applied sciences

MDPI

Article

Reviewing Automated Analysis of Feature Model Solutions for the Product Configuration

Cristian Vidal-Silva [1,*], Vannessa Duarte [2], Jesennia Cardenas-Cobo [3], Jorge Serrano-Malebran [4], Iván Veas [4] and José Rubio-León [5]

[1] School of Videogame Development and Virtual Reality Engineering, Faculty of Engineering, University of Talca, Campus Talca, Talca 3460000, Chile
[2] Escuela de Ciencias Empresariales, Universidad Católica del Norte, Coquimbo 1781421, Chile
[3] Facultad de Ciencias e Ingenierías, Universidad Estatal de Milagro, Milagro 091706, Ecuador
[4] Department of Administration, Universidad Católica del Norte, Angamos 0610, Antofagasta 1270236, Chile
[5] Escuela de Computación e Informática, Facultad de Ingeniería, Ciencia y Tecnología, Universidad Bernardo O'Higgins, Santiago 8370993, Chile
* Correspondence: cvidal@utalca.cl; Tel.: +56-9-62002702

Abstract: Feature models (FMs) appeared more than 30 years ago, and they are valuable tools for modeling the functional variability of systems. The automated analysis of feature models (AAFM) is currently a thriving, motivating, and active research area. The product configuration of FMs is a relevant and helpful operation, a crucial activity overall with large-scale feature models. The minimal conflict detection, the diagnosis of in-conflict configuration, and the product completion of consistent partial configuration are significant operations for obtaining consistent and well-defined products. Overall, configuring products for large-scale variability intensive systems (VIS) asks for efficient automated solutions for minimal conflict, diagnosis, and product configuration. Given the relevance of minimal conflict, diagnosis, and product configuration, and the current application of large-scale configuration and FMs for representing those systems and products, the main goals of this research paper are to establish the fundaments of the product configuration of feature models and systematically review existing solutions for the conflict detection, diagnosis, and product completion in FMs from 2010 to 2019. We can perceive that even though modern computing approaches exist for AAFM operations, no solutions exist for assisting the product configurations before 2020. This article reports that in 2020, new solutions appear regarding applying parallel computing for those goals. This research highlights research opportunities for developing new and more efficient solutions for conflict detection, diagnosis, and product completion of large-scale configurations.

Keywords: automated analysis of feature model (AAFM); product configuration; Python; speculative programming

check for updates

Citation: Vidal-Silva, C.; Duarte, V.; Cardenas-Cobo, J.; Serrano-Malebran, J.; Veas, I.; Rubio-León, J. Reviewing Automated Analysis of Feature Model Solutions for the Product Configuration. *Appl. Sci.* **2023**, *13*, 174. https://doi.org/10.3390/app13010174

Academic Editor: Krzysztof Koszela

Received: 11 October 2022
Revised: 6 December 2022
Accepted: 17 December 2022
Published: 23 December 2022

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Variability is a system's capacity to be expanded, altered, customized, or configured for use in unique settings [1]. We can tailor or adapt variability intensive systems (VIS) to particular needs in different domains [2,3]. As Wang et al. [4] highlights, variability has become a key in software systems for the steadily increasing demand for highly customizable products. In software product line engineering (SPLE), variability models such as FMs are primary artifacts for the SPLs' success [5]. Variability modeling specifies all significant and legal combinations of features in a software product line [6].

The manual analysis of variability models is a complicated, tedious, and error-prone task, and the automated analysis of variability models (AAVM) appears to face that issue. For example, Ross et al. [7] present the tool FAMA-OVM for the automated analysis of orthogonal variability models (OVM). Likewise, the works of [8,9] describe the structure, components, and results improvements in applying a process for the AAVM of OVM. In

the FM context, the automated analysis of feature models (AAFM) is a current and active research area. As Benavides et al. [10] indicated, the AAFM and product configurations have many potential synergies because, in the product reconfiguration in dynamic SPLs, both concepts overlap. An FM configuration describes a valid product if the selection of features satisfies all the FM constraints [11].

Inconsistencies in models imply the presence of errors [12,13]. The main challenge of fixing variability models' inconsistencies is setting them for all configurations [14]. The consistency of an FM means that it remains well-formed (syntactic consistency) and defines at least one valid product (semantic consistency) [15]. Checking and resolving FM inconsistencies are essential tasks during the product line evolution [16]. Automated tools for inconsistency checking of FM and configuration are critical activities for successful SPLs [17]. A traditional AAFM approach is to convert the Feature Model into logic formulas supportable by reasoning solvers such as CSP and SAT solvers, to define and execute analysis tasks regarding the model information and reasoning analysis on those solvers [18]. This article focuses on AAFM operations for product configuration.

Benavides et al. [19] in 2010 presented a systematic literature review (SLR) about FM and AAFM operations. Different AAFM proposals exist using various formal approaches such as CSP, SAT, and BDD solvers [20]. The following lines synthesize the state of the art regarding AAFM operations for minimal conflict detection, minimal diagnosis, and the minimal completion of partial product configuration from 2010 to 2019. This chapter has the following organization: first, we present this review's primary method and objectives. Then, we summarize the main discovered results.

Product configuration is the activity of designing a product according to a set of requirements and configuration rules [21]. With further details, product configuration systems need the knowledge base regarding the components and combination rules and the customers' requirements for selecting product components (configuration) that match their preferences [22]. A valid product configuration only depends on the selected features in a consistent knowledge base scenario; that is, each valid product results from the composition of component type instances that respect the set of defined combination rules [23].

Product configuration systems require systematically managing all features and composition rules to analyze the feature selection for desired products. Variability models (VMs) describe the different relationships and configuration options for the variability management of software systems in software engineering. Different VMs exist, such as feature models (FMs) and orthogonal variability models (OVMs). FMs permit representing functional commonalities and variabilities of software systems [19], whereas OVMs permit describing the variant parts of the base model of systems [20]. Kang et al. [24] introduced FMs as part of the FODA (Feature-Oriented Domain Analysis) method, and they became the most used VM in the SPL community afterward.

An FM defines a set of features and their relationships for defining valid feature combinations or products, that is, sets of features that respect the FM's defined relationships. Such as Apel et al. [25] remark, FM permits representing all the products of an SPL. An FM organizes in a tree-like structure that starts at the root feature that identifies the SPL and from which tree branches of features emerge. We can then define a product in terms of the set of features that compose it, and each feature describes an increment of functionality in the products containing that feature [26]. An FM supports binary and set relationships between parent and child features and cross-tree relationships to symbolize dependencies between features. Figure 1 shows an FM for describing an operating system SPL and a valid configuration of it (the gray-colored features). Figure 2 shows the same FM along with a non-valid configuration of it.

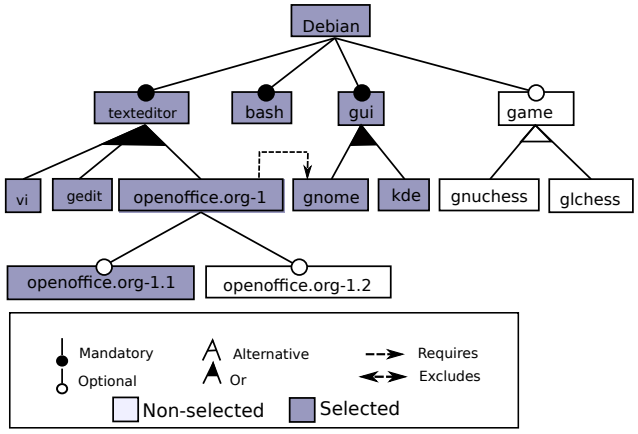


Figure 1. Feature model with a valid configuration example [1].

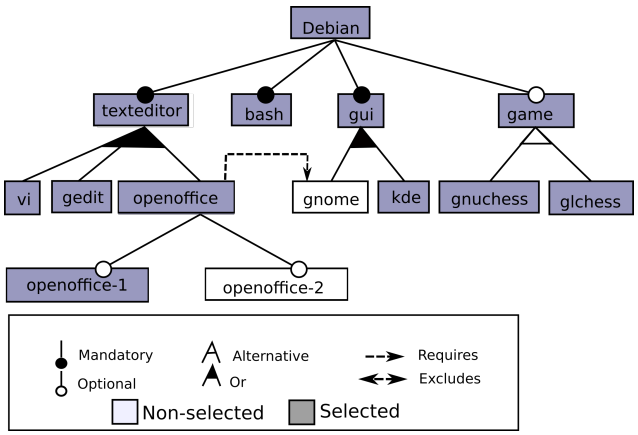


Figure 2. Feature model with a non-valid configuration example [1].

FMs permit organizing the configuration space to facilitate the construction of software variants by describing configuration options using interdependent features or functionalities. A set of features in an FM is called a configuration, and each software variant in an FM identifies a valid configuration or product [27].

Problem Statement, Goal and Contributions

Product configuration permits assisting the mass customization production [23]. AAFM operations for helping the obtention of conflict-free FMs and product configuration are high-value tasks. Nonetheless, existing AAFM operations usually follow a sequential computing approach and cannot scale to work on large-scale and high-variability models. As [1] remark, relevant AAFM solutions for assisting the product configuration are those for the minimal conflict set detection, minimal-preferred diagnosis in a set of constraints in conflict, and the completion of partial products. Next, we summarize those operations.

- Minimal conflict set detection. For a consistent FM that we can define as a set of constraints, a non-consistent configuration violates those constraints. The features model of Figure 1 exemplifies a consistent configuration; that is, this configuration does not violate the FM constraints. For that FM and configuration, if a user selects the features *game*, *gnuchess*, and *glchess*, the resulting configuration will be non-consistent because a conflict exists between features *gnuchess* and *glchess*. That is a minimal conflict set (MCS) because we cannot find a subset of it that results in it being a conflict set. As [28] remark, we can solve an MCS by merely deleting one of its constraints. After finding and solving all the MCS instances, the resulting configuration is valid (conflicts-free).
- Minimal diagnosis. Given the set of constraints of a consistent FM and a non-consistent configuration that violates the FM constraints, a diagnosis is the set of constraints

that permits obtaining a consistent configuration after removing those constraints. For the consistent configuration of Figure 2 after selecting features *game*, *glchess* and *gnuchess*, we know that {*gnuchess*, *glchess*} is a minimal conflict set (MCS). Then, either deleting *gnuchess* or *glchess*, we can obtain a valid configuration (conflict-free); that is, *gnuchess* and *glchess* are examples of diagnosis.

- Minimal completion of products. Achieving valid configurations in large-scale FMs is a time-demanding and complex task. Reasoning tools, usually part of the AAFM process, permit solving that issue.

Therefore, the main contributions of this paper are to identify, evaluate, and classify research work from 2010 to 2019 regarding the minimal conflict set detection, minimal diagnosis, and product completion for the FM product configuration.

The rest of this paper is organized as follows. Section 2 describes the main steps of a Systematic Literature Review (SLR). Section 3 details the applied review process, that is, research questions (RQs), source material, and inclusion criteria. After that, Section 4 defines the main SLR components and presents its main application results. Section 5 describes existing solutions for efficiently achieving the conflict detection and diagnosis tasks along with their main issues to work with large-scale product configurations. Then, that section also summarizes the principal elements and benefits of speculative programming, along with summarizing application result examples. The paper concludes by outlining the benefits of our academic experience and detailing the motivation for continuing with it in the current and future years.

2. Background

A systematic literature review (SLR) is a secondary research study based on previous (primary) studies to answer defined research questions (RQs) [29]. An SLR identifies, evaluates, integrates, and synthesizes evidence and results of relevant primary studies to answer a defined set of RQs, and to analyze possible new results [30,31]. An SLR traditionally consists of the following phases (Figure 3) [32].

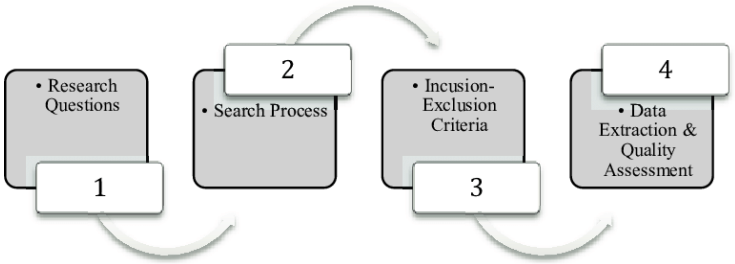


Figure 3. Steps of a traditional SLR process.

1. Research Questions: After defining a topic of research, the first step of an SLR is to determine the research questions (RQs), the main questions to answer in the SLR process.
2. Search Process: The search process presents the sources of search and keywords to select primary studies.
3. Inclusion/Exclusion Criteria: The next step is to establish inclusion/exclusion criteria in the search process and quality criteria for the primary articles.
4. Data Extraction and Quality Assessment: During data extraction, the information is extracted, collected, and organized for its evaluation and final results.

3. Review Process

This SLR aims to discover details about existing AAFM solutions for conflict detection, diagnosis detection, and product completion in FMs from 2010 to 2019. We want to know the main focus, implementation approaches, execution requirements, and computing

performance of existing AAFM solutions for those purposes. Then, we will classify existing options, their scenarios of use, along with their benefits and issues.

3.1. Definition of Research Questions

The RQs are built based on the PICO (Population, Intervention, Comparison, Outcome) strategy [33] to derive search strings applicable to digital libraries. In that context, Population corresponds to FMs; Intervention represents a solution for the conflict detection, diagnosis, or product configuration; the Comparison is blank; and the Outcomes correspond to the found research from 2010 to 2019. The next lines define RQs and search strings based on the PICO approach.

- RQ1: What are the automated solutions for conflict detection in FMs from 2010 to 2019? Following the process described by [31], we defined the next sub-questions.
    1. What conflicts do these solutions focus on?
    2. Are those solutions new or extensions of existing operations?
    3. What are the computing approaches and functioning requirements of those solutions? Do they differ in their base solutions?
    4. Are those solutions able to work on large-scale feature model instances?
    5. What is the computing performance and degree of improvement or deterioration of those solutions regarding their base solutions?
- RQ2: What are the automated solutions for diagnosis in feature model instances from 2010 to 2019? Following the process described by [31], this question inspires the next sub-questions.
    1. What conflict do these solutions focus on?
    2. Are those solutions new or extensions of existing operations?
    3. What are the computing approaches and functioning requirements of those solutions? Do they differ in their base solutions?
    4. Are those solutions able to work on large-scale feature model instances?
    5. What is the computing performance and degree of improvement/deterioration of those solutions regarding their base ones?
- RQ3: What are the automated solutions for the completion of product configuration in feature model instances from 2010 to 2019? Following the process described by [31], this question inspires the next sub-questions.
    1. Are those solutions new or extensions of existing operations?
    2. What are the computing approaches and functioning requirements of those solutions? Do they differ in their base solutions?
    3. Are those solutions able to work on large-scale feature model instances?
    4. What is the computing performance and degree of improvement/deterioration of those solutions regarding their base ones?

3.2. Source Material

We follow an automated selection method to look for papers in Google scholar [34], Scopus [35], and WebOfKnowledge [36]. We used the next key-phrase to look for research publications in those databases.

- $(\{[f \mid F]eature[s \mid \sqcup]\}\ \{[m \mid M]odel[s \mid \sqcup]\}\ AND\ [\{[a \mid A]utomated\}\ OR\ \{[a \mid A]utomatic\}]\ AND\ (\{[a \mid A]nomal[y \mid ies]\}\ OR\ [c \mid C]onflict[s \mid \sqcup]\}\ OR\ (\{[d \mid D]iagnosis\})\ OR\ (\{[c \mid C]ompletion\})\ AND\ \{PubYear >= 2010\}\ AND\ \{PubYear <= 2019\}\ AND\ \{Language\ in(Spanish \mid English)\}$

Examples of valid search strings for our SLR are:

- Feature models automated anomaly.
- Feature models automated conflicts.
- Feature models automated diagnosis.
- Feature models automatic diagnosis.

- Feature models automated completion.
- Feature models automatic completion.

We accomplish a systematic review using a valid search string in the title, abstract, or keywords set for each analyzed paper. PubYear refers to the publication date of the article. Specifically, we looked for papers from 2010 to 2019, the search date, both years inclusive. Scopus and WebOfKnowlede libraries support search strings accordingly to our definition. Regarding the language, we look for papers written in English only. Google scholar permits looking for defined terms either in the title or somewhere in the document. In Google Scholar, we looked for results year-per-year from 2010. We used the primary 20 results per-year for the high quantity of results. We obtained 200 results from Google Scholar, 43 results in Scopus, and 23 results in WebOfKnowledge. Hence, we received 266 results in our automated search. As a first filter step, we discarded duplicated work for obtaining 223 results. Second, after rejecting works that were non-available or non-related with the research topic, we obtained 120 works. According to the Google Scholar database data, there was only one non-available work from the SPLC 2010 proceedings. Still, nobody presented that article in the SPLC 2010 conference, and it did not appear in its proceeding. We contacted the paper's authors, who reported that the paper appeared in another conference using a different title. We decided to discard that work. Third, we applied a filter to reject works non-related to the research questions. Finally, we obtain 51 primary studies in the scope of this review. The next section describes the last inclusion criteria.

3.3. Inclusion Criteria

On the primary results without duplication and non-available data, we applied a general inclusion filter to consider only articles of the AAFM area. We analyzed the title, keywords, and abstract of each found paper to positively support at least one of the inclusion criteria (IC) of Table 1.

Table 1. Quick review research questions.

	Inclusion Criteria
IC-1:	Does this paper presents an AAFM solution for the conflict detection?
IC-2:	Does this paper present an AAFM solution for diagnosis?
IC-3:	Does this paper present an AAFM solution for the product completion?

4. Results

Applying inclusion criteria, we obtain 51 research works. Tables 2–4 the resulting papers ordered alphabetically by the author's name and ascendingly by the publication year. The values of column *FM Kind* are 1 for FODA FMs, 2 for Cardinality-based FMs, 3 for Extended FMs, and 4 for all these options. We use the yellow background color to mark the accomplished options. The column *Large FM* is yellow for those studies that work with FM of 5000 features or more.

Table 2. List of articles ordered by the authors' last name from A to F in the SLR vs. our thesis work.

			Source			Main Goal				Solution			
#	Year	Author	Journal	Conference	Workshop	Conflict	Diagnosis	Completion	Focused Anomaly	Tool	Solver	FMs Kind	Large FMs
1	2013	Acher et al. [37]	✓			✓			Conflict in product/model	✓	✓	1	
2	2014	Achour et al. [38]		✓		✓			Conflict in product/model	✓		1	
3	2014	Afzal et al. [39]		✓		✓			Wrong cardinality, exclusion of mandatory features, requires of excluded features	✓		3	
4	2016	Ananieva et al. [40]	✓			✓			Partial FM dependencies	✓	✓	1	
5	2017	Arcaini et al. [41]	✓			✓			Dead feature, redundant constraint, false optional	✓	✓	1	
6	2015	Arcaini et al. [42]		✓		✓			Faults for mutation	✓	✓	1	
7	2016	Arcaini et al. [43]		✓		✓			Conformance faults	✓	✓	1	
8	2016	Asadi et al. [44]	✓			✓			Potential inconsistency, strong inconsistencs	✓	✓	1	
9	2014	Asadi et al. [45]	✓			✓			Potential inconsistency, strong inconsistency, configuration inconsistency	✓	✓	1	
10	2014	Barreirros & Moreria [46]		✓		✓			Product configuration inconsistencies	✓	✓	1	
11	2018	Bhushan et al. [47]	✓			✓			Conflicts in the FM definition	✓	✓	1	
12	2017	Bhushan et al. [48]	✓			✓			Conflicts in the FM definition	✓	✓	1	
13	2010	Bin Abid [49]			✓	✓			Consistency checking in the product derivation	✓		1	
14	2011	Choi [17]	✓			✓			Dead features, void FM, validity of product configuration	✓	✓	1	
15	2015	Da Silva Costa et al. [50]		✓		✓			False optional, dead features	✓	✓	3	
16	2014	Elfaki et al. [51]	✓			✓			False optional, wrong cardinality	✓		2	
17	2012	Felfernig et al. [52]		✓			✓		Diagnosis	✓	✓	1	
18	2013	Felfernig et al. [28]		✓			✓		Diagnosis	✓	✓	1	
19	2015	Felfernig et al. [53]		✓			✓		Anytime diagnosis	✓	✓	1	
20	2018	Felfernig et al. [54]	✓				✓		Anytime diagnosis	✓	✓	1	
	2020	Vidal-Silva PhD. Thesis		✓	✓	✓	✓	✓	Minimal conflicts set, diagnosis, and minimal preferred completion of product configuration by diagnosis	✓	✓	1	✓

Table 3. List of articles ordered by the authors' last name from G to M in the SLR vs. our thesis work.

			Source			Main Goal				Solution			
#	Year	Author	Journal	Conference	Workshop	Conflict	Diagnosis	Completion	Focused Anomaly	Tool	Solver	FMs Kind	Large FMs
21	2010	Galindo et al. [55]		✓		✓			Discovering inconsistencies between packages	✓	✓	1	
22	2011	Gheyi et al. [56]	✓			✓			Soundness of FM refactoring	✓		1	
23	2012	Guo et al. [15]	✓			✓			Consistent evolution	✓		1	✓
24	2013	Henard et al. [57]		✓			✓		Fixing and re-engineering a FM	✓	✓	1	
25	2019	Hinterreiter et al. [16]		✓		✓			Potential inconsistencies	✓		1	
26	2016	Javed et al. [58]	✓			✓			Void feature model, inconsistent product configuration	✓		1	
27	2015	Khtira et al. [59]	✓			✓			Duplication in evolving FM	✓		1	
28	2015	Khtira et al. [60]	✓			✓			Duplication in evolving FM	✓		1	
29	2016	Kowal et al. [61]		✓			✓		Dead feature, redundancies, false optional	✓	✓	1	
30	2015	Lesta et al. [62]	✓			✓			Void FM, dead feature, false-optional, dead attribute, false-optional attribute values	✓	✓	3	
31	2010	Lisboa et al. [63]	✓			✓			Orphan features, void FMs	✓		1	
32	2011	Lisboa et al. [64]		✓		✓			Redundancies, configuration anomalies, inconsistencies	✓		1	
33	2011	López-Herrejon & Egyed [65]		✓			✓		To effectively identify features to fix for guaranteeing consistent product line configurations	✓	✓	1	
34	2012	López-Herrejon & Egyed [14]		✓			✓		Product configuration	✓	✓	1	
35	2012	Marinho et al. [66]			✓	✓			False optional, dead feature, wrong cardinality	✓		3	
36	2017	Mauro et al. [67]		✓		✓	✓		Void FM, dead features	✓	✓	1	
37	2011	Mazo et al. [68]		✓		✓			Conformance FM & product checking	✓		3	✓
	2020	Vidal-Silva PhD. Thesis		✓	✓	✓	✓	✓	Minimal conflicts set, diagnosis, and minimal preferred completion of configuration by diagnosis	✓	✓	1	✓

Table 4. List of articles ordered by the authors' last name from N to Z in the SLR vs. our thesis work.

			Source			Main Goal			Solution						
#	Year	Author	Journal	Conference	Workshop	Conflict	Diagnosis	Completion	Focused Anomaly	Tool	Solver	FMs Kind	Large FMs
38	2010	Nakajima [69]		✓		✓			FM propositional interpretation and algorithm for locating bugs	✓	✓	1	
39	2010	Nakajima [70]		✓		✓			Unsatisfiable portion (components) of the unsatisfiable FM	✓	✓	1	
40	2018	Nieke et al. [11]		✓		✓	✓		Anomalies in the FM evolution	✓	✓	1	✓
41	2011	Noorian et al. [71]		✓		✓	✓		Detecting and fixing inconsistencies	✓	✓	1	
41	2011	Noorian et al. [71]		✓		✓	✓		Detecting and fixing inconsistencies	✓	✓	1	
42	2014	Quinton et al. [72]		✓		✓			Range inconsistencies	✓	✓	1	✓
43	2012	Ripon et al. [73]	✓				✓		Analysis and verification of FMs	✓		1	
44	2014	Ripon et al. [74]	✓			✓			FM consistency check	✓	✓	1	
45	2016	Schnabel et al. [75]		✓		✓			Bound & gap of cardinality-based FM configuration	✓		2	
46	2019	Vidal [76]	✓				✓		FM configuration	✓	✓	1	
47	2010	Wang et al. [77]	✓			✓			To tolerate inconsistencies in feature models	✓	✓	1	
48	2014	Wang et al. [78]	✓			✓	✓		Detecting and fixing inconsistencies	✓	✓	1	
49	2016	Weckesser et al. [79]	✓			✓			Analysis of bound and interval gaps	✓	✓	2	✓
50	2010	White et al. [80]	✓				✓		Product configuration	✓	✓	1	✓
51	2010	Zhang et al. [81]		✓		✓			Conflicts & anomalies	✓	✓	1	
	2020	Vidal-Silva PhD. Thesis		✓	✓	✓	✓	✓	Minimal conflicts set, diagnosis, and minimal preferred completion of configuration by diagnosis	✓	✓	1	✓

Table 5 presents the number of articles concerning their applied process and main goal. We group the papers into those that use tools only and those that use tools and solvers. This table presents results for the conflict and diagnosis detection only since we do not obtain results for the completion of product configuration in FMs. In the next lines, we answer the RQs along with each sub-question.

Table 5. Filtered articles classified by year and way for reaching their main goal.

			2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Inclusion Criteria (IC)	IC-1	Tools	4	3	2	0	3	2	2	0	0	1
		Tools & Solvers	3	2	0	2	4	3	4	2	1	1
		# Papers	7	5	2	2	7	5	6	2	1	2
	IC-2	Tools	0	0	1	0	0	0	0	0	0	0
		Tools & Solvers	0	1	2	1	0	1	1	0	1	0
		# Papers	0	1	3	1	0	1	1	0	1	0
	IC-1 & IC-2	Tools	0	0	0	0	0	0	0	0	0	0
		Tools & Solvers	2	0	0	0	0	0	0	1	1	0
		# Papers	2	0	0	0	0	0	0	1	1	0
	# Papers	Tools	4	3	3	0	3	2	2	0	0	1
		Tools & Solvers	5	3	2	3	4	4	5	3	3	1

4.1. RQ1: What Are the Automated Solutions for Detecting Conflicts in Feature Models from 2010 to August of 2019?

Table 5 shows that a no-normal tendency exists from 2010 to August of 2019 concerning the research work for the conflict detection in FMs. In the next lines, we answer each sub-question of RQ1.

- Sub-question 1: We classify the obtained papers regarding their main focus in the next classes: (i) Non-Consistent Feature Model (NC FM); (ii) Non-Consistent Configuration (NC Conf); (iii) Non-Consistent Feature Model and Configuration (NC FM/Conf); (iv) Mutation or Evolution of Feature Model (Mut/Evol); and (v) Specific Fault (Spec Fault) such as void FM, empty FM, dead feature, false optional, partial dependencies, and orphan features). The higher number of works focus on the conflict detection on specific FM inconsistencies. Figure 4 depicts a summary of the obtained papers for the conflict detection in FMs.

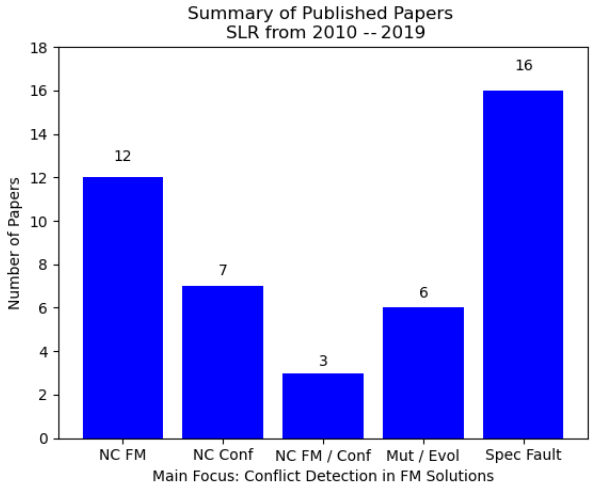


Figure 4. Summary of reviewed papers for the conflict detection in FMs from 2010 to August of August of 2019.

- Sub-question 2: Almost all the articles present new solution proposals for conflict detection in FM.
- Sub-question 3: Table 5 resumes the following results for articles that focused on conflict detection only:

- Seven articles in 2010: four articles using tools, and three articles using tools and solvers.
- Five articles in 2011: three articles using tools, and two articles using both tools and solvers.
- Two articles in 2012: both articles using tools.
- Two articles in 2013: both articles using tools and solvers.
- Seven articles in 2014: three articles using tools, and four articles using tools and solvers.
- Five articles in 2015: two articles using tools, and three articles using tools and solvers.
- Six articles in 2016: two articles using tools, and four articles using tools and solvers.
- Two articles in 2017 using tools and solvers.
- One article in 2018 using tools and solvers.
- Two articles in 2019: 1 article using tools, and 1 article using tools and solvers.

- Sub-question 4: Only six articles present results on the conflict detection of large-scale FMs.
- Sub-question 5: All the reviewed solutions present new or similar performance results concerning their base solutions; that is, those works that are based in previous works are applied to new data sets, and their results preserve their base ones.

4.2. RQ2: What Are the Automated Solutions for Diagnosis in Feature Models from 2010 to August of 2019?

Table 5 shows that a no-normal tendency exists from 2010 to August of 2019 concerning the research work for diagnosis detection in FMs. In the next lines, we answer each sub-question of RQ2.

- Sub-question 1: Such as for sub-question 1 of RQ1, we classify the obtained papers regarding their main focus in the next classes: (i) Non-Consistent Feature Model (NC FM); (ii) Non-Consistent Configuration (NC Conf); (iii) Non-Consistent Feature Model and Configuration (NC FM/Conf); (iv) Mutation or Evolution of Feature Model (Mut/Evol); and (v) Specific Fault (Spec Fault) such as void FM, empty FM, dead feature, false optional, partial dependencies, and orphan features. We appreciate that the number of papers for diagnosis detection in FM is much less than the number of papers for conflict detection in FM. Figure 5 summarizes a classification of the obtained papers for diagnosis detection in FMs.

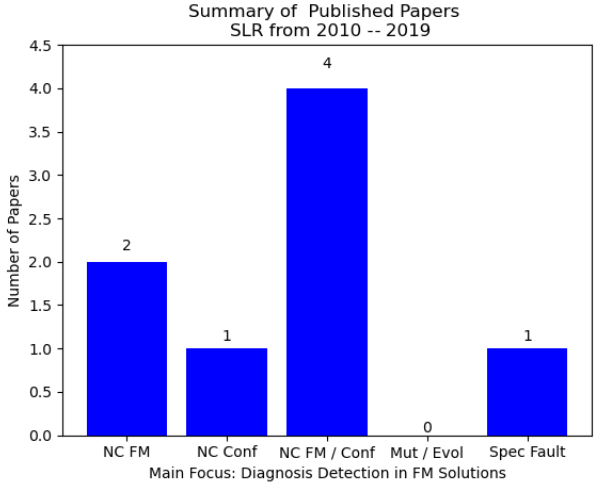


Figure 5. Resume of published papers for diagnosis in FMs from 2010 to August of 2019.

- Sub-question 2: Almost all the articles present new solution proposals. The work of Felfernig et al. [52] proposes FastDiag, and [28] present its application for the FM analysis. The work of [53] presents FlexDiag, base works for [54]. Likewise,

Vidal [76] highlights efficient results of FASTDIAG and FLEXDIAG solutions. The work of [65] focuses on identifying features necessary to fix for obtaining valid product configurations. Other selected articles present automatic solutions, and then they require a diagnosis process.

- Sub-question 3: Table 5 resumes the following results for articles that focused on diagnosis detection only:
  - One article in 2011 using tools and solvers.
  - Three articles in 2012: one article using tools, and two articles using tools and solvers.
  - One article in 2013 using tools and solvers.
  - One article in 2015 using tools and solvers.
  - One article in 2016 using tools and solvers.
  - One article in 2018 using tools and solvers.
- Sub-question 4: No article presents results on large-scale FMs, that is, FMs with 5000 or more features.
- Sub-question 5: All the reviewed solutions present new or similar performance results concerning their base solutions; that is, those works based in previous works are applied to new data sets, and their results preserve their base ones.

Figure 6 depicts a summary regarding the papers for this study that focus in conflict and diagnosis detection.

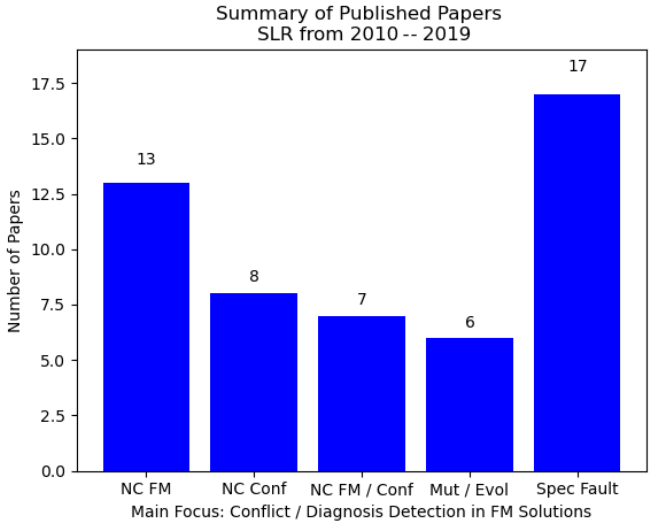


Figure 6. Summary of published papers for the conflict detection, diagnosis and product completion from 2010 to 2019.

4.3. RQ3: What Are the Automated Solutions for the Completion of Products in Feature Models from 2010 to August of 2019?

Concerning the completion of partial products, we did not find research works in the search process. The work of Ananieva et al. [40] focus on the partial feature model, but they did not consider the completion of those models.

5. Discussion

This article performs a standard SLR in software engineering and feature model solutions following recognized previous research [19,29]. We also utilize Google Scholar, Scopus, and WebOfKnowledge, standard source material with higher accuracy and quality of data [82–84], and inclusion criteria according to the research. The results show that solutions existed mainly for the conflict and diagnosis of specific faults. The following lines summarize and exemplify existing solutions for efficient conflict detection and diagnosis. Then, we remark on the main practical issues of those solutions to work on large-scale

configuration models. At last, this section describes speculative programming as a computing approach to looking for efficient solutions for the automated analysis of feature model product configuration in large-scale contexts.

5.1. Efficient Conflict Detection Solution

QUICKXPLAIN [85] is an efficient approach to determine a minimal conflict set. QUICKXPLAIN receives C as the set of suspicious constraints with conflict and B as the set of consistent constraints of the background knowledge. Then, a conflict does not exist if $B \cup C$ is consistent or C is empty. On the other hand, QUICKXPLAIN proceeds by returning the results of the function QX. QX receives the parameters C (initially the complete set of constraints with conflict), B (initially the knowledge base), and $B\delta$ (initially empty) that represents the last items added to B. Function QX follows a divide-and-conquer approach for conflict detection. Hence, $B\delta$ corresponds to the set of constraints added for reviewing the consistency of the knowledge base, and C is the set of constraints to continue analyzing if the current B is consistent. Algorithms 1 and 2 show the pseudo-code of the functions of QUICKXPLAIN.

Algorithm 1 QUICKXPLAIN$(C, B) : CS$

1: **if** CONSISTENT$(B \cup C)$ **then**
2: *return*('no conflict')
3: **else if** $C = \emptyset$ **then**
4: *return*$(\emptyset)$
5: **else**
6: *return*(QX$(C, B, \emptyset)$)
7: **end if**

Algorithm 2 QX$(C = \{c_1..c_m\}, B, B\delta) : CS$

1: **if** $B\delta \neq \emptyset$ **and** INCONSISTENT(B) **then**
2: return$(\emptyset)$
3: **end if**
4: **if** $C = \{c_\alpha\}$ **then**
5: return$(\{c_\alpha\})$
6: **end if**
7: $k = \lfloor \frac{m}{2} \rfloor$
8: $C_a \leftarrow c_1...c_k; C_b \leftarrow c_{k+1}...c_m;$
9: $\Delta_2 \leftarrow$ QX$(C_a, B \cup C_b, C_b);$
10: $\Delta_1 \leftarrow$ QX$(C_b, B \cup \Delta_2, \Delta_2);$
11: return$(\Delta_1 \cup \Delta_2)$

QUICKXPLAIN permits determining one MCS per computation. Felfernig et al. [23] indicate that we need to update adequately or delete one of the constraints of an MCS for solving it, and, if the model is non-consistent yet, to apply QUICKXPLAIN and repeat the process. When the resulting model is consistent, the updated constraints represent a diagnosis or solution for the model. Figure 7 illustrates a FM of the Debian operating system with a non-valid configuration of it: the selection of features {*Debian*, *texteditor*, *vi*, *gedit*, *openoffice*, *openoffice-1*, *bash*, *gui*, *kde*, *game*, *gnuchess*, *glchess*} exemplifies a non-valid configuration that does not respect the required cross-tree constraint between the option *openoffice.org-1* and *gnome* (only the first option is selected), and *gnuchess* and *glchess* are selected for the option *game* when only one feature must be selected (*game* represents an alternative set of features).

Table 6 shows the tracking steps of a QUICKXPLAIN application for the FM configuration example of Figure 7. Column B represents the set of consistent constraints of the base knowledge for the FEATURE MODEL definition and column C is the set of selected

features. The final result represents a minimal conflict that we can solve by updating one of its constraints adequately. In this case, a solution is to update the state of ¬*gnome*.

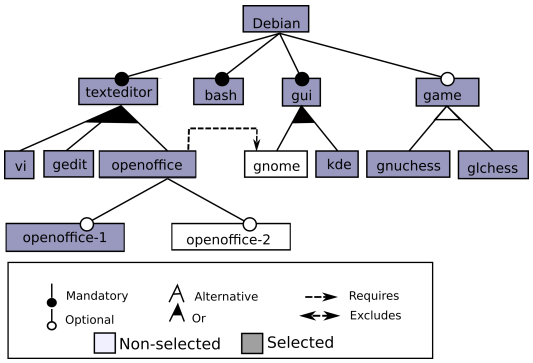


Figure 7. Feature model of a Debian derivative with a non-valid configuration (gray features).

Table 6. QUICKXPLAIN execution tracking on configuration example of Figure 7.

Step	C	B	Bδ	C_a	C_b	Return
1	{Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	B	∅	{Debian, texteditor, bash, gui, game, vi, gedit}	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice-1, ¬ gnome}
2	{Debian, texteditor bash, gui, game, vi, gedit}	$B \cup$ {kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome},	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}			∅
3	{openoffice, kde, gnuchess glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	B	∅	{openoffice kde, gnuchess}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice-1, ¬ gnome}
4	{openoffice, kde, gnuchess}	$B \cup$ {glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}			∅
5	{glchess, openoffice-1, ¬openoffice-2, ¬gnome}	B	∅	{glchess, openoffice-1}	{¬openoffice-2, ¬gnome}	{openoffice-1, ¬gnome}
6	{glchess, openoffice-1}	$B \cup$ {¬openoffice-2, ¬gnome}	{¬openoffice-2, ¬gnome}	{glchess}	{openoffice-1}	{openoffice-1}
7	{glchess}	$B \cup$ {¬openoffice-2, ¬gnome, openoffice-1}	{openoffice-1}			∅
8	{openoffice-1}	$B \cup$ {¬openoffice-2, ¬gnome}	∅			{openoffice-1}
9	{¬openoffice-2, ¬ gnome}	$B \cup$ {¬gnome}	{openoffice-1}	{¬openoffice-2}	{gnome}	{openoffice-1}
10	{¬openoffice-2}	$B \cup$ {openoffice-1} gnome}	{¬gnome}			∅
11	{¬gnome}	$B \cup$ {openoffice-1}	∅			{¬gnome}

5.2. Efficient Diagnosis Solution

FASTDIAG algorithm permits determining a preferred or leading diagnosis concerning a previously defined relevance order of constraints in the knowledge base. FASTDIAG follows the algorithmic structure and reasoning of QUICKXPLAIN for a different purpose, that is, the diagnosis detection without the calculation of MCS instances. Hence, FASTDIAG is based on conflict-independent search strategies [52]. Algorithms 3 and 4 give the pseudo-code of FASTDIAG functions.

Algorithm 3 FASTDIAG(C, AC) : *diagnosis* Δ

1: **if** $C = \emptyset$ **or** INCONSISTENT$(AC - C)$ **then**
2: *return*$(\emptyset)$
3: **else**
4: *return*(FD$(\emptyset, C, AC)$)
5: **end if**

Algorithm 4 FD$(D, C = \{c_1..c_q\}, AC)$: *diagnosis* Δ

1: **if** $D \neq \emptyset$ **and** CONSISTENT(AC) **then**
2: return$(\emptyset)$
3: **end if**
4: **if** $|C| = 1$ **then**
5: return(C)
6: **end if**
7: $k = \lfloor \frac{q}{2} \rfloor$
8: $C_a \leftarrow c_1...c_k$; $C_b \leftarrow c_{k+1}...c_q$;
9: $\Delta_1 \leftarrow$ FD$(C_b, C_a, AC - C_b)$;
10: $\Delta_2 \leftarrow$ FD$(\Delta_1, C_b, AC - \Delta_1)$;
11: return$(\Delta_1 \cup \Delta_2)$

Assuming that conflicts to diagnosis exist, If the conflict set C is non-empty, and AC without C is consistent, algorithm FASTDIAG calls and waits for the recursive results algorithm FD. FD first reviews the consistency of AC as a source of diagnosis. Because always AC contains C and does not contain D, initially S is the constraints set with conflicts and D is empty, when D is not empty, and AC is consistent D is the source of conflict. When that base case is not accomplished, either because D is empty (such as at the beginning) or AC is consistent (this is only possible after removing elements from $AC - D$ represents the last removed elements from AC), then AC is still in conflict, and C is a source of conflict. Then, FD reviews the size of C since if it were minimal (size 1), then C is the diagnosis. If C is not of minimal size, FD proceeds to partition C in the sets C_1 and C_2, of which the last one corresponds to the most preferred partition. Afterwards, FD calls FD over C_2, C_1 and $AC - C_2$ to review if C_2 is the diagnosis source, and if not so, to continue reviewing C_1 with that goal. Tables 7 and 8 show the tracking execution of FASTDIAG on the FEATURE MODEL product configuration of Figure 7. The diagnosis corresponds to the solution of each MCS, in this case, {glchess, ¬gnome}. In column AC, the symbol Γ represents the consistent knowledge base of the FEATURE MODEL.

Table 7. Tracking of the FASTDIAG execution on configuration example of Figure 7 (part 1).

Step	D	C	AC	C_a	C_b	Return
1	∅	{Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{Debian, texteditor, bash, gui, game, vi, gedit}	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
2	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{Debian, texteditor, bash, gui, game, vi, gedit}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit}			∅
3	∅	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice, kde, gnuchess}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
4	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice, kde, gnuchess}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess}			{glchess, ¬ gnome}
5	∅	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1}	{¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
6	{¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1}	{glchess}	{openoffice-1}	{glchess, ¬ gnome}
7	{openoffice-1}	{glchess}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess}			{glchess}
8	{glchess}	{openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1}			∅

Table 8. Tracking of the FASTDIAG execution of configuration example of Figure 7 (part 2).

Step	D	C	AC	C_a	C_b	Return
8	{glchess}	{openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1}			∅
9	{glchess}	{¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{¬ openoffice-2}	{¬ gnome}	{¬ gnome}
10	{¬ gnome}	{¬ openoffice-2}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2}			∅
11	∅	{¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2, ¬gnome}			{¬ gnome}

5.3. Current Issues of Existing Solutions

FMs permit organizing the configuration space to facilitate the construction of software variants by describing configuration options using interdependent features or functionalities. A set of features in an FM is called a configuration, and each software variant in an FM identifies a valid configuration or product [27]. Hence, AAFM operations for assisting the obtention of conflict-free FMs and configurations are high-value tasks.

Even though QUICKXPLAIN and FASTDIAG are computationally efficient in theory, those algorithms are examples of solutions that take a long time to work on large-scale FM configurations. Both solutions cannot use additional computing resources for their sequential computing nature, such as multiple core or network technologies for parallel and distributed computing, respectively. Concerning the completion of products, defining and accomplishing all users' requirements for configuring large-scale systems is a tedious and complex task, and non-efficient solutions exist for assisting in the completion of partial configurations.

5.4. Speculative Programming

Speculative programming is a method for building programs that can run some of their code before the results are required. The pre-execution or pre-calculation of findings that might help calculate the anticipated outcome is known as speculative execution [86]. We may create parallel processes to carry out such thread-level speculative (TLS) pre-calculations [87]. We can distinguish between valid and invalid speculative executions depending on whether the cost of coordinating the parallel activities causes the program execution time to increase or decrease. Because there are already parallel computing resources available, it is possible to use speculative programming to research solutions and ultimately provide more effective ones [88]. A flow diagram containing the tasks *A*, *B*, *C*, and *D* is shown in Figure 8. The initial job is *A*, followed by *B* or *C* depending on the circumstance, and *D* would run just before the work's conclusion. To proceed, task *D* needs the outcomes of *B* or *C*. Figure 9 illustrates the tracking for the sequential execution choices,

each of which consists of three steps in the specified sequence. The speculative execution shown in Figure 10 runs tasks *A*, *B*, and *C* concurrently before moving on to task *D*. Once task *D* had received the right input from the earlier tasks, it would then continue. It can be vital to choose the appropriate input for task *D* before its execution. If this example runs on a system with enough parallel resources, the speculative solution will execute more quickly than the sequential one. This example demonstrates how, when resources for parallel computing are available, speculating about future events enables us to improve the execution time of a group of jobs. To solve the first two problems, we could use speculative programming to look for and efficiently obtain similar answers to the ones we already have.

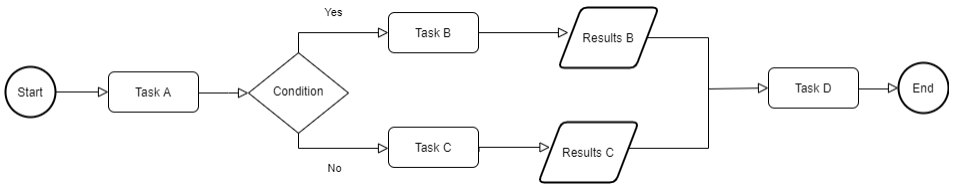


Figure 8. Flow diagram example with four tasks *A*, *B*, *C*, and *D* [1].

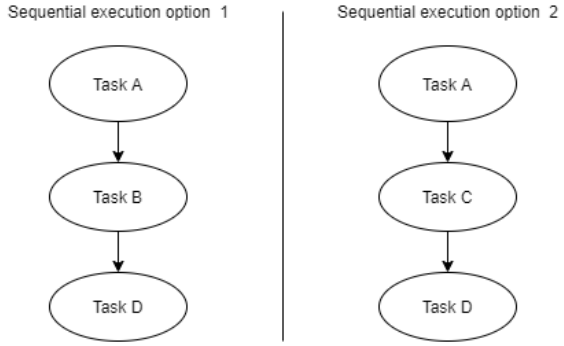


Figure 9. Tracking options for the tasks of Figure 8.

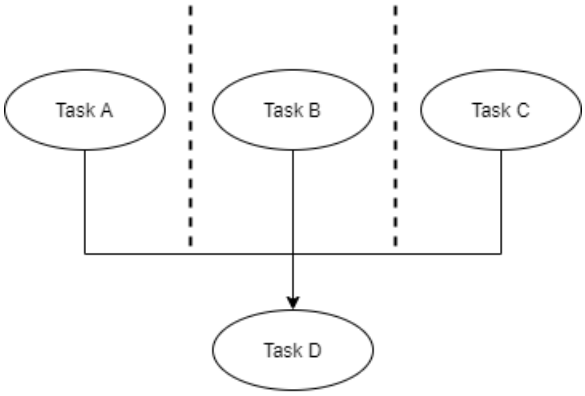


Figure 10. Speculative tracking for the tasks of Figure 8.

The work of Vidal et al. [89] presents the use of speculative programming for conflict detection. We can apply the approach to diagnosis (one of this research team's ongoing projects) because it expands the explanation for using parallel computing and efficiently detecting conflicts. Regarding product completion, the work of Vidal [1] presents PARALLELFASTDIAG, a speculative version of FASTDIAG, an efficient solution for diagnosis applying speculative programming. PARALLELFASTDIAG invites us to use it to obtain a more efficient product completion solution.

6. Conclusions

This survey research points out the maturity of AAFM. Almost half of the obtained papers appear in journals. Concretely, journals published 24 of the 51 reviewed articles, that is, 47.1%; and 27 articles out of 51 in conferences and workshops, that is, 52.9%. We pinpointed the main trends and current state in the AAFM area for the configuration of products. Specifically, this research focused on conflict detection, diagnosis, and product completion solutions. This review guides researchers in defining eventual contributions in the AAFM product configuration area. The obtained results motivate us to continue researching the AAFM product configuration area.

This research demonstrates that there is still work to be done in the AAFM process for conflict detection, diagnosis, and product completion operations. This review did not find work concerning speculative programming in AAFM operations before 2020, nor for product completion. The main author of this research proposed ParallelQuickXPlain [89,90], ParallelFastDiag [1] along with a solution for the product completion [91].

In the context of Industry 4.0, the new manufacturing revolution of mass customization for high-complexity and low-volume production is moving forward [92]. The configuration and customization of products represent high-value activity. For the current computing and information advances, developing more efficient solutions for the product configuration seems suitable. The feasibility of applying speculative programming represents a significant advance. Following the steps of Galindo et al. [93] for using BigData on the AAFM process also seems like a great opportunity.

Author Contributions: Methodology, C.V.-S., V.D., J.C.-C., J.S.-M., I.V. and J.R.L.; Investigation, C.V.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Described and cited research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vidal-Silva, C. Configuration Analysis for Large Scale Feature Models: Towards Speculative-Based Solutionse. Ph.D. Thesis, University of Seville, Sevilla, Spain, 2021.
2. Naselaris, T.; Allen, E.; Kay, K. Extensive sampling for complete models of individual brains. *Curr. Opin. Behav. Sci.* **2021**, *40*, 45–51. [CrossRef]
3. Fortz, S. LIFTS: Learning Featured Transition Systems. In Proceedings of the SPLC '21: 25th ACM International Systems and Software Product Line Conference, Leicester, UK, 6–11 September 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 1–6. [CrossRef]
4. Wang, X.; Wang, W.; Liu, H. Product Model Derivation from Feature Model and Formal Specification. *Appl. Sci.* **2022**, *12*, 6241. [CrossRef]
5. Lindohf, R.; Krüger, J.; Herzog, E.; Berger, T. Software Product-Line Evaluation in the Large. *Empir. Softw. Eng.* **2021**, *26*, 30. [CrossRef]
6. Bhushan, M.; Ángel Galindo Duarte, J.; Samant, P.; Kumar, A.; Negi, A. Classifying and resolving software product line redundancies using an ontological first-order logic rule based method. *Expert Syst. Appl.* **2021**, *168*, 114167. [CrossRef]
7. Roos-Frantz, F.; Benavides, D.; Ruiz-Cortés, A. Automated Analysis of Orthogonal Variability Models using Constraint Programming. In *XV Jornadas de Ingeniería del Software y Bases de Datos*; Los Autores: Valencia, Spain, 2010; pp. 269–280.
8. Pol'la, M.; Buccella, A.; Cechich, A. Automated Analysis of Variability Models: The SeVaTax Process. In *Computational Science and Its Applications—ICCSA 2018, Proceedings of the International Conference on Computational Science and Its Applications, Melbourne, VIC, Australia, 2–5 July 2018*; Gervasi, O., Murgante, B., Misra, S., Stankova, E., Torre, C.M., Rocha, A.M., Taniar, D., Apduhan, B.O., Tarantino, E., Ryu, Y., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 365–381.

9. Pol'la, M.; Buccella, A.; Cechich, A. Using Scope Scenarios to Verify Multiple Variability Models. In *Computational Science and Its Applications—ICCSA 2019, Proceedings of the International Conference on Computational Science and Its Applications, Saint Petersburg, Russia, 1–4 July 2019*; Misra, S., Gervasi, O., Murgante, B., Stankova, E., Korkhov, V., Torre, C., Rocha, A.M., Taniar, D., Apduhan, B.O., Tarantino, E., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 383–399.
10. Benavides, D.; Felfernig, A.; Galindo, J.A.; Reinfrank, F. Automated Analysis in Feature Modelling and Product Configuration. In Proceedings of the ICSR 2013, Pisa, Italy, 18–20 June 2013; pp. 160–175.
11. Nieke, M.; Mauro, J.; Seidl, C.; Thüm, T.; Yu, I.C.; Franzke, F. Anomaly Analyses for Feature-model Evolution. In Proceedings of the GPCE 2018: 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, Boston, MA, USA, 5–6 November 2018; ACM: New York, NY, USA, 2018; pp. 188–201. [CrossRef]
12. Nöhrer, A.; Biere, A.; Egyed, A. Managing SAT Inconsistencies with HUMUS. In Proceedings of the VaMoS '12: Sixth International Workshop on Variability Modeling of Software-Intensive Systems, Leipzig, Germany, 25–27 January 2012; ACM: New York, NY, USA, 2012; pp. 83–91. [CrossRef]
13. Nöhrer, A.; Biere, A.; Egyed, A. A Comparison of Strategies for Tolerating Inconsistencies During Decision-making. In Proceedings of the SPLC '12: 16th International Software Product Line Conference, Salvador, Brazil, 2–7 September 2012; ACM: New York, NY, USA, 2012; Volume 1, pp. 11–20. [CrossRef]
14. Lopez-Herrejon, R.E.; Egyed, A. Towards Fixing Inconsistencies in Models with Variability. In Proceedings of the VaMoS '12: Sixth International Workshop on Variability Modeling of Software-Intensive Systems, Leipzig, Germany, 25–27 January 2012; ACM: New York, NY, USA, 2012; pp. 93–100. [CrossRef]
15. Guo, J.; Wang, Y.; Trinidad, P.; Benavides, D. Consistency maintenance for evolving feature models. *Expert Syst. Appl.* **2012**, *39*, 4987–4998. [CrossRef]
16. Hinterreiter, D.; Feichtinger, K.; Linsbauer, L.; Prähofer, H.; Grünbacher, P. Supporting Feature Model Evolution by Lifting Code-Level Dependencies: A Research Preview. In *Requirements Engineering: Foundation for Software Quality, Proceedings of the 25th International Working Conference, REFSQ 2019, Essen, Germany, 18–21 March 2019*; Springer: Cham, Switzerland, 2019; pp. 169–175. [CrossRef]
17. Choi, S.H. Verification Tool for Feature Models and Configurations using Semantic Web Technologies. *J. Korea Soc. Serv.* **2011**, *10*, 189–201. [CrossRef]
18. Sepúlveda, S.; Cravero, A. Reasoning Algorithms on Feature Modeling—A Systematic Mapping Study. *Appl. Sci.* **2022**, *12*, 5563. [CrossRef]
19. Benavides, D.; Segura, S.; Ruiz-Cortés, A. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Inf. Syst.* **2010**, *35*, 615–636. [CrossRef]
20. Galindo, J.A.; Roos-Frantz, F.; Benavides, D.; Ruiz-Cortés, A.; García-Galán, J. Automated Analysis of Diverse Variability Models with Tool Support. In Proceedings of the XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2014), Cadiz, Spain, 16–19 September 2014; pp. 160–168.
21. Brezočnik, L.; Fister, I.; Podgorelec, V. Swarm intelligence algorithms for feature selection: A review. *Appl. Sci.* **2018**, *8*, 1521. [CrossRef]
22. Felfernig, A.; Friedrich, G.; Jannach, D. Conceptual modeling for configuration of mass-customizable products. *Artif. Intell. Eng.* **2001**, *15*, 165–176. [CrossRef]
23. Felfernig, A.; Hotz, L.; Bagley, C.; Tiihonen, J. *Knowledge-Based Configuration: From Research to Business Cases*, 1st ed.; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2014.
24. Kang, K.C.; Cohen, S.G.; Hess, J.A.; Novak, W.E.; Peterson, A.S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*; Technical Report; Carnegie-Mellon University Software Engineering Institute: Pittsburgh, PA, USA, 1990.
25. Apel, S.; Batory, D.; Kstner, C.; Saake, G. *Feature-Oriented Software Product Lines: Concepts and Implementation*; Springer Publishing Company: Berlin/Heidelberg, Germany, 2013.
26. Batory, D. Feature Models, Grammars, and Propositional Formulas. In *SPLC 2005: Software Product Lines, Proceedings of the 9th International Conference on Software Product Lines, Rennes, France, 26–29 September 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 7–20. [CrossRef]
27. Lienhardt, M.; Damiani, F.; Johnsen, E.B.; Mauro, J. Lazy Product Discovery in Huge Configuration Spaces. In Proceedings of the ICSE '20: ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1509–1521. [CrossRef]
28. Felfernig, A.; Benavides, D.; Galindo, J.; Reinfrank, F. Towards Anomaly Explanation in Feature Models. In Proceedings of the 15th International Configuration Workshop, Vienna, Austria, 29–30 August 2013.
29. Kitchenham, B.; Pearl-Brereton, O.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Inf. Softw. Technol.* **2009**, *51*, 7–15. [CrossRef]
30. Napoleão, B.; Romero-Felizardo, K.; de Souza, É.F.; Vijaykumar, N.L. Practical similarities and differences between Systematic Literature Reviews and Systematic Mappings: A tertiary study. In Proceedings of the SEKE 2017, Pittsburgh, PA, USA, 5–7 July 2017; pp. 85–90. [CrossRef]
31. Petersen, K.; Feldt, R.; Mujtaba, S.; Mattsson, M. Systematic Mapping Studies in Software Engineering. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), Bari, Italy, 26–27 June 2008; pp. 68–77.

32. Halim, A.; Ahmad, R.; Muhamad-Noh, Z.A.; Hazwani, F.; Mohd-Alwi, N. Systematic Review for Network Survivability Analysis in MANETS. *Procedia-Soc. Behav. Sci.* **2015**, *195*, 1872–1881. [CrossRef]
33. Aboud, C.; Andrucioli de Mattos Pimenta, C.; Nobre, M. The PICO strategy for the research question construction and evidence search. *Revista Latino-Americana de Enfermagem* **2007**, *15*, 508–511. [CrossRef]
34. Google. Google Scholar. 2020. Available online: https://scholar.google.cl/ (accessed on 2 February 2020).
35. Elsevier. Scopus. 2020. Available online: https://www.scopus.com/search/form.uri?display=basic (accessed on 2 January 2020).
36. Analytics, C. Web of Science. 2020. Available online: https://apps.webofknowledge.com/ (accessed on 3 January 2020).
37. Acher, M.; Collet, P.; Lahire, P.; France, R.B. FAMILIAR: A domain-specific language for large scale management of feature models. *Sci. Comput. Program.* **2013**, *78*, 657–681. [CrossRef]
38. Achour, I.; Jilani, L.; Ben-Ghezala, H. Towards an extended tool for analysis of extended feature models. In Proceedings of the The 2014 International Symposium on Networks, Computers and Communications, Hammamet, Tunisia, 17–19 June 2014; pp. 1–5. [CrossRef]
39. Afzal, U.; Mahmood, T.; Rauf, I.; Shaikh, Z.A. Minimizing feature model inconsistencies in Software Product Lines. In Proceedings of the 17th IEEE International Multi Topic Conference 2014, Karachi, Pakistan, 8–10 December 2014; pp. 137–142. [CrossRef]
40. Ananieva, S.; Kowal, M.; Thüm, T.; Schaefer, I. Implicit Constraints in Partial Feature Models. In Proceedings of the FOSD 2016: 7th International Workshop on Feature-Oriented Software Development, Amsterdam, The Netherlands, 30 October 2016; ACM: New York, NY, USA, 2016; pp. 18–27. [CrossRef]
41. Arcaini, P.; Gargantini, A.; Riccobene, E.; Vavassori, P. A novel use of equivalent mutants for static anomaly detection in software artifacts. *Inf. Softw. Technol.* **2017**, *81*, 52–64. [CrossRef]
42. Arcaini, P.; Gargantini, A.; Vavassori, P. Generating Tests for Detecting Faults in Feature Models. In Proceedings of the 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, Austria, 13–17 April 2015; pp. 1–10. [CrossRef]
43. Arcaini, P.; Gargantini, A.; Vavassori, P. Automatic Detection and Removal of Conformance Faults in Feature Models. In Proceedings of the 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), Chicago, IL, USA, 11–15 April 2016; pp. 102–112. [CrossRef]
44. Asadi, M.; Gröner, G.; Mohabbati, B.; Gašević, D. Goal-oriented Modeling and Verification of Feature-oriented Product Lines. *Softw. Syst. Model.* **2016**, *15*, 257–279. [CrossRef]
45. Asadi, M.; Mohabbati, B.; Gröner, G.; Gasevic, D. Development and Validation of Customized Process Models. *J. Syst. Softw.* **2014**, *96*. [CrossRef]
46. Barreiros, J.; Moreira, A. Flexible modeling and product derivation in software product lines. *Proc. Int. Conf. Softw. Eng. Knowl. Eng. SEKE* **2014**, *2014*, 67–70.
47. Bhushan, M.; Goel, S.; Kaur, K. Analyzing inconsistencies in software product lines using an ontological rule-based approach. *J. Syst. Softw.* **2018**, *137*, 605–617. [CrossRef]
48. Bhushan, M.; Goel, S.; Loura, A. Improving quality of software product line by analyzing inconsistencies in feature models using an ontological rule-based approach. *Expert Syst.* **2017**, *35*, e12256. [CrossRef]
49. bin Abid, S. Resolving Feature Dependency Implementations Inconsistencies During Product Derivation. In Proceedings of the ECMFA-TW '10: 6th ECMFA Traceability Workshop, Paris, France, 15 June 2010; ACM: New York, NY, USA, 2010; pp. 31–38. [CrossRef]
50. Costa, P.; Marinho, F.; Andrade, R.; Oliveira, T. Fixture—A Tool for Automatic Inconsistencies Detection in Context-aware SPL. In Proceedings of the ICEIS 2015—17th International Conference on Enterprise Information Systems, Barcelona, Spain, 27–30 April 2015; Volume 2, pp. 114–125. [CrossRef]
51. Elfaki, A.; Sim, L.; Vijayaprasad, P.; Md-Johar, M.G.; Fadhil, M. Using a Rule-based Method for Detecting Anomalies in Software Product Line. *Res. J. Appl. Sci. Eng. Technol.* **2014**, *7*, 275–281. [CrossRef]
52. Felfernig, A.; Schubert, M.; Zehentner, C. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artif. Intell. Eng. Des. Anal. Manuf.* **2012**, *26*, 53–62. [CrossRef]
53. Felfernig, A.; Walter, R.; Reiterer, S. FlexDiag: AnyTime Diagnosis for Reconfiguration. In Proceedings of the 17th International Configuration Workshop, Vienna, Austria, 10–11 September 2015.
54. Felfernig, A.; Walter, R.; Galindo, J.A.; Benavides, D.; Erdeniz, S.P.; Atas, M.; Reiterer, S. Anytime diagnosis for reconfiguration. *J. Intell. Inf. Syst.* **2018**, *51*, 161–182. [CrossRef]
55. Galindo, J.; Benavides, D.; Segura, S. Debian Packages Repositories as Software Product Line Models. Towards Automated Analysis. In Proceedings of the 1st International Workshop on Automated Configuration and Tailoring of Applications, Antwerp, Belgium, 20 September 2010; pp. 29–34.
56. Gheyi, R.; Massoni, T.; Borba, P. Automatically Checking Feature Model Refactorings. *J. UCS* **2011**, *17*, 684–711. [CrossRef]
57. Henard, C.; Papadakis, M.; Perrouin, G.; Klein, J.; Le-Traon, Y. Towards automated testing and fixing of re-engineered Feature Models. In Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 1245–1248. [CrossRef]
58. Javed, M.; Naeem, M. Automated inconsistency detection in feature models: A generative programming based approach. *Selforganizology* **2016**, *3*, 59–74.

59. Khtira, A.; Benlarabi, A.; Asri, B. Duplication Detection When Evolving Feature Models of Software Product Lines. *Information* **2015**, *6*, 592–612. [CrossRef]
60. Khtira, A.; Benlarabi, A.; Asri, B. A Tool Support for Automatic Detection of Duplicate Features during Software Product Lines Evolution. *IJCSI Int. J. Comput. Sci. Issues* **2015**, *12*, 1–10.
61. Kowal, M.; Ananieva, S.; Thüm, T. Explaining Anomalies in Feature Models. *SIGPLAN Not.* **2016**, *52*, 132–143. [CrossRef]
62. Lesta, U.; Schaefer, I.; Winkelmann, T. Detecting and Explaining Conflicts in Attributed Feature Models. In Proceedings of the Proceedings 6th Workshop on Formal Methods and Analysis in SPL Engineering, FMSPLE@ETAPS 2015, London, UK, 11 April 2015; pp. 31–43. [CrossRef]
63. Barachisio-Lisboa, L.; Cardoso-Garcia, V.; de Lemos Meira, S.R.; de Almeida, E.S. A Support Tool for Domain Analysis. In Proceedings of the Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, 27–29 January 2010; pp. 175–178.
64. Barachisio-Lisboa, L.; Cardoso-Garcia, V.; Santana-de Almeida, E.; Romero-de Lemos Meira, S. ToolDAy: A Tool for Domain Analysis. *Int. J. Softw. Tools Technol. Transf.* **2011**, *13*, 337–353. [CrossRef]
65. Lopez-Herrejon, R.E.; Egyed, A. Searching the variability space to fix model inconsistencies: A preliminary assessment. In Proceedings of the Third International Symposium Search Based Software Engineering SSBSE 2011, Szeged, Hungary, 10–12 September 2011.
66. Marinho, F.G.; Maia, P.H.M.; Andrade, R.M.C.; Vidal, V.M.P.; Costa, P.A.S.; Werner, C. Safe Adaptation in Context-aware Feature Models. In Proceedings of the FOSD '12: 4th International Workshop on Feature-Oriented Software Development, Dresden, Germany, 24–25 September 2012; ACM: New York, NY, USA, 2012; pp. 54–61. [CrossRef]
67. Mauro, J.; Nieke, M.; Seidl, C.; Yu, I.C. Anomaly Detection and Explanation in Context-Aware Software Product Lines. In Proceedings of the SPLC '17: 21st International Systems and Software Product Line Conference, Sevilla, Spain, 25–29 September 2017; ACM: New York, NY, USA, 2017; Volume B, pp. 18–21. [CrossRef]
68. Mazo, R.; Lopez-Herrejon, R.; Salinesi, C.; Diaz, D.; Egyed, A. Conformance Checking with Constraint Logic Programming: The Case of Feature Models. In Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference, Munich, Germany, 18–22 July 2011; pp. 456–465. [CrossRef]
69. Nakajima, S. Semi-automated Diagnosis of FODA Feature Diagram. In Proceedings of the SAC '10: 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; ACM: New York, NY, USA, 2010; pp. 2191–2197. [CrossRef]
70. Nakajima, S. Non-clausal Encoding of Feature Diagram for Automated Diagnosis. In *SPLC 2010: Software Product Lines: Going Beyond, Proceedings of the 14th International Conference on Software Product Lines: Going Beyond, Jeju Island, Republic of Korea, 13–17 September 2010*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 420–424.
71. Noorian, M.; Ensan, A.; Bagheri, E.; Boley, H.; Biletskiy, Y. Feature model debugging based on description logic reasoning. In Proceedings of the DMS 2011—17th International Conference on Distributed Multimedia Systems, Florence, Italy, 18–20 October 2011; pp. 158–164.
72. Quinton, C.; Pleuss, A.; Berre, D.L.; Duchien, L.; Botterweck, G. Consistency Checking for the Evolution of Cardinality-based Feature Models. In Proceedings of the SPLC '14: 18th International Software Product Line Conference, Florence, Italy, 15–19 September 2014; ACM: New York, NY, USA, 2014; Volume 1, pp. 122–131. [CrossRef]
73. Ripon, S.; Azad, K.; Hossain, S.J.; Hassan, M. Modeling and Analysis of Product-line Variants. In Proceedings of the SPLC '12: 16th International Software Product Line Conference, Salvador Brazil, 2–7 September 2012; ACM: New York, NY, USA, 2012; Volume 2, pp. 26–31. [CrossRef]
74. Ripon, S.; Piash, M.; Hossain, A.; Uddin, M.S. Semantic WebBased Analysis of Product Line Variant Model. *Int. J. Comput. Electr. Eng.* **2014**, *6*, 1–6. [CrossRef]
75. Schnabel, T.; Weckesser, M.; Kluge, R.; Lochau, M.; Schürr, A. CardyGAn: Tool Support for Cardinality-based Feature Models. In Proceedings of the VaMoS '16: Tenth International Workshop on Variability Modelling of Software-intensive Systems, Salvador, Brazil, 27–29 January 2016; ACM: New York, NY, USA, 2016; pp. 33–40. [CrossRef]
76. Vidal-Silva, C. Reviewing Diagnosis Solutions for Valid Product Configurations in the Automated Analysis of Feature Models. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 526–532. [CrossRef]
77. Wang, B.; Hu, Z.; Xiong, Y.; Zhao, H.; Zhang, W.; Mei, H. Tolerating Inconsistency in Feature Models. *CEUR Workshop Proc.* **2010**, *661*, 15–20.
78. Wang, B.; Xiong, Y.F.; Hu, Z.J.; Zhao, H.Y.; Zhang, W.; Mei, H. Interactive Inconsistency Fixing in Feature Modeling. *J. Comput. Sci. Technol.* **2014**, *29*, 724–736. [CrossRef]
79. Weckesser, M.; Lochau, M.; Schnabel, T.; Richerzhagen, B.; Schürr, A. Mind the Gap! Automated Anomaly Detection for Potentially Unbounded Cardinality-Based Feature Models. In *Fundamental Approaches to Software Engineering, Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, Eindhoven, The Netherlands, 2–8 April 2016*; Springer New York, Inc.: New York, NY, USA, 2016; Volume 9633, pp. 158–175. [CrossRef]
80. White, J.; Benavides, D.; Schmidt, D.C.; Trinidad, P.; Dougherty, B.; Ruiz-Cortes, A. Automated Diagnosis of Feature Model Configurations. *J. Syst. Softw.* **2010**, *83*, 1094–1107. [CrossRef]
81. Zhang, G.; Ye, H.; Lin, Y. Modelling quality attributes in feature models in software product line engineering. In Proceedings of the 6th International Conference on Software and Database Technologies, Seville, Spain, 18–21 July 2011; Volume 2, pp. 249–254. [CrossRef]

82. Paul, J.; Lim, W.M.; O'Cass, A.; Hao, A.W.; Bresciani, S. Scientific procedures and rationales for systematic literature reviews (SPAR-4-SLR). *Int. J. Consum. Stud.* **2021**, *45*, O1–O16. [CrossRef]
83. Ali, N.b.; Tanveer, B. A Comparison of Citation Sources for Reference and Citation-Based Search in Systematic Literature Reviews. *e-Inform. Softw. Eng. J.* **2022**, *16*, 220106.
84. Valente, A.; Holanda, M.; Mariano, A.M.; Furuta, R.; Da Silva, D. Analysis of Academic Databases for Literature Review in the Computer Science Education Field. In Proceedings of the 2022 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 8–11 October 2022; pp. 1–7.
85. Junker, U. QuickXPlain: Preferred Explanations and Relaxations for Over-constrained Problems. In Proceedings of the 19th national conference on Artifical intelligence, San Jose, CA, USA, 25–29 July 2004; AAAI Press: San Jose, CA, USA, 2004; pp. 167–172.
86. Tatemura, J. Speculative parallelism of intelligent interactive systems. In Proceedings of the IECON '95—21st Annual Conference on IEEE Industrial Electronics, Orlando, FL, USA, 6–10 November 1995; Volume 1, pp. 193–198. [CrossRef]
87. Martínez, J.F.; Torrellas, J. Speculative Synchronization: Applying Thread-Level Speculation to Explicitly Parallel Applications. In Proceedings of the ASPLOS X: 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, 5–9 October 2002; Association for Computing Machinery: New York, NY, USA, 2002; pp. 18–29. [CrossRef]
88. Bérenger, B. Increasing the degree of parallelism using speculative execution in task-based runtime systems. *PeerJ Comput. Sci.* **2019**, *5*, e183. [CrossRef]
89. Vidal-Silva, C.; Felfernig, A.; Galindo, J.A.; Atas, M.; Benavides, D. Explanations for Over-Constrained Problems with Parallelized QUICKXPLAIN. In *Proceedings of the Journal of Intelligent Information Systems—Integrating Artificial Intelligence and Database Technologies*; Helic, D., Leitner, G., Stettinger, M., Felfernig, A., Raś, Z.W., Eds.; Springer International Publishing: Cham, Switzerland, 2021.
90. Vidal-Silva, C.; Felfernig, A.; Galindo, J.A.; Atas, M.; Benavides, D. A Parallelized Variant of Junker's QuickXPlain Algorithm. In *Proceedings of the Foundations of Intelligent Systems*; Helic, D., Leitner, G., Stettinger, M., Felfernig, A., Raś, Z.W., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 457–468.
91. Vidal, C.; Galindo, J.A.; Giráldez, J.; Benavides, D. Automated completion of partial configurations as a diagnosis task. Using FastDiag to improve performance. In *ISMIS 2020: Intelligent Systems in Industrial Applications, Proceedings of the International Symposium on Methodologies for Intelligent Systems, Graz, Austria, 23–25 September 2020*; Springer: Cham, Switzerland, 2020; pp. 107–117.
92. Lee, C.H.; Chen, C.H.; Lin, C.; Li, F.; Zhao, X. Developing a quick response product configuration system under industry 4.0 based on customer requirement modelling and optimization method. *Appl. Sci.* **2019**, *9*, 5004. [CrossRef]
93. Galindo, J.A.; Turner, H.; Benavides, D.; White, J. Testing Variability-intensive Systems Using Automated Analysis: An Application to Android. *Softw. Qual. J.* **2016**, *24*, 365–405. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.