published in recent years. In contrast, previous literature reviews have not reported this level of test in Application Engineering (e.g., Pérez et al. 2009). This indicates no consensus on the test levels executed during Domain Engineering and Application Engineering.

Another aspect that needs further exploration involves examining the influence of variabilities inherent in SPLs on different test levels. This requires understanding how the presence of variable features across products affects test activities, including planning, design, and execution at each testing level. Additionally, there is a need to investigate how test levels adapt to requirements and feature set changes throughout the SPL lifecycle. This requires exploring the challenges and opportunities associated with maintaining effective testing strategies in response to the dynamic nature of evolving product configurations.

- **Preserving the traceability between test assets and development artifacts**: Preserving traceability between test assets and development artifacts in SPLs is particularly challenging due to the complex relationships between product variants and the shared assets (Sect. 4.5). Studies that target testing SPLs (very) rarely consider traceability explicitly. Examining the challenges associated with preserving traceability is crucial, especially when dealing with evolving product configurations within the SPL testing environment. While researchers have proposed certain methods, such as Reis et al. (2007) which preserved the traceability between requirements and test case scenarios using UML models and by refining use case scenarios into test case scenarios, Reuys et al. (2006) enabled traceability between artifacts, there remains a necessity to investigate more efficient approaches for modeling and representing traceability relationships, considering feature variability and configuration management. Furthermore, exploring the creation of automated tools and techniques for establishing and consistently updating traceability links in response to the evolving nature of SPLs presents an engaging area for future research.

To compare findings with previous SLRs, Table 12 presents a summary of the findings from both the current study and prior literature reviews (Pérez et al. 2009; Engström and Runeson 2011; Da Mota Silveira Neto et al. 2011; do Carmo Machado et al. 2014).

7 Related work

This research aims to provide researchers and practitioners with an overview of state-of-the-art testing practices applied to SPL and identify the gaps between required techniques and existing approaches. Accordingly, we conducted an SLR to analyze existing approaches to SPL testing. Therefore, SLRs and SMSs on SPL testing can be considered as works related to this research. To the best of our knowledge, four papers have systematically analyzed approaches focused on SPL testing (Pérez et al. 2009; Engström and Runeson 2011; Da Mota Silveira Neto et al. 2011; do Carmo Machado et al. 2014).

Pérez et al. (2009) conducted an SLR to identify experience reports and initiatives carried out in the SPL testing area. In this work, primary studies were classified into seven categories: Unit testing, Integration testing, functional testing, SPL Architecture testing,

Table 12 Comparison of findings between current study and prior literature reviews

Findings	Current Study	Previous LRs
Non-functional testing	Emphasizes the need for greater focus and specific methodologies in addressing non-functional testing aspects in SPLs. Highlights diverse dimensions such as performance, security, usability, and scalability, with challenges in resource allocation and integration with functional testing	Emphasized the need to explore various non-functional requirements and the trade-offs between quality attributes like modularity and testability. However, they did not discuss challenges related to resource allocation and the integration of non-functional with functional testing
Regression testing	Identifies challenges such as the impact of SPL evolution on regression testing, efficiency gains through automated regression testing, traceability challenges in regression testing, and selective regression testing strategies	Challenges related to integration testing were not reported in detail
Tool support	Emphasizes the critical importance of tool support for SPL testing, highlighting the need for more robust and user-friendly tools. Specific areas for further exploration include evaluating tool effectiveness and efficiency, adaptability to changes in SPL configurations, and user experience/usability	Underscored the importance of tool support for SPL testing, without detailed exploration of specific challenges and recommendations for improvement
Industrial evaluations	Emphasizes the need for industrial evaluations of proposed approaches and guidance for industry-specific SPL testing	Acknowledged similar challenges
Test levels throughout SPL lifecycle	Identifies challenges related to integrating test levels across SPL phases, the impact of product line variabilities on test levels, and the adaptation of test levels to changing requirements	Identified the challenge of not examining the effects of not performing a test level in SPL testing. The current study, however, provides more detailed insights into specific challenges associated with test levels
Preserving traceability	Underscores the difficulties in maintaining traceability between test assets and development artifacts within SPLs due to complex relationships and the oversight of traceability in SPL testing. It stresses the necessity for improved methodologies and automated tools to establish and maintain traceability links effectively	Highlighted also the difficulties of maintaining consistency between models and test code artifacts as systems evolve, stressing the significance of updating traceability links
Variability management	Highlights the challenges associated with variability control in SPLs, emphasizing the complexities introduced by numerous potential combinations and the possibility of unforeseen interactions among variable elements. Identifies a need for more efficient approaches and automated tools for managing variability throughout the SPL lifecycle	Identified similar challenges related to variability control, such as handling test design at different levels of abstraction. Emphasized also the need for automated tools to manage the maintenance effort of variable test assets

Embedded system testing, testing process and testing effort in SPL. Then, they presented a summary of each area. The similarity of this SLR to our work is testing levels investigated in both works; however, our work is broader in scope than this SLR since we investigated more aspects of SPL testing.

Engström and Runeson (2011) conducted an SMS by analyzing papers published up to 2008. The authors mapped studies into seven categories based on their research focus: Test organization and process, Test management, Testability, System and acceptance testing, Integration testing, Unit testing, and Test automation. They also identified challenges in SPL testing and needs for future research. This SMS has similarities with our work regarding specific SPL aspects investigated, including testing levels and test automation. However, the research questions designed by Engström and Runeson (2011) are more general, focusing on specifying challenges and topics investigated in SPL testing.

Da Mota Silveira Neto et al. (2011) conducted an SMS to investigate state-of-the-art testing practices by analyzing a set of 45 publications dated from 1993 to 2009. Primary studies are mapped into nine categories: Testing strategy, Static and dynamic analysis, Testing levels, Regression testing, Non-functional testing, Commonality and variability testing, Variant binding time, Effort reduction, and Test measurement. Some of the research questions designed by Da Mota Silveira Neto et al. (2011) are similar to the ones investigated in our work (e.g., testing SPLs while considering commonalities and variabilities). However, our work is broader in scope since we analyzed 110 papers published up to 2022. Furthermore, we only included empirically evaluated studies in our review.

do Carmo Machado et al. (2014) conducted an SLR by analyzing 49 studies published up to 2013; this SLR aimed to identify testing strategies that could achieve higher defect detection rates and reduced quality assurance effort. Identifying strategies to handle the selection of products to test has been investigated in both (do Carmo Machado et al. 2014) and our work. Furthermore, similar to our work, the initial set of primary studies in study (do Carmo Machado et al. 2014) has been identified by investigating previously conducted SLRs or SMSs, published up to the year 2009; also, the authors of this SLR only included empirically evaluated studies. However, our work investigates more aspects of SPL testing (e.g., preserving traceability between test assets and other artifacts) and analyzes more studies (110 papers).

Literature reviews also specifically focused on analyzing one aspect of SPL testing. As an example, Lopez-Herrejon et al. (2015) conducted an SMS to identify techniques that have been applied for combinatorial interaction testing of SPLs. However, our work is broader in scope since we did not limit the studies to a specific technique.

In general, the previous literature reviews and our work complement each other regarding the research questions addressed. Some aspects of SPL testing have not been considered in detail in previous reviews: techniques used for preserving traceability between test artifacts and other artifacts, techniques employed for identifying valid and invalid configurations, and different ways to control cost/effort of SPL testing were not covered in an extent that makes it possible to identify the current status of research and practice from the perspective of those aspects.

8 Conclusions and future work

The goal of SPLE is to improve the effectiveness and efficiency of software development by managing commonalities and variabilities among products. Testing is an essential part of SPLE to achieve the benefits of an SPL. It is focused on detecting potential faults in core assets created during Domain Engineering and products created during Application Engineering by reusing core assets. This paper presents the results of a systematic literature review of testing in SPLE. The SLR aimed to investigate specific aspects of SPL testing that were formulated as seven research questions, identify gaps, and address specific points of SPLE that still need to be fully addressed.

The analysis that we conducted based on 118 studies from 2003 to 2022 has uncovered a range of issues and considerations that researchers and practitioners can work on. It is shown that managing variability in SPL testing is vital but can complicate the testing process. Model-based methods show promise in generating test assets, but there is room for improvement in automating test case creation and ensuring comprehensive coverage. Non-functional testing aspects like performance, security, and usability require more attention and specific methodologies. Having the right tools is important, but most tool implementations are still in the proof-of-concept stage. Regression testing poses a complex challenge, and future research should concentrate on areas like regression test selection, prioritization, minimization, and architecture-based regression testing. Establishing benchmark datasets and standard evaluation criteria for SPL testing methods would simplify comparing and adopting various techniques.

Exploring test levels throughout the SPL lifecycle and illustrating the challenges of neglecting a particular test level would offer valuable insights. Additionally, studies focusing on testing SPLs need to address traceability explicitly. Maintaining traceability between test assets and development artifacts is especially difficult due to the intricate relationships between product variants and shared assets, which requires effective approaches. It is also worth mentioning that, throughout selecting studies for final analysis, we included only the studies empirically evaluated. By analyzing the evaluation conducted in the studies, we noticed that most of the studies were assessed by applying only one empirical method. Furthermore, most of the assessments undertaken do not refer to real-world practice. This indicates the need to evaluate SPL testing approaches not in academia but in industry.

Based on the findings of this SLR, further research in the SPL testing field can be expended on specific areas we identified throughout this research as the potential points for future research (e.g., SPL regression testing). Furthermore, empirical assessment of existing techniques for the investigated aspects (e.g., selection of products to test or creating reusable test assets) to compare those techniques would be helpful for both researchers and practitioners, mainly if those techniques are applied to real-world and large-scale scenarios. Furthermore, this research can be strengthened by examining studies published in the field of testing configurable systems. Such analysis can investigate how techniques from this broader domain might be applied to SPL testing to address existing deficiencies in this area.

Appendix A

Table 13 Search strings and results

Engine	Search String	Results
IEEE Xplore	("All Metadata":"software product line" OR "All Metadata":"software product lines" OR "All Metadata":"software product family" OR "All Metadata":"software product families""") AND ("All Metadata":"test" OR "All Metadata":"testing") Filters Applied: 2013–2022	173

Table 13 Search strings and results

Engine	Search String	Results
Scopus	TITLE-ABS-KEY (("Software Product Line" OR "Software Product Family" OR "Software Product Lines" OR "Software Product Families") AND ("Test" OR "Testing")) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (LANGUAGE , "English")) AND (LIMIT-TO (PUBYEAR , 2022) OR LIMIT-TO (PUBYEAR , 2021) OR LIMIT-TO (PUBYEAR , 2020) OR LIMIT-TO (PUBYEAR , 2019) OR LIMIT-TO (PUBYEAR , 2018) OR LIMIT-TO (PUBYEAR , 2017) OR LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014) OR LIMIT-TO (PUBYEAR , 2013))	333
ACM DL	[[*Full Text*: "software product line"] AND [Full Text: "test"]] OR [[Full Text: "software product lines"] AND [Full Text: "test"]] OR [[Full Text: "software product family"] AND [Full Text: "test"]] OR [[Full Text: "software product line"] AND [Full Text: "testing"]] OR [[Full Text: "software product lines"] AND [Full Text: "testing"]] OR [[Full Text: "software product family"] AND [Full Text: "testing"]] AND [Publication Date: (01/01/2013 TO 31/12/2022)]	866
Springer	'("Software Product Line" OR "Software Product Famil*)"AND (Test*)' within English, Computer Science, 2013–2022	1016
Wiley online library	"("software product line" OR "software product lines" OR "software product family" OR "software product families") AND (test OR testing)" anywhere	220

Appendix B

Table 14 Quality assessment criteria (Dybå and Dingsøyr 2008)

No	Question	Issue
1	Is this a research paper?	Reporting
2	Is there a clear statement of the aims of the research?	Reporting
3	Is there an adequate description of the context in which the proposed approach has been applied?	Reporting
4	Was the research design appropriate to address the aims of the research?	Rigor
5	Was there a control group with which to compare the treatments?	Rigor
6	Was the data collected in a way that addressed the research issue?	Rigor
7	Was the data analysis sufficiently rigorous?	Rigor
8	Has the relationship between researcher and participants been considered to an adequate degree?	Credibility
9	Is there a clear statement of findings?	Credibility
10	Is the study of value for research or practice?	Relevance
11	Are there any practitioner-based guidelines?	Relevance

Appendix C

Table 15 Primary studies investigated in the SLR

No.	Title	Author(s)/Year	Venue
[S1]	System testing of product lines: From requirements to test cases	(Nebut et al. 2006)	Book Chapter: Software Product Lines
[S2]	Integration testing in software product line engineering: A model-based technique	(Reis et al. 2007)	FASE
[S3]	Testing software assets of framework-based product families during application engineering stage	(Al-Dallal and Sorenson 2008)	J Softw
[S4]	PACOGEN: Automatic generation of pairwise test configurations from feature models	(Hervieu et al. 2011)	ISSRE
[S5]	Properties of realistic feature models make combinatorial testing of product lines feasible	(Johansen et al. 2011)	MODELS
[S6]	Reducing combinatorics in testing product lines	(Kim et al. 2011)	AOSD
[S7]	Automated incremental pairwise testing of software product lines	(Oster et al. 2010)	SPLC
[S8]	Automated and scalable t-wise test case generation strategies for software product lines	(Perrouin et al. 2010)	ICST
[S9]	Integration testing of software product lines using compositional symbolic execution	(Shi et al. 2012)	FASE
[S10]	A regression testing approach for software product lines architectures	(Neto et al. 2010)	SBCARS
[S11]	Avoiding redundant testing in application engineering	(Stricker et al. 2010)	SPLC
[S12]	Incremental test generation for software product lines	(Uzuncaova et al. 2010)	IEEE Trans Softw Eng
[S13]	Continuous test suite augmentation in software product lines	(Xu et al. 2013)	SPLC
[S14]	Evolutionary search-based test generation for software product line feature models	(Ensan et al. 2012)	CAiSE
[S15]	Multi-objective test generation for software product lines	(Henard et al. 2013)	SPLC
[S16]	Shared execution for efficiently testing product lines	(Kim et al. 2012)	ISSRE
[S17]	SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems	(Kim et al. 2013)	ESEC/FSE
[S18]	A variability-based testing approach for synthesizing video sequences	(Galindo et al. 2014)	ISSTA
[S19]	Fault-based product-line testing: Effective sample generation based on feature-diagram mutation	(Reuling et al. 2015)	SPLC
[S20]	Test control algorithms for the validation of cyber-physical systems product lines	(Arrieta et al. 2015)	SPLC
[S21]	Recommending faulty configurations for interacting systems under test using multi-objective search	(Safdar et al. 2021)	ACM Trans Softw Eng Methodol
[S22]	Reducing the concretization effort in FSM-based testing of software product lines	(Fragal et al. 2017)	ICSTW
[S23]	Automated code-based test selection for software product line regression testing	(Jung et al. 2019)	J Syst Softw
[S24]	Supporting software product line testing by optimizing code configuration coverage	(Vidács et al. 2015)	ICSTW
[S25]	Combinatorial test generation for software product lines using minimum invalid tuples	(Yu et al. 2014)	HASE

Table 15 Primary studies investigated in the SLR

No.	Title	Author(s)/Year	Venue
[S26]	A comparison of test case prioritization criteria for software product lines	(Sánchez et al. 2014)	ICST
[S27]	Product selection based on upper confidence bound MOEA/D-DRA for testing software product lines	(do Nascimento Ferreira et al. 2016)	CEC
[S28]	Selecting products for high-strength t-wise testing of software product line by multi-objective method	(Qian et al. 2018)	PIC
[S29]	Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines	(Henard et al. 2014a)	IEEE Trans Softw Eng
[S30]	Efficient product-line testing using cluster-based product prioritization	(Al-Hajjaji et al. 2017a)	AST
[S31]	Weighted rank ant colony metaheuristics optimization-based test suite reduction in combinatorial testing for improving software quality	(Bharathi and Sangeetha 2018)	ICICCS
[S32]	Model-based test design of product lines: Raising test design to the product line level	(Lackner et al. 2014)	ICST
[S33]	Extending HSI test generation method for software product lines	(Fragal et al. 2019)	Comput J
[S34]	A prioritization method for SPL pairwise testing based on user profiles	(Akimoto et al. 2019)	APSEC
[S35]	Balancing soundness and efficiency for practical testing of configurable systems	(Souto et al. 2017)	ICSE
[S36]	Delta-oriented product prioritization for similarity-based product-line testing	(Al-Hajjaji et al. 2017b)	VACE
[S37]	Incorporating user preferences in a software product line testing hyper-heuristic approach	(Jakubovski Filho et al. 2018)	CEC
[S38]	FORMAT: A tool for adapting test models based on feature models	(Ergun et al. 2017)	COMPSAC
[S39]	CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines	(Ferrer et al. 2021)	J Heuristics
[S40]	Learning-based prioritization of test cases in continuous integration of highly-configurable software	(Lima et al. 2020)	SPLC
[S41]	YASA: Yet another sampling algorithm	(Krieter et al. 2020)	VaMoS
[S42]	Many-objective test suite generation for software product lines	(Hierons et al. 2020)	ACM Trans Softw Eng Methodol
[S43]	Derivation of test cases for model-based testing of software product line with hybrid heuristic approach	(Aduni Sulaiman et al. 2019)	IRICT
[S44]	Optimize SPL test cases with adaptive simulated annealing genetic algorithm	(Yan et al. 2019)	ACM TURC
[S45]	Preference based multi-objective algorithms applied to the variability testing of software product lines	(Jakubovski Filho et al. 2019)	J Syst Softw
[S46]	Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints	(Luthmann et al. 2019a)	J Syst Softw
[S47]	Retest test selection for product-line regression testing of variants and versions of variants	(Lity et al. 2019)	J Syst Softw
[S48]	A method for prioritizing integration testing in software product lines based on feature model	(Akbari et al. 2017)	Int J Softw Eng Knowl Eng
[S49]	Generating test cases and procedures from use cases in dynamic software product lines	(Araújo et al. 2017)	SAC

Table 15 Primary studies investigated in the SLR

No.	Title	Author(s)/Year	Venue
[S50]	Deriving products for variability test of feature models with a hyper-heuristic approach	(Strickler et al. 2016)	Appl Soft Comput
[S51]	IncLing: Efficient product-line testing using incremental pairwise sampling	(Al-Hajjaji et al. 2016)	ACM SIGPLAN Not
[S52]	Mutation-based generation of software product line test configurations	(Henard et al. 2014b)	SSBSE
[S53]	A parallel evolutionary algorithm for prioritized pairwise testing of software product lines	(Lopez-Herrejon et al. 2014)	GECCO
[S54]	Facilitating reuse in multi-goal test-suite generation for software product lines	(Bürdek et al. 2015)	FASE
[S55]	Software product line test suite reduction with constraint optimization	(Carlsson et al. 2016)	ICSOFT
[S56]	Hybrid algorithms based on integer programming for the search of prioritized test data in software product lines	(Ferrer et al. 2017)	EvoCOP
[S57]	Effective product-line testing using similarity-based product prioritization	(Al-Hajjaji et al. 2019)	Softw Syst Model
[S58]	Using ant colony optimisation to select features having associated costs	(Ibias et al. 2022)	ICTSS
[S59]	A multi-objective test data generation approach for mutation testing of feature models	(Matnei et al. 2016)	J Softw Eng Res Dev
[S60]	Multi-objective test suite optimization for incremental product family testing	(Baller et al. 2014)	ICST
[S61]	Practical minimization of pairwise-covering test configurations using constraint programming	(Hervieu et al. 2016)	Inf Softw Technol
[S62]	Efficient regression testing of software product lines by reducing redundant test executions	(Jung et al. 2020)	Appl Sci
[S63]	Software product line testing based on feature model mutation	(Ferreira et al. 2017a)	Int J Softw Eng Knowl Eng
[S64]	Automated product line test case selection: Industrial case study and controlled experiment	(Wang et al. 2017)	Softw Syst Model
[S65]	Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems	(Scheidemann 2006)	SPLC
[S66]	Modeling variability and testability interaction in software product line engineering	(Jaring et al. 2008)	ICCBSS
[S67]	Automated requirements-based generation of test cases for product families	(Nebut et al. 2003)	ASE
[S68]	Model-based system testing of software product families	(Reuys et al. 2005)	CAiSE
[S69]	Specification based software product line testing: A case study	(Mishra 2006)	CS&P
[S70]	A reuse technique for performance testing of software product lines	(Reis et al. 2006)	SPLiT
[S71]	The scented method for testing software product lines	(Reuys et al. 2006)	SPLC
[S72]	Reusable model-based testing	(Olimpiew and Gomaa 2009)	ICSR
[S73]	Improving the testing and testability of software product lines	(Cabral et al. 2010)	SPLC
[S74]	Model-based pairwise testing for feature interaction coverage in software product line engineering	(Lochau et al. 2012a)	Softw Qual J

Table 15 Primary studies investigated in the SLR

No.	Title	Author(s)/Year	Venue
[S75]	Combinatorial testing for feature models using CitLab	(Calvagna et al. 2013)	ICSTW
[S76]	Incremental model-based testing of delta-oriented software product lines	(Lochau et al. 2012b)	TAP
[S77]	Practical pairwise testing for software product lines	(Marijan et al. 2013)	SPLC
[S78]	Risk-based compatibility analysis in automotive systems engineering	(Pett et al. 2020)	MODELS
[S79]	Search-based test case selection of cyber-physical system product lines for simulation-based validation	(Arrieta et al. 2016)	SPLC
[S80]	Systematic software product line test case derivation for test data reuse	(Kang et al. 2015)	COMPSAC
[S81]	Featured event sequence graphs for model-based incremental testing of software product lines	(Tuglular et al. 2019)	COMPSAC
[S82]	Heterogeneous modeling and testing of software product lines	(Belli et al. 2021)	QRS-C
[S83]	Behavior-driven development of software product lines	(Tuglular and Coşkun 2021)	DSA
[S84]	Model-based software product line testing by coupling feature models with hierarchical markov chain usage models	(Gebizli and Sözer 2016)	QRS-C
[S85]	Model-based generation of test scripts across product variants: An experience report from the railway industry	(Bucaioni et al. 2022)	J. Softw. Evol. Process
[S86]	Executable test case generation from specifications written in natural language and test execution environment	(Aoyama et al. 2021)	CCNC
[S87]	Using combinatorial testing for distributed automotive features: Applying combinatorial testing for automated feature-interaction-testing	(Dominka et al. 2018)	CCWC
[S88]	Reducing redundant test executions in software product line testing—A case study	(Jung et al. 2022)	Electronics
[S89]	Design and implementation of a test automation framework for configurable devices	(Soe et al. 2022)	APIT
[S90]	Applying product line testing for the electric drive system	(Ebert et al. 2019)	SPLC
[S91]	Combinatorial test design using design-time decisions for variability	(Lee and Hwang 2019)	Int J Softw Eng Knowl Eng
[S92]	Test case selection using structural coverage in software product lines for time-budget constrained scenarios	(Markiegi et al. 2019)	SAC
[S93]	Risk-based integration testing of software product lines	(Lachmann et al. 2017)	VaMoS
[S94]	Fine-grained test case prioritization for integration testing of delta-oriented software product lines	(Lachmann et al. 2016)	FOSD
[S95]	Input–output conformance testing for software product lines	(Beohar and Mousavi 2016)	J Log Algebr Methods Program
[S96]	Delta-oriented test case prioritization for integration testing of software product lines	(Lachmann et al. 2015)	SPLC
[S97]	PROW: A pairwise algorithm with constraints, order and weight	(Lamancha et al. 2015)	J Syst Softw
[S98]	Multi-objective test prioritization in software product line testing: An industrial case study	(Wang et al. 2014)	SPLC

Table 15 Primary studies investigated in the SLR

No.	Title	Author(s)/Year	Venue
[S99]	Genetic algorithm-based test generation for software product line with the integration of fault localization techniques	(Li et al. 2018)	Empir Softw Eng
[S100]	Testing variability-intensive systems using automated analysis: An application to Android	(Galindo et al. 2016)	Softw Qual J
[S101]	Sampling strategies for product lines with unbounded parametric real-time constraints	(Luthmann et al. 2019b)	Int J Softw Tools Technol Transf
[S102]	Automating system test case classification and prioritization for use case-driven testing in product lines	(Hajri et al. 2020)	Empir Softw Eng
[S103]	Multi-objective test case prioritization in highly configurable systems: A case study	(Parejo et al. 2016)	J Syst Softw
[S104]	Cost-effective test suite minimization in product lines using search techniques	(Wang et al. 2015)	J Syst Softw
[S105]	Efficient and effective testing of automotive software product lines	(Cmyrev and Reissing 2014)	Appl Sci Eng Prog
[S106]	Time-space efficient regression testing for configurable systems	(Souto and d'Amorim 2018)	J Syst Softw
[S107]	Search-based product line fault detection allocating test cases iteratively	(Markiegi et al. 2017)	SPLC
[S108]	Search-based test case prioritization for simulation-based testing of cyber-physical system product lines	(Arrieta et al. 2019)	J Syst Softw
[S109]	Dissimilar test case selection for behavioural software product line testing	(Devroey et al. 2017)	SPLC
[S110]	Titan: Test suite optimization for highly configurable software	(Marijan et al. 2017)	ICST
[S111]	A semi-automated iterative process for detecting feature interactions	(Rocha et al. 2020)	SBES
[S112]	Combinatorial interaction testing with multi-perspective feature models	(Patel et al. 2013)	ICSTW
[S113]	Delta-oriented model-based integration testing of large-scale systems	(Lochau et al. 2014)	J Syst Softw
[S114]	Functional Testing of Conflict Detection and Diagnosis Tools in Feature Model Configuration: A Test Suite Design	(Vidal Silva et al. 2020)	ConfWS
[S115]	Generic Solution-Space Sampling for Multi-domain Product Lines	(Hentze et al. 2022)	GPCE
[S116]	Predicting higher order structural feature interactions in variable systems	(Fischer et al. 2018)	ICSME
[S117]	Search-based diverse sampling from real-world software product lines	(Xiang et al. 2022)	ICSE
[S118]	Software Product Line Testing—A Proposal of Distance-Based Approach	(Saini et al. 2022)	AISE

Appendix D

Table 16 Evidence level of the studies

Study	Lev1	Lev2	Lev3	Lev4	Lev5	Lev6
S1		Yes				
S2		Yes				
S3		Yes				
S4				Yes		
S5				Yes		
S6				Yes		
S7				Yes		
S8				Yes		
S9				Yes		
S10		Yes				
S11				Yes		
S12				Yes		
S13				Yes		
S14				Yes		
S15				Yes		
S16				Yes		
S17				Yes	Yes	
S18						Yes
S19					Yes	
S20		Yes				
S21					Yes	
S22				Yes		
S23				Yes		
S24					Yes	
S25				Yes		
S26				Yes		
S27				Yes		
S28				Yes		
S29				Yes		
S30				Yes		
S31		Yes				
S32		Yes				
S33		Yes				
S34		Yes				
S35				Yes		
S36				Yes		
S37				Yes		
S38						Yes
S39				Yes		
S40				Yes		
S41				Yes		
S42				Yes		
S43				Yes		
S44				Yes		

Table 16 Evidence level of the studies

Study	Lev1	Lev2	Lev3	Lev4	Lev5	Lev6
S45				Yes		
S46				Yes		
S47				Yes		
S48				Yes		
S49		Yes				
S50				Yes		
S51				Yes		
S52				Yes		
S53				Yes		
S54				Yes		
S55				Yes		Yes
S56				Yes		
S57				Yes		
S58		Yes				
S59				Yes		
S60		Yes				
S61				Yes		Yes
S62				Yes		
S63				Yes		
S64				Yes		Yes
S65					Yes	
S66					Yes	
S67		Yes				
S68						Yes
S69					Yes	
S70						Yes
S71						Yes
S72		Yes				
S73				Yes		
S74					Yes	
S75				Yes		
S76				Yes		
S77					Yes	
S78				Yes		
S79		Yes				
S80				Yes		
S81		Yes				
S82				Yes		
S83		Yes				
S84						Yes
S85						Yes
S86						Yes
S87					Yes	
S88				Yes		
S89					Yes	
S90						Yes
S91				Yes		

Table 16 Evidence level of the studies

Study	Lev1	Lev2	Lev3	Lev4	Lev5	Lev6
S92					Yes	
S93				Yes		
S94				Yes		
S95				Yes		
S96				Yes		
S97					Yes	
S98		Yes			Yes	
S99				Yes		
S100		Yes				
S101				Yes		
S102						Yes
S103				Yes		
S104		Yes			Yes	
S105					Yes	
S106				Yes		
S107				Yes		
S108				Yes	Yes	
S109				Yes		
S110						Yes
S111				Yes		
S112						Yes
S113						Yes
S114				Yes		
S115					Yes	
S116				Yes		
S117				Yes		
S118				Yes		

Lev1: No evidence, Lev2: Toy examples, Lev3: Expert opinions, Lev4: Academic studies, Lev5: Industrial studies, Lev6: Industrial practice

Appendix E

Table 17 shows the results of the evaluation based on the quality assessment criteria, described in Table 14 in Appendix B. Regarding the issue Reporting (QA1-QA3 in Table 14), most of the studies performed well; all the studies are based on research and almost 82% of them have a clear statement of the aims of the research. However, the description of the context is bad in some of the studies (~ 30%); this compromises the validity of these studies since, without enough information about the subjects of the study, it is usually difficult to specify whether the selected case is suitable to evaluate different aspects of the proposed approach.

In terms of rigor (QA4-QA7), the studies performed, on average, fairly well. In 77 studies (~ 62%), the researchers have justified the research design so that it can address the aims of the research. In 71 studies (~ 60%), the proposed approach has been compared with a base approach; the researcher(s) has tried to justify that the selected controls are representative of a defined population. The way data collected is satisfactory in 85 studies (~ 72%) since the researchers have clearly defined the measure(s) selected and justified their selection. Furthermore, the data has been analyzed rigorously in 80 studies (68%) by providing sufficient data to support the findings. Although these findings are promising, 32% of the studies, overall, fail in rigor; this compromises the validity and usefulness of these studies since failing in rigor, as a key issue in Evidence-Based Software Engineering, indicates that the empirical methods have been applied in an informal way.

Regarding the issue Credibility, 95% of the studies provide a clear statement of the findings (QA9) by discussing the findings in relation to the research questions and also presenting the limitations of the study. However, most studies perform poorly in establishing relationships between the researcher(s) and participants and the data collected to address the research issue (QA8); this quality attribute is considered in only 12 studies (~10%). This can threaten the quality of the research due to not considering potential bias and influence of the researcher(s) during the formulation of research questions, data collection, and analysis and selection of data for presentation.

In terms of Relevance, 114 studies (~97%) explicitly deal with SPL testing and discuss the contributions the study makes to existing knowledge, identify new areas in which research is necessary, and discuss the ways in which the research can be used (QA10). This result is in line with the nature of the research goals, described as inclusion and exclusion criteria in Sect. 3.2. However, only 18 studies (~15%) present practitioner-based guidelines (QA11). This indicates that the SPL testing field needs more practical guidance to strengthen the adoption of industry.

Table 17 Evaluation of the studies based on the Quality Assessment (QA) criteria

Study	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9	QA10	QA11
S1	1	1	0	0	0	0	0	0	1	1	0
S2	1	1	1	1	0	1	1	0	1	1	0
S3	1	1	1	1	0	1	1	0	1	1	0
S4	1	1	1	1	1	1	1	0	1	1	0
S5	1	0	1	1	0	0	0	0	0	1	0
S6	1	1	1	0	0	1	0	0	1	1	0
S7	1	0	1	0	0	1	1	0	0	1	0
S8	1	1	1	1	0	1	1	0	1	1	0
S9	1	1	1	1	0	1	1	0	1	1	0
S10	1	1	1	1	0	1	1	1	1	1	0
S11	1	1	1	1	0	1	1	0	1	1	0
S12	1	1	1	1	1	1	1	0	1	1	0
S13	1	1	1	1	1	1	1	1	1	1	0
S14	1	1	1	1	0	1	1	0	1	1	0
S15	1	1	1	1	1	1	1	0	1	1	0
S16	1	0	1	0	1	0	1	0	1	1	0
S17	1	1	1	1	1	1	1	0	1	1	0
S18	1	1	1	1	0	1	1	0	1	1	0
S19	1	1	1	1	0	1	1	0	1	1	0
S20	1	1	0	0	0	1	1	0	1	1	0
S21	1	1	1	1	1	1	1	0	1	1	1
S22	1	1	0	0	0	0	1	0	1	1	0
S23	1	1	1	1	1	1	1	0	1	1	1
S24	1	0	1	1	0	1	1	0	1	1	0
S25	1	0	1	0	1	0	0	0	1	1	0

Table 17 Evaluation of the studies based on the Quality Assessment (QA) criteria

Study	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9	QA10	QA11
S26	1	1	0	1	0	1	1	0	1	1	0
S27	1	1	1	0	0	1	1	0	1	1	0
S28	1	1	1	1	1	1	1	0	1	1	0
S29	1	1	1	1	1	1	1	1	1	1	1
S30	1	1	0	1	1	1	1	0	1	1	0
S31	1	1	0	0	1	1	1	0	1	1	0
S32	1	0	1	0	0	0	0	0	1	1	0
S33	1	1	0	1	1	0	0	0	1	1	0
S34	1	1	0	0	1	0	0	0	0	1	1
S35	1	1	0	1	1	1	1	0	1	1	0
S36	1	1	1	1	1	0	0	0	1	1	0
S37	1	1	1	1	1	1	1	0	1	1	0
S38	1	1	0	1	1	1	1	0	1	1	0
S39	1	1	1	1	1	1	1	0	1	1	0
S40	1	1	1	1	1	1	1	0	1	1	0
S41	1	1	1	1	1	1	0	1	1	1	0
S42	1	1	1	1	1	1	1	0	1	1	0
S43	1	1	0	0	1	0	0	0	1	0	0
S44	1	0	1	0	1	0	0	0	1	0	0
S45	1	1	1	1	1	1	1	1	1	1	0
S46	1	1	1	1	1	1	1	0	1	1	0
S47	1	1	1	1	1	1	1	0	1	1	0
S48	1	1	0	1	1	1	1	0	1	1	0
S49	1	1	0	0	0	0	0	1	1	1	0
S50	1	1	1	1	1	1	1	0	1	1	0
S51	1	1	0	1	1	1	1	0	1	1	0
S52	1	1	1	1	1	1	1	0	1	1	0
S53	1	1	1	1	1	1	1	0	1	1	0
S54	1	0	1	0	1	0	0	0	0	1	0
S55	1	1	0	0	1	0	0	0	1	1	0
S56	1	0	1	1	1	0	0	0	1	1	0
S57	1	1	1	1	1	1	1	0	1	1	0
S58	1	1	0	0	1	0	0	0	1	1	0
S59	1	1	1	1	0	1	1	0	1	1	0
S60	1	1	0	0	1	0	0	0	1	1	0
S61	1	1	1	1	1	1	1	0	1	1	0
S62	1	1	1	1	1	1	1	0	1	1	0
S63	1	1	0	0	1	0	0	0	1	1	0
S64	1	1	1	1	1	1	1	1	1	1	1
S65	1	0	1	0	0	0	0	0	1	1	0
S66	1	1	1	1	0	1	1	1	1	1	0
S67	1	0	1	0	0	0	0	0	1	0	0
S68	1	0	1	0	0	1	1	0	1	1	0
S69	1	0	1	0	0	1	0	0	1	1	0
S70	1	0	1	0	0	1	0	0	1	1	0
S71	1	1	1	1	0	1	1	0	1	1	0
S72	1	1	0	0	0	0	0	0	1	1	1

Table 17 Evaluation of the studies based on the Quality Assessment (QA) criteria

Study	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9	QA10	QA11
S73	1	1	1	1	1	1	1	0	1	1	0
S74	1	0	1	0	0	1	1	0	1	1	0
S75	1	1	0	0	1	1	1	0	1	1	0
S76	1	0	1	0	1	0	0	0	1	1	0
S77	1	1	1	1	1	1	1	0	1	1	0
S78	1	1	1	1	1	1	1	1	1	1	0
S79	1	1	0	1	1	1	1	0	1	1	0
S80	1	1	0	0	0	0	0	0	1	1	0
S81	1	0	1	0	0	0	0	0	1	0	0
S82	1	1	0	0	0	0	0	0	1	1	0
S83	1	0	1	0	0	1	1	0	1	1	0
S84	1	1	0	0	0	1	0	0	0	1	0
S85	1	1	1	0	1	1	1	1	1	1	0
S86	1	1	0	1	1	0	0	0	1	1	0
S87	1	1	0	0	0	1	0	0	0	1	0
S88	1	1	1	1	1	1	1	0	1	1	1
S89	1	1	0	0	0	0	0	0	1	1	0
S90	1	1	0	0	0	1	1	0	1	1	0
S91	1	1	0	0	0	1	1	0	1	1	0
S92	1	1	1	1	0	1	1	0	1	1	0
S93	1	1	1	1	1	1	1	0	1	1	0
S94	1	1	1	1	1	1	1	0	1	1	0
S95	1	1	0	0	0	1	1	0	1	1	0
S96	1	0	1	0	0	0	0	0	1	1	0
S97	1	1	0	0	1	1	1	0	1	1	0
S98	1	1	0	1	1	1	1	0	1	1	0
S99	1	1	1	1	1	1	1	1	1	1	1
S100	1	1	1	1	1	1	1	0	1	1	0
S101	1	1	0	1	0	0	0	0	1	1	0
S102	1	1	1	1	0	1	1	1	1	1	1
S103	1	1	1	1	1	1	1	0	1	1	1
S104	1	1	1	1	1	1	1	0	1	1	1
S105	1	1	0	0	0	0	0	0	1	1	0
S106	1	1	0	1	1	1	1	0	1	1	0
S107	1	1	1	1	1	1	1	0	1	1	0
S108	1	1	1	1	1	1	1	0	1	1	1
S109	1	1	1	1	1	0	0	0	1	1	0
S110	1	0	1	1	1	0	0	0	1	1	0
S111	1	1	1	1	0	1	1	0	1	1	1
S112	1	1	1	1	1	1	1	0	1	1	0
S113	1	1	1	1	1	1	1	0	1	1	1
S114	1	0	1	1	0	1	0	0	1	1	1
S115	1	1	1	1	1	1	1	0	1	1	0
S116	1	1	1	1	1	1	1	0	1	1	1
S117	1	1	1	1	1	1	1	0	1	1	1
S118	1	1	1	1	1	0	0	0	1	1	0

Funding This work was funded by the Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg in the Innovation Campus Mobility of the Future, projects SWUpCar and TESSOF.
Open Access funding enabled and organized by Projekt DEAL.

Data availability All data generated during this study are available in the "Zenodo" repository: https://zenodo.org/doi/10.5281/zenodo.10018266.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Aduni Sulaiman R, Jawawi DN, Halim SA (2019) Derivation of test cases for model-based testing of software product line with hybrid heuristic approach. In: IRICT'19, pp 199–208. https://doi.org/10.1007/978-3-030-33582-3_19

Akbari Z, Khoshnevis S, Mohsenzadeh M (2017) A method for prioritizing integration testing in software product lines based on feature model. Int J Softw Eng Knowl Eng 27(04):575–600. https://doi.org/10.1142/S0218194017500218

Akimoto H, Isogami Y, Kitamura T, Noda N, Kishi T (2019) A prioritization method for SPL pairwise testing based on user profiles. In: APSEC'19, pp 118–125. https://doi.org/10.1109/APSEC48747.2019.00025

Al-Dallal J, Sorenson PG (2008) Testing software assets of framework-based product families during application engineering stage. J Softw 3(5):11–25

Al-Hajjaji M, Krieter S, Thüm T, Lochau M, Saake G (2016) IncLing: efficient product-line testing using incremental pairwise sampling. ACM SIGPLAN Not 52(3):144–155. https://doi.org/10.1145/3093335.2993253

Al-Hajjaji M, Krüger J, Schulze S, Leich T, Saake G (2017a) Efficient product-line testing using cluster-based product prioritization. In: AST'17, pp 16–22. https://doi.org/10.1109/AST.2017.7

Al-Hajjaji M, Lity S, Lachmann R, Thüm T, Schaefer I, Saake G (2017b) Delta-oriented product prioritization for similarity-based product-line testing. In: VACE'17, pp 34–40. https://doi.org/10.1109/VACE.2017.8

Al-Hajjaji M, Thüm T, Lochau M, Meinicke J, Saake G (2019) Effective product-line testing using similarity-based product prioritization. Softw Syst Model 18(1):499–521. https://doi.org/10.1007/s10270-016-0569-2

AL-Msie'deen RF, Seriai A, Huchard M, Urtado C, Vauttier S, Salman HE (2013) Feature location in a collection of software product variants using formal concept analysis. In: ICSR'13, pp 302–307. https://doi.org/10.1007/978-3-642-38977-1_22

Alves V, Niu N, Alves C, Valença G (2010) Requirements engineering for software product lines: a systematic literature review. Inf Softw Technol 52(8):806–820

Alves Pereira J, Acher M, Martin H, Jézéquel JM (2020) Sampling effect on performance prediction of configurable systems: A case study. In: ICPE'20, pp 277–288. https://doi.org/10.1145/3358960.3379137

Ammann P, Offutt J (2008) Introduction to software testing. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511809163

Ampatzoglou A, Bibi S, Avgeriou P, Verbeek M, Chatzigeorgiou A (2019) Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Inf Softw Technol 106:201–230

Aoyama Y, Kuroiwa T, Kushiro N (2021) Executable test case generation from specifications written in natural language and test execution environment. In: CCNC'21, pp 1–6.https://doi.org/10.1109/CCNC49032.2021.9369549

Apel S, Batory D, Kästner C, Saake G (2013) Feature-oriented software product lines: concepts and implementation. Springer, Berlin

Araújo IL, Santos IS, Filho JB, Andrade RM, Neto PS (2017) Generating test cases and procedures from use cases in dynamic software product lines. In: SAC'17, pp 1296–1301. https://doi.org/10.1145/3019612.3019790

Arrieta A, Sagardui G, Etxeberria L (2015) Test control algorithms for the validation of cyber-physical systems product lines. In: SPLC'15, pp 273–282. https://doi.org/10.1145/2791060.2791095

Arrieta A, Wang S, Sagardui G, Etxeberria L (2016) Search-based test case selection of cyber-physical system product lines for simulation-based validation. In: SPLC'16, pp 297–306. https://doi.org/10.1145/2934466.2946046

Arrieta A, Wang S, Sagardui G, Etxeberria L (2019) Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. J Syst Softw 149:1–34. https://doi.org/10.1016/j.jss.2018.09.055

Baller H, Lity S, Lochau M, Schaefer I (2014) Multi-objective test suite optimization for incremental product family testing. In: ICST'14, pp 303–312. https://doi.org/10.1109/ICST.2014.43

Belli F, Tuglular T, Ufuktepe E (2021) Heterogeneous modeling and testing of software product lines. In: QRS-C'21, pp 1079–1088. https://doi.org/10.1109/QRS-C55045.2021.00162

Beohar H, Mousavi MR (2016) Input–output conformance testing for software product lines. J Log Algebr Methods Program 85(6):1131–1153. https://doi.org/10.1016/j.jlamp.2016.09.007

Bharathi M, Sangeetha V (2018) Weighted rank ant colony metaheuristics optimization-based test suite reduction in combinatorial testing for improving software quality. In: ICICCS'18, pp 525–534. https://doi.org/10.1109/ICCONS.2018.8663102

Bucaioni A, Di Silvestro F, Singh I, Saadatmand M, Muccini H (2022) Model-based generation of test scripts across product variants: an experience report from the railway industry. J Softw Evol Process 34(11):e2498. https://doi.org/10.1002/smr.2498

Bürdek J, Lochau M, Bauregger S, Holzer A, Rhein AV, Apel S, Beyer D (2015) Facilitating reuse in multi-goal test-suite generation for software product lines. In: FASE'15, pp 84–99. https://doi.org/10.1007/978-3-662-46675-9_6

Cabral I, Cohen MB, Rothermel G (2010) Improving the testing and testability of software product lines. In: SPLC'10, pp 241–255. https://doi.org/10.1007/978-3-642-15579-6_17

Calvagna A, Gargantini A, Vavassori P (2013) Combinatorial testing for feature models using CitLab. In: ICSTW'13, pp 338–347. https://doi.org/10.1109/ICSTW.2013.45

Carlsson M, Gotlieb A, Marijan D (2016) Software product line test suite reduction with constraint optimization. In: ICSOFT'16, pp 68–87. https://doi.org/10.1007/978-3-319-62569-0_4

Catal C, Mishra D (2012) Test case prioritization: a systematic mapping study. Softw Qual J 21(3):445–478. https://doi.org/10.1007/s11219-012-9181-z

Chen L, Babar MA (2011) A systematic review of evaluation of variability management approaches in software product lines. Inf Softw Technol 53(4):344–362. https://doi.org/10.1016/j.infsof.2010.12.006

Clarke D, Helvensteijn M, Schaefer I (2010) Abstract delta modeling. ACM SIGPLAN Not 46(2):13–22. https://doi.org/10.1145/1942788.1868298

Clements P, Northrop L (2002) Software product lines: practices and patterns. Addison-Wesley, Boston

Cmyrev A, Reissing R (2014) Efficient and effective testing of automotive software product lines. Appl Sci Eng Prog 7(2):53–57. https://doi.org/10.14416/j.ijast.2014.05.001

Cohen MB, Dwyer MB, Shi J (2006) Coverage and adequacy in software product line testing. In: ROSATEA'06, pp 53–63. https://doi.org/10.1145/1147249.1147257

Cohen MB, Dwyer MB, Shi J (2007) Interaction testing of highly-configurable systems in the presence of constraints. In: ISSTA'07, pp 129–139. https://doi.org/10.1145/1273463.1273482

Cruzes DS, Dybä T (2011) Research synthesis in software engineering: a tertiary study. Inf Softw Technol 53(5):440–455. https://doi.org/10.1016/j.infsof.2011.01.004

Czarnecki K, Eisenecker UW (2000) Generative programming: methods, tools and applications. Addison-Wesley, New York

Da Mota Silveira Neto PA, do, Carmo Machado I, McGregor JD, De Almeida ES, de Lemos Meira SR (2011) A systematic mapping study of software product lines testing. Inf Softw Technol 53(5):407–423. https://doi.org/10.1016/j.infsof.2010.12.003

Denger C, Kolb R (2006) Testing and inspecting reusable product line components: First empirical results. In: ISESE'06, pp 184–193. https://doi.org/10.1145/1159733.1159762

Devroey X, Perrouin G, Legay A, Schobbens PY, Heymans P (2017) Dissimilar test case selection for behavioural software product line testing, In: SPLC'17, pp 1–9

do Carmo Machado I, McGregor JD, Cavalcanti YC, De Almeida ES (2014) On strategies for testing software product lines: a systematic literature review. Inf Softw Technol 56(10):1183–1199. https://doi.org/10.1016/j.infsof.2014.04.002

do Nascimento Ferreira T, Kuk JN, Pozo A, Vergilio SR (2016) Product selection based on upper confidence bound MOEA/D-DRA for testing software product lines. In: CEC’16, pp 4135–4142. https://doi.org/10.1109/CEC.2016.7744315

Dominka S, Mandl M, Dübner M, Ertl D (2018) Using combinatorial testing for distributed automotive features: Applying combinatorial testing for automated feature-interaction-testing. In: CCWC’18, pp 490–495. https://doi.org/10.1109/CCWC.2018.8301632

Drave I, Hillemacher S, Greifenberg T, Kriebel S, Kusmenko E, Markthaler M, Orth P, Salman KS, Richenhagen J, Rumpe B, Schulze C (2019) SMArDT modeling for automotive software testing. Softw Pract Exp 49(2):301–328. https://doi.org/10.1002/spe.2650

Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: a systematic review. Inf Softw Technol 50(9–10):833–859. https://doi.org/10.1016/j.infsof.2008.01.006

Ebert R, Jolianis J, Kriebel S, Markthaler M, Pruenster B, Rumpe B, Salman KS (2019) Applying product line testing for the electric drive system. In: SPLC’19, pp 14–24. https://doi.org/10.1145/3336294.3336318

Engström E, Runeson P (2011) Software product line testing–A systematic mapping study. Inf Softw Technol 53(1):2–13. https://doi.org/10.1016/j.infsof.2010.05.011

Ensan F, Bagheri E, Gašević D (2012) Evolutionary search-based test generation for software product line feature models. In: CAiSE’12, pp 613–628. https://doi.org/10.1007/978-3-642-31095-9_40

Ergun B, Gebizli CŞ, Sözer H (2017) FORMAT: A tool for adapting test models based on feature models. In: COMPSAC’17, pp 66–71. https://doi.org/10.1109/COMPSAC.2017.134

Ferreira JM, Vergilio SR, Quináia MA (2013) A mutation approach to feature testing of software product lines. In: SEKE’13, pp 231–237

Ferreira JM, Vergilio SR, Quinaia MA (2017a) Software product line testing based on feature model mutation. Int J Softw Eng Knowl Eng 27(05):817–839. https://doi.org/10.1142/S0218194017500309

Ferreira TN, Vergilio SR, de Souza JT (2017b) Incorporating user preferences in search-based software engineering: a systematic mapping study. Inf Softw Technol 90:55–69. https://doi.org/10.1016/j.infsof.2017.05.003

Ferrer J, Chicano F, Alba E (2017) Hybrid algorithms based on integer programming for the search of prioritized test data in software product lines. In: EvoCOP’17, pp 3–19. https://doi.org/10.1007/978-3-319-55792-2_1

Ferrer J, Chicano F, Ortega-Toro JA (2021) CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. J Heuristics 27(1):229–249. https://doi.org/10.1007/s10732-020-09462-w

Fischer S, Linsbauer L, Egyed A, Lopez-Herrejon RE (2018) Predicting higher order structural feature interactions in variable systems. In: ICSME’18, pp 252–263. https://doi.org/10.1109/ICSME.2018.00035

Fomel S (2002) Object management group: UML profile for schedulability, performance and time specification. OMG Doc 2(03):1–101

Fragal VH, Simao A, Endo AT, Mousavi MR (2017) Reducing the concretization effort in FSM-based testing of software product lines. In: ICSTW’17, pp 329–336. https://doi.org/10.1109/ICSTW.2017.61

Fragal VH, Simao A, Mousavi MR, Turker UC (2019) Extending HSI test generation method for software product lines. Comput J 62(1):109–129. https://doi.org/10.1093/comjnl/bxy046

Galindo JA, Alférez M, Acher M, Baudry B, Benavides D (2014) A variability-based testing approach for synthesizing video sequences. In: ISSTA’14, pp 293–303. https://doi.org/10.1145/2610384.2610411

Galindo JA, Turner H, Benavides D, White J (2016) Testing variability-intensive systems using automated analysis: an application to Android. Softw Qual J 24(2):365–405. https://doi.org/10.1007/s11219-014-9258-y

Gebizli CS, Sözer H (2016) Model-based software product line testing by coupling feature models with hierarchical markov chain usage models. In: QRS-C’16, pp 278–283. https://doi.org/10.1109/QRS-C.2016.42

Ghanam Y, Andreychuk D, Maurer F (2010) Reactive variability management in agile software development. In: 2010 Agile Conference, pp 27–34. https://doi.org/10.1109/AGILE.2010.6

Hajri I, Goknil A, Pastore F, Briand LC (2020) Automating system test case classification and prioritization for use case-driven testing in product lines. Empir Softw Eng 25(5):3711–3769. https://doi.org/10.1007/s10664-020-09853-4

Henard C, Papadakis M, Perrouin G, Klein J, Traon YL (2013) Multi-objective test generation for software product lines. In: SPLC’13, pp 62–71. https://doi.org/10.1145/2491627.2491635

Henard C, Papadakis M, Perrouin G, Klein J, Heymans P, Le Traon Y (2014a) Bypassing the combinatorial explosion: using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans Softw Eng 40(7):650–670. https://doi.org/10.1109/TSE.2014.2327020

Henard C, Papadakis M, Traon YL (2014b) Mutation-based generation of software product line test configurations. In: SSBSE’14, pp 92–106. https://doi.org/10.1007/978-3-319-09940-8_7

Hentze M, Pett T, Sundermann C, Krieter S, Thüm T, Schaefer I (2022) Generic Solution-Space Sampling for Multi-domain Product Lines. In: GPCE’22, pp 135–147. https://doi.org/10.1145/3564719.3568695

Hervieu A, Baudry B, Gotlieb A (2011) PACOGEN: Automatic generation of pairwise test configurations from feature models. In: ISSRE'11, pp 120–129. https://doi.org/10.1109/ISSRE.2011.31

Hervieu A, Marijan D, Gotlieb A, Baudry B (2016) Practical minimization of pairwise-covering test configurations using constraint programming. Inf Softw Technol 71:129–146. https://doi.org/10.1016/j.infsof.2015.11.007

Hierons RM, Li M, Liu X, Parejo JA, Segura S, Yao X (2020) Many-objective test suite generation for software product lines. ACM Trans Softw Eng Methodol 29(1):1–46. https://doi.org/10.1145/3361146

Ibias A, Llana L, Núñez M (2022) Using ant colony optimisation to select features having associated costs. In: ICTSS'22, pp 106–122. https://doi.org/10.1007/978-3-031-04673-5_8

Jackson D (2012) Software abstractions: logic, language, and analysis. MIT Press

Jakubovski Filho HL, Ferreira TN, Vergilio SR (2018) Incorporating user preferences in a software product line testing hyper-heuristic approach. In: CEC'18, pp 1–8. https://doi.org/10.1109/CEC.2018.8477803

Jakubovski Filho HL, Ferreira TN, Vergilio SR (2019) Preference based multi-objective algorithms applied to the variability testing of software product lines. J Syst Softw 151:194–209. https://doi.org/10.1016/j.jss.2019.02.028

Jaring M, Krikhaar RL, Bosch J (2008) Modeling variability and testability interaction in software product line engineering. In: ICCBSS'08, pp 120–129. https://doi.org/10.1109/ICCBSS.2008.9

Johansen MF, Haugen Ø, Fleurey F (2011) Properties of realistic feature models make combinatorial testing of product lines feasible. In: MODELS'11, pp 638–652. https://doi.org/10.1007/978-3-642-24485-8_47

Jorgensen PC (2013) Software testing: a craftsman's approach. Auerbach Publications

Jung P, Kang S, Lee J (2019) Automated code-based test selection for software product line regression testing. J Syst Softw 158:110419. https://doi.org/10.1016/j.jss.2019.110419

Jung P, Kang S, Lee J (2020) Efficient regression testing of software product lines by reducing redundant test executions. Appl Sci 10(23):8686. https://doi.org/10.3390/app10238686

Jung P, Kang S, Lee J (2022) Reducing redundant test executions in software product line testing—A case study. Electronics 11(7):1165. https://doi.org/10.3390/electronics11071165

Käkölä T, Dueñas JC (2006) Research issues in software product lines—Engineering and management. Springer, Heidelberg

Kang S, Baek H, Kim J, Lee J (2015) Systematic software product line test case derivation for test data reuse. In: COMPSAC'15, pp 433–440. https://doi.org/10.1109/COMPSAC.2015.174

Kim CH, Batory DS, Khurshid S (2011) Reducing combinatorics in testing product lines. In: AOSD'11, pp 57–68. https://doi.org/10.1145/1960275.1960284

Kim CH, Khurshid S, Batory D (2012) Shared execution for efficiently testing product lines. In: ISSRE'12, pp 221–230. https://doi.org/10.1109/ISSRE.2012.23

Kim CH, Marinov D, Khurshid S, Batory D, Souto S, Barros P, d'Amorim M (2013) SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems. In: ESEC/FSE'13, pp 257–267. https://doi.org/10.1145/2491411.2491459

Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Technical Report, Keele University and Durham University

Kitchenham B, Budgen D, Brereton P (2016) Evidence-based software engineering and systematic reviews. CRC Press

Krieter S, Thüm T, Schulze S, Saake G, Leich T (2020) YASA: Yet another sampling algorithm. In: VaMoS'20, pp 1–10. https://doi.org/10.1145/3377024.3377042

Kuhn DR, Wallace DR, Gallo AM (2004) Software fault interactions and implications for software testing. IEEE Trans Softw Eng 30(6):418–421. https://doi.org/10.1109/TSE.2004.24

Kumar S (2016) Test case prioritization techniques for software product line: A survey. In: ICCCA, pp 884–889. https://doi.org/10.1109/CCAA.2016.7813841

Lachmann R, Lity S, Lischke S, Beddig S, Schulze S, Schaefer I (2015) Delta-oriented test case prioritization for integration testing of software product lines. In: SPLC'15, pp 81–90. https://doi.org/10.1145/2791060.2791073

Lachmann R, Lity S, Al-Hajjaji M, Fürchtegott F, Schaefer I (2016) Fine-grained test case prioritization for integration testing of delta-oriented software product lines. In: FOSD'16, pp 1–10. https://doi.org/10.1145/3001867.3001868

Lachmann R, Beddig S, Lity S, Schulze S, Schaefer I (2017) Risk-based integration testing of software product lines. In: VaMoS'17, pp 52–59. https://doi.org/10.1145/3023956

Lackner H, Thomas M, Wartenberg F, Weißleder S (2014) Model-based test design of product lines: Raising test design to the product line level. In: ICST'14, pp 51–60. https://doi.org/10.1109/ICST.2014.16

Lamancha BP, Polo M, Piattini M (2015) PROW: a pairwise algorithm with constraints, order and weight. J Syst Softw 99:1–19. https://doi.org/10.1016/j.jss.2014.08.005

Lee J, Hwang S (2019) Combinatorial test design using design-time decisions for variability. Int J Softw Eng Knowl Eng 29(08):1141–1158. https://doi.org/10.1142/S0218194019400138

Li Z, Harman M, Hierons RM (2007) Search algorithms for regression test case prioritization. IEEE Trans Softw Eng 33(4):225–237. https://doi.org/10.1109/TSE.2007.38

Li X, Wong WE, Gao R, Hu L, Hosono S (2018) Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. Empir Softw Eng 23(1):1–51. https://doi.org/10.1007/s10664-016-9494-9

Lima JA, Mendonça WD, Vergilio SR, Assunção WK (2020) Learning-based prioritization of test cases in continuous integration of highly-configurable software. In: SPLC'20, pp 1–11. https://doi.org/10.1145/3382025.3414967

Lity S, Nieke M, Thüm T, Schaefer I (2019) Retest test selection for product-line regression testing of variants and versions of variants. J Syst Softw 147:46–63. https://doi.org/10.1016/j.jss.2018.09.090

Lochau M, Oster S, Goltz U, Schürr A (2012a) Model-based pairwise testing for feature interaction coverage in software product line engineering. Softw Qual J 20(3):567–604. https://doi.org/10.1007/s11219-011-9165-4

Lochau M, Schaefer I, Kamischke J, Lity S (2012b) Incremental model-based testing of delta-oriented software product lines. In: TAP'12, pp 67–82. https://doi.org/10.1007/978-3-642-30473-6_7

Lochau M, Lity S, Lachmann R, Schaefer I, Goltz U (2014) Delta-oriented model-based integration testing of large-scale systems. J Syst Softw 91:63–84. https://doi.org/10.1016/j.jss.2013.11.1096

Lopez-Herrejon RE, Javier Ferrer J, Chicano F, Haslinger EN, Egyed A, Alba E (2014) A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In: GECCO'14, pp 1255–1262. https://doi.org/10.1145/2576768.2598305

Lopez-Herrejon RE, Fischer S, Ramler R, Egyed A (2015) A first systematic mapping study on combinatorial interaction testing for software product lines. In: ICSTW'15, pp 1–10. https://doi.org/10.1109/ICSTW.2015.7107435

Luo L (2001) Software testing techniques. Institute for Software Research International Carnegie Mellon University. Pittsburgh PA 15232(19):1–19

Luo G, Petrenko A, Bochmann GV (1995) Selecting test sequences for partially-specified nondeterministic finite state machines. In: IFIP WG, pp 95–110. https://doi.org/10.1007/978-0-387-34883-4_6

Luthmann L, Gerecht T, Stephan A, Bürdek J, Lochau M (2019a) Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints. J Syst Softw 149:535–553. https://doi.org/10.1016/j.jss.2018.12.028

Luthmann L, Gerecht T, Lochau M (2019b) Sampling strategies for product lines with unbounded parametric real-time constraints. Int J Softw Tools Technol Transf 21(6):613–633. https://doi.org/10.1007/s10009-019-00532-4

Marijan D, Gotlieb A, Sen S, Hervieu A (2013) Practical pairwise testing for software product lines. In: SPLC'13, pp 227–235. https://doi.org/10.1145/2491627.2491646

Marijan D, Liaaen M, Gotlieb A, Sen S, Ieva C (2017) Titan: Test suite optimization for highly configurable software. In: ICST'17, pp 524–531. https://doi.org/10.1109/ICST.2017.60

Markiegi U, Arrieta A, Sagardui G, Etxeberria L (2017) Search-based product line fault detection allocating test cases iteratively. In: SPLC'17, pp 123–132. https://doi.org/10.1145/3106195.3106210

Markiegi U, Arrieta A, Etxeberria L, Sagardui G (2019) Test case selection using structural coverage in software product lines for time-budget constrained scenarios. In: SAC'19, pp 2362–2371. https://doi.org/10.1145/3297280.3297512

Matnei Filho RA, Vergilio SR (2016) A multi-objective test data generation approach for mutation testing of feature models. J Softw Eng Res Dev 4(1):1–29. https://doi.org/10.1186/s40411-016-0030-9

Mayring P (2014) Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution. Klagenfurt. Available at Social Science Open Access Repository (SSOAR) https://nbn-resolving.de/urn:nbn:de:0168-ssoar-395173 (accessed 04 June 2024)

McGregor JD (2001) Testing a software product line. Technical Report, Carnegie Mellon University

Mendes E, Wohlin C, Felizardo K, Kalinowski M (2020) When to update systematic literature reviews in software engineering. J Syst Softw 167:110607. https://doi.org/10.1016/j.jss.2020.110607

Mishra S (2006) Specification based software product line testing: A case study. In: CS&P'06, pp 243–254

Nebut C, Pickin S, Le Traon Y, Jézéquel JM (2003) Automated requirements-based generation of test cases for product families. In: ASE'03, pp 263–266. https://doi.org/10.1109/ASE.2003.1240317

Nebut C, Traon YL, Jézéquel JM (2006) System testing of product lines: from requirements to test cases. In: Käköla T, Duenas JC (eds) Software Product lines. Springer, Berlin, Heidelberg, pp 447–477. https://doi.org/10.1007/978-3-540-33253-4_12

Neto PA, do Carmo Machado I, Cavalcanti YC, De Almeida ES, Garcia VC, de Lemos Meira SR (2010) A regression testing approach for software product lines architectures. In: SBCARS'10, pp 41–50. https://doi.org/10.1109/SBCARS.2010.14

Nguyen QL (2009) Non-functional requirements analysis modeling for software product lines. In: ICSE'09, pp 56–61. https://doi.org/10.1109/MISE.2009.5069898

Northrop L, Clements P, Bachmann F, Bergey J, Chastek G, Cohen S, Donohoe P, Jones L, Krut R, Little R (2007) A framework for software product line practice, version 5.0. Technical report, Carnegie Mellon University

Olimpiew EM, Gomaa H (2009) Reusable model-based testing. In: ICSR'09, pp 76–85. https://doi.org/10.1007/978-3-642-04211-9_8

Oster S, Markert F, Ritter P (2010) Automated incremental pairwise testing of software product lines. In: SPLC'10, pp 196–210. https://doi.org/10.1007/978-3-642-15579-6_14

Parejo JA, Sánchez AB, Segura S, Ruiz-Cortés A, Lopez-Herrejon RE, Egyed A (2016) Multi-objective test case prioritization in highly configurable systems: a case study. J Syst Softw 122:287–310. https://doi.org/10.1016/j.jss.2016.09.045

Parra C, Giral L, Infante A, Cortés C (2012) Extractive SPL adoption using multi-level variability modeling. In: SPLC'12, pp 99–106. https://doi.org/10.1145/2364412.2364429

Patel S, Gupta P, Shah V (2013) Combinatorial interaction testing with multi-perspective feature models. In: ICSTW'13, pp 321–330. https://doi.org/10.1109/ICSTW.2013.43

Pérez B, Polo M, Piatini M (2009) Software product line testing-A systematic review. In: ICSOFT'09, pp 1–8

Perrouin G, Sen S, Klein J, Baudry B, Le Traon Y (2010) Automated and scalable t-wise test case generation strategies for software product lines. In: ICST'10, pp 459–468. https://doi.org/10.1109/ICST.2010.43

Petry KL, OliveiraJr E, Zorzo AF (2020) Model-based testing of software product lines: mapping study and research roadmap. J Syst Softw 167:110608. https://doi.org/10.1016/j.jss.2020.110608

Pett T, Eichhorn D, Schaefer I (2020) Risk-based compatibility analysis in automotive systems engineering. In: MODELS'20, pp 1–10. https://doi.org/10.1145/3417990.3421263

Pohl K, Metzger A (2006) Software product line testing. Commun ACM 49(12):78–81. https://doi.org/10.1145/1183236.1183271

Pohl K, Böckle G, Van Der Linden F (2005) Software product line engineering: foundations, principles, and techniques. Springer Berlin, Heidelberg. https://doi.org/10.1007/3-540-28901-1

Qian Y, Zhang C, Wang F (2018) Selecting products for high-strength t-wise testing of software product line by multi-objective method. In: PIC'18, pp 370–378. https://doi.org/10.1109/PIC.2018.8706270

Reis S, Metzger A, Pohl K (2006) A reuse technique for performance testing of software product lines. In: SPLiT'06, pp 5–10

Reis S, Metzger A, Pohl K (2007) Integration testing in software product line engineering: a model-based technique. In: FASE'07, pp 321–335. https://doi.org/10.1007/978-3-540-71289-3_25

Reuling D, Bürdek J, Rotärmel S, Lochau M, Kelter U (2015) Fault-based product-line testing: Effective sample generation based on feature-diagram mutation. In: SPLC'15, pp 131–140. https://doi.org/10.1145/2791060.2791074

Reuys A, Kamsties E, Pohl K, Reis S (2005) Model-based system testing of software product families. In: CAiSE'05, pp 519–534. https://doi.org/10.1007/11431855_36

Reuys A, Reis S, Kamsties E, Pohl K (2006) The scented method for testing software product lines. In: SPLC'06, pp 479–520. https://doi.org/10.1007/978-3-540-33253-4_13

Rocha L, Machado I, Almeida E, Kästner C, Nadi S (2020) A semi-automated iterative process for detecting feature interactions. In: SBES'20, pp 778–787. https://doi.org/10.1145/3422392.3422418

Roggenbach M (2006) CSP-CASL—A new integration of process algebra and algebraic specification. Theor Comput Sci 354(1):42–71. https://doi.org/10.1016/j.tcs.2005.11.007

Rubin J, Chechik M (2013) A survey of feature location techniques. In: Reinhartz-Berger I, Sturm A, Clark T, Cohen S, Bettin J (eds) Domain Engineering. Springer, Berlin, Heidelberg, pp 29–58. https://doi.org/10.1007/978-3-642-36654-3_2

Safdar SA, Yue T, Ali S (2021) Recommending faulty configurations for interacting systems under test using multi-objective search. ACM Trans Softw Eng Methodol 30(4):1–36. https://doi.org/10.1145/3464939

Saini A, Rajkumar, Kumar S (2022) Software Product Line Testing—A Proposal of Distance-Based Approach. In: AISE'20, pp 187–198. https://doi.org/10.1007/978-981-16-8542-2_15

Sánchez AB, Segura S, Ruiz-Cortés A (2014) A comparison of test case prioritization criteria for software product lines. In: ICST'14, pp 41–50. https://doi.org/10.1109/ICST.2014.15

Schaefer I, Bettini L, Bono V, Damiani F, Tanzarella N (2010) Delta-oriented programming of software product lines. In: SPLC'10, pp 77–91. https://doi.org/10.1007/978-3-642-15579-6_6

Scheidemann KD (2006) Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems. In: SPLC'06, pp 75–84. https://doi.org/10.1109/SPLINE.2006.1691579

Shi J, Cohen MB, Dwyer MB (2012) Integration testing of software product lines using compositional symbolic execution. In: FASE'12, pp 270–284. https://doi.org/10.1007/978-3-642-28872-2_19

Sjoberg DI, Dyba T, Jorgensen M (2007) The future of empirical methods in software engineering research. In: FOSE'07, pp 358–378. https://doi.org/10.1109/FOSE.2007.30

Soe NT, Wild N, Tanachutiwat S, Lichter H (2022) Design and implementation of a test automation framework for configurable devices. In: APIT'22, pp 200–207. https://doi.org/10.1145/3512353.3512383
Souto S, d'Amorim M (2018) Time-space efficient regression testing for configurable systems. J Syst Softw 137:733–746. https://doi.org/10.1016/j.jss.2017.08.010
Souto S, d'Amorim M, Gheyi R (2017) Balancing soundness and efficiency for practical testing of configurable systems. In: ICSE'17, pp 632–642. https://doi.org/10.1109/ICSE.2017.64
Steffens M, Oster S, Lochau M, Fogdal T (2012) Industrial evaluation of pairwise SPL testing with MoSoPoLiTe. In: VaMoS'12, pp 55–62. https://doi.org/10.1145/2110147.2110154
Stricker V, Metzger A, Pohl K (2010) Avoiding redundant testing in application engineering. In: SPLC'10, pp 226–240. https://doi.org/10.1007/978-3-642-15579-6_16
Strickler A, Lima JA, Vergilio SR, Pozo AT (2016) Deriving products for variability test of feature models with a hyper-heuristic approach. Appl Soft Comput 49:1232–1242. https://doi.org/10.1016/j.asoc.2016.07.059
Tevanlinna A, Taina J, Kauppinen R (2004) Product family testing: a survey. ACM SIGSOFT Softw Eng Notes 29(2):12–12. https://doi.org/10.1145/979743.979766
Tuglular T, Coşkun DE (2021) Behavior-driven development of software product lines. In: DSA'21, pp 230–239. https://doi.org/10.1109/DSA52907.2021.00035
Tuglular T, Beyazıt M, Öztürk D (2019) Featured event sequence graphs for model-based incremental testing of software product lines. In: COMPSAC'19, pp 197–202. https://doi.org/10.1109/COMPSAC.2019.00035
Uzuncaova E, Khurshid S, Batory D (2010) Incremental test generation for software product lines. IEEE Trans Softw Eng 36(3):309–322. https://doi.org/10.1109/TSE.2010.30
Varshosaz M, Al-Hajjaji M, Thüm T, Runge T, Mousavi MR, Schaefer I (2018) A classification of product sampling for software product lines. In: SPLC'18, pp 1–13. https://doi.org/10.1145/3233027.3233035
Vazirani VV (2001) Approximation algorithms. Springer, Berlin. https://doi.org/10.1007/978-3-662-04565-7
Vidács L, Horváth F, Mihalicza J, Vancsics B, Beszédes Á (2015) Supporting software product line testing by optimizing code configuration coverage. In: ICSTW'15, pp 1–7. https://doi.org/10.1109/ICSTW.2015.7107478
Vidal Silva C, Galindo Duarte JÁ, Benavides Cuevas DF (2020) Functional testing of conflict detection and diagnosis tools in feature model configuration: a test suite design. In: ConfWS'20, pp 17–24
Wang S, Buchmann D, Ali S, Gotlieb A, Pradhan D, Liaaen M (2014) Multi-objective test prioritization in software product line testing: an industrial case study. In: SPLC'14, pp 32–41. https://doi.org/10.1145/2648511.2648515
Wang S, Ali S, Gotlieb A (2015) Cost-effective test suite minimization in product lines using search techniques. J Syst Softw 103:370–391. https://doi.org/10.1016/j.jss.2014.08.024
Wang S, Ali S, Gotlieb A, Liaaen M (2017) Automated product line test case selection: industrial case study and controlled experiment. Softw Syst Model 16(2):417–441. https://doi.org/10.1007/s10270-015-0462-4
Webster J, Watson RT (2002) Analyzing the past to prepare for the future: writing a literature review. MIS Q 26(2):xiii–xxiii
Weiss DM (2008) The product line hall of fame. In: SPLC'08, pp 39. https://doi.org/10.1109/SPLC.2008.56
Wohlin C, Höst M, Henningsson K (2003) Empirical research methods in software engineering. Empirical methods and studies in Software Engineering-experiences. Springer, Berlin, Heidelberg, pp 7–23. https://doi.org/10.1007/978-3-540-45143-3_2
Xiang Y, Huang H, Zhou Y, Li S, Luo C, Lin Q, Yang X (2022) Search-based diverse sampling from real-world software product lines. In: ICSE'22, pp 1945–1957. https://doi.org/10.1145/3510003.3510053
Xu Z, Cohen MB, Motycka W, Rothermel G (2013) Continuous test suite augmentation in software product lines. In: SPLC'13, pp 52–61. https://doi.org/10.1145/2491627.2491650
Yan L, Hu W, Han L (2019) Optimize SPL test cases with adaptive simulated annealing genetic algorithm. In: ACM TURC'19, pp 1–7. https://doi.org/10.1145/3321408.3326676
Yu L, Duan F, Lei Y, Kacker RN, Kuhn DR (2014) Combinatorial test generation for software product lines using minimum invalid tuples. In: HASE'14, pp 65–72. https://doi.org/10.1109/HASE.2014.18
Zhang L, Tian JH, Jiang J, Liu YJ, Pu MY, Yue T (2018) Empirical research in software engineering-A literature survey. JCST 33(5):876–899. https://doi.org/10.1007/s11390-018-1864-x