ISSN (Print) : 2319-8613
ISSN (Online) : 0975-4024

A Systematic Literature Review on Feature Oriented Software Development Paradigm

Kala K.U. [#1], M. Nandhini [#2]
[1]Research Scholar, [2]Assistant Professor
[1,2]Department of Computer Science, Pondicherry University, Puducherry-14, India
[1]kalaunni88@gmail.com
[2]mnandhini2005@yahoo.com

***Abstract* - Feature-Oriented Software Development (FOSD) is a software development paradigm focusing on the overall process of software development of large-scale software systems. The main focus of feature orientation is to divide the software product line systems (SPLs) in terms of features. A feature represents a product requirement of a stakeholder. From the set of features, various software products have automatically derived that share some features and differ in another feature based on customer requirement. The main objective of the work is to identify the state of the art in the overall software development process using FOSD paradigm. The other focuses are finding the phase of FOSD which need more focus, based on automation, and identify the quantity and the types of researches on FOSD techniques to address the software product line development concerns and thus finding the areas discussed and problems addressed so far in the published papers of FOSD. The review was conducted by following the Kitchenham's guidelines for conducting the systematic literature review, with five research questions and assessed 118 publications from the origin to 2017 from the 5 electronic sources of relevance to FOSD Research and selected 63 from them for detailed study. FOSD is an emerging trend in software development based on automation. Our Study analyses each area, in which existing research is focused and propose the ongoing phases of interest in feature-oriented software development and revealed the need of further research in this field.**

Keywords - Feature Oriented Software Development, Systematic Review, Feature Model, Automatic software construction, Systematic literature Review.

I. INTRODUCTION

Feature-oriented software development (FOSD) is one of the software product line development paradigms for the construction, customization, and synthesis of large-scale software systems [1]. The focus of feature orientation is to arrange and structure the overall product line development process in terms of features and corresponding artifacts. A feature represents a product characteristic of a stakeholder. In this paradigm, the decomposition of a software system in terms of features occurs and from that set different software products are often generated that share main features and differ in other features [1].

The software products can be developed automatically using FOSD paradigm. That automatic development is the main aim of FOSD. The major phases of FOSD that leads to the automatic generation of software products are domain engineering and application engineering. In domain engineering, the major processes are domain analysis (domain scoping and variability modeling) and domain implementation (development of feature artifacts). In Application engineering feature selection and product derivation is the two processes. The software product is derived automatically by integrating the artifacts of corresponding features selected.

In methodological stipulations, feature acclimatization tackle features explicit in requirements, design code, testing and so forth-across the entire wheel of life. It would be tranquil to flesh out software if its features would be precise and overt in design and code and if software could be generated solely by composing features. Currently, there is a multitude of different methods, languages, frameworks and tools for FOSD. In order to develop models and fully automated frame work, a thorough understanding of FOSD, by considering all aspects of concerns of it.

A. Background

Automated software production is an advanced and trending area of software development. Software products are so often build by hand that have gotten it down to a Science. Aim of Science of Automated Design (SOAD) is the beginning of automated production. FOSD is playing a major role in SOAD. FOSD is a software product line generation technique in software engineering. It provides a newly automated manner for the development, customization, and synthesizers of large-scale software package systems from a collection of features. A feature may be a unit of the practicality of software that satisfies a demand, represents a design decision and provides a possible configuration problem. In alternative words, a feature represents a product characteristic of a stakeholder. FOSD aims at the separation of concerns in terms of features, even for crosscutting and interacting features, and provides corresponding abstraction and implementation mechanisms. The main focus of FOSD is

to decompose a software system as a collection of features and from that set different software products are often generated that share fundamental features and differ in other features.

The essential properties of FOSD are reuse, structure and variation. Developers structure the system design and code in terms of features, utilizes features a unit for reuse, and variants is achieved by some features unique to the specific customer. While discussing about FOSD it is important to clear the concept of feature in detail. Features can be defined technically by Apel et al. [1] as "a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder's requirement, to implement and encapsulate a design decision, and to offer a configuration option". The concept of features is emphasized in each phases of the software life cycle and appropriate concern is needed for the analysis, design, programming techniques, method, tools also in theory. Features can be defined in different ways using simple abstract words or using technical terms [1]. Each researcher defines the feature in his view point some of them are summarized in Table I.

Table I. Definition of Features

Author	**Definition**
Kang et al. [2]	"a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems"
Kang et al. [3]	"a distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained"
Czarnecki and Eisenecker [4]	"a distinguishable characteristic of a concept (e.g., system, component, and so on) that is relevant to some stakeholder of the concept"
Bosh [5]	"a logical unit of behavior specified by a set of functional and non-functional requirements"
Chen et al. [6]	"a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements"
Batory et al. [7]	"a product characteristic that is used in distinguishing programs within a family of related programs"
Classen et al. [8]	"a triplet, f = (R, W, S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification"
Zave [9]	"an optional or incremental unit of functionality"
Batory [10]	"an increment of program functionality"
Apel et al. [11]	"a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder's requirement, to implement and encapsulate a design decision, and to offer a configuration option"

The difference between feature and other programming paradigm concepts (object, function and aspect) are huge. Features differentiate one product from the other by its externally visible characteristics, where as other programming concepts deals with only internal details of the system. The concept of abstraction is only understandable to the persons with knowledge of programming paradigms.

B. Problem Statement and Justification

The objective of the study is to explore all the aspects of research in the FOSD paradigm. There is no other review published regarding FOSD so far. To that end, we conducted a systematic literature review because it is the best method for identifying and evaluating the studies of a particular domain by make use of a set of predefined research questions. Our systematic literature review helps to obtain a fair and goal oriented evaluation of FOSD. In this survey; we give an overview about the roots and the advancements in feature oriented software development approach focusing on development of efficient and effective software product lines. So our systematic literature review helps the researchers to catch the information about the state of the art techniques and practices for FOSD and the research gaps in existing work. Our systematic literature review points out the strength and weaknesses of existing research based on the review questions and reveal the future scope of research.

We use Goal- Question-Metric (GQM) paradigm [3] to define the goal of our systematic review.

Purpose: Study and analyze.
Issue: Feature Oriented software development paradigm.
Object: In software product line development.
Viewpoint: From the view point of researchers.

In this systematic review, we aim at identifying and assessing approaches for implementing FOSD, optional feature problem, feature defect prediction, feature interaction problem, and find the scope of fully automation in the era of automatic development of software. Our research questions focus on investigating different aspects of existing methods and helping to identify the areas for improvement. The audience of this systematic review includes researchers who would like to obtain an overview of the topic, and practitioners who would like to get familiar with existing methods in order to apply them in industry.

C. Related Reviews

In 2009, Apel Swen et al. [1] conducted a literature study on feature oriented software development that is the one and only one review regarding this topic. In that survey, they compare FOSD with other development approaches and thus provide the roots and advancements in the field of FOSD. Their systematic summarization of several promising work in the field of FOSD give the idea of different lines of research and open issues present at that time. Their prominent aim is to attract more researchers to FOSD community. It has been around 10 years advancements now so we conduct further systematic study on the same topic.

D. Review Questions

The overall research objective of this study is to find and analyze state of the art in FOSD researches based on the available journals. This objective has been broken down into three high-level research questions (RQs) which, in turn, will drive the review method. Table II shows the RQs and their motivation.

Table II. Review Questions and Their Motivation.

Research Question	**Interest and Motivation**
RQ1. What is the amount of automation acquired in each phase of FOSD?	The motivation behind this question is to find the level of automation in each phase of feature oriented software development in research focus on developing a fully automated software framework.
RQ2. Which will be the best approach for Implementing variability in FOSD?	Consequently, by answering this RQ, we can get information about: existing implementation approaches and can measure the criteria to find the best one
RQ3. What are the implementation techniques for solving feature interaction problem and optional feature problem?	To identify which are the implementation techniques proposed in the literature for feature interaction problem, identifying modeling techniques, analysis, particular notations and guidelines
RQ4. How refactoring technique is used in FOSD?	To identify and understand the different refactoring used in feature oriented software development.
RQ5. How can we derive features from an application domain?	The main process of FOSD is to analyze the requirements of the domain and derive corresponding features. The various methods for deriving feature and feature relationship are understood by answering this question
RQ6. What are the uses of defects prediction techniques in features?	Quality assurance is difficult in SPLs because codes of features may scattered so individual and integration testing may difficult. If we can predict feature defects quality assurance will become easy

II. REVIEW APPROACH

The systematic review was designed in accordance with the systematic review procedures and processes defined by Kitchenham [12,13]. According to Kitchenham [13], there are 10 sections in the structure of a systematic review: 1. Title; 2. Authorship; 3. Executive summary or abstract; 4.Background; 5. Review questions; 6. Review Method; 7. Inclusion and exclusion of studies; 8.Results; 9. Discussion; and 10.Conclusion. The first 5 sections have been covered so far. The review method comprises four sections: 1. Data sources and search strategy; 2. Study selection; 3. Data extraction; and 4. Data synthesis. This section comprises the review method and the inclusion and exclusion of studies. The results, discussion and conclusion are presented in the next section

A. Data Sources and Search Strategy

1) *Data sources:*

In every systematic mapping, the primary studies are identified by using automatic searches on scientific bibliographies or browsing manually by the acquiring the works from journals and conferences of the target field. In our systematic mapping, we applied an automatic search that was complemented with manual searches in the specific venues listed. The aim of this search process was to find as many published papers related to the research questions as possible using a systematic search strategy.

In order to get the primary basic concepts of feature oriented software development and to gather the keywords for searching the data sources, we decided to add the book by Sven Apel et al.[16] about Feature Oriented Software Product Lines because this is the only book that is completely devoted to the study of the concept of FOSD. Sven Apel and his co-researchers are behind most of the researches happened in this field. The sources of our studies were: ACM, IEEE, Springer, Wiley Inter Science, and Google Scholar. The quality of these sources guarantees the quality of the study. The main source of recent papers is from the FOSD conference held continuously every year from 2009 to 2016.

2) *Search strategy:*

The papers which are written in English are selected to study. To answer the questions and carry out the initial search using five steps mentioned in table III using the keywords mentioned in Table IV.

Table III. Data Search Strategy

Steps	**Explanation**
1.Research Scoping	We identify the fundamental areas of research from[14] and define the scope by considering the research questions.
2.Extract Keywords	Keywords or strings for searching is extracted from the defined area. The extracted keywords are shown in table 4.
3.Design Search Expression	Describe search expressions based on the scope and keywords
4.Application of Search Expression	Use the search expressions in the sources listed in the section 2.1.1 to find the papers already published in our review area
5.Snow ball readings	Analyze the reference list of identified papers to further search and thus we do not miss any researches

Table IV. Search Keywords

Category	**Area**	**Keywords/Strings**
A	Feature Oriented Software Development	X1- Phases of FOSD process
		X2- Domain engineering
		X3- Application engineering
		X4- Automated software construction
		X5- Feature model
		X6- Variability implementation techniques
		X7- Optional feature problem
		X8- Feature interaction problem
		X9- Product line refactoring
		X10- Refactoring
		X11- Feature extraction
		X12- Feature defect prediction
B	Nature of Study	Y1 – Case study
		Y2 – Experiment
		Y3 – Surveys
		Y4 – Industrial
		Y5 – Literature reviews

B. Study Selection (Inclusion and Exclusion of Studies)

From the sources mentioned in previous section, study selection is started by applying the keyword shown in table 4 using AND or OR logical connectives and then followed the steps as shown in table V based on the inclusion exclusion criteria. In initial search we got 118 papers and after the study selection process we select 63 papers for the detailed study.

Table V. Study Selection Steps and Inclusion Criteria

Analysis Phase	Inclusion Criteria	Exclusion Criteria	Number of Papers
1. Initial search	✓ Papers written in English ✓ Available online ✓ Contain search keywords and strings	▪ Publications not written in English ▪ Publications before 1990	118
2.Removing duplicates	✓ Different in concepts ✓ Different in methods ✓ Different in strategies	▪ Duplicate Papers ▪ Similar concepts, methods and strategies	109
3.Scrutinizing titles	✓ Only published in journals, conferences, workshops and books ✓ Not an editorial, seminar, tutorial or discussion	▪ Not Subject to peer review	92
4.Analyzing abstracts	✓ Experiments, case studies, literature reviews, ✓ industrial and surveys	▪ Methodologically unsound studies ▪ Lack of quality	80
5.Analysing Introduction and conclusion	✓ Main contribution in the areas of search strings related to review questions	▪ Lack of focus on FOSD	69
6. Fast reading focusing on contributions to review questions	✓ Reported significant contribution ✓ Originality of work ✓ Sole focus related to the theme of this review study	▪ Publications in unsystematic manner ▪ Not contributing to solving research questions	63

C. Study Quality Assessment

Our research source is only the publications, which maintained high quality levels. So the initial basic quality is achieved from the initial stage itself. The papers which are selected by applying inclusion exclusion criteria is passed through some quality assessment questions shown in table VI.The quality criteria for the study is aggregated and shown in the quality assessment form. The objective is to assess the quality of the paper before data extraction. The papers which have satisfied the quality criteria are moved to the next section for collecting more detailed information extraction.

Table VI. Quality Criteria

Item	Quality criteria
1	Does study report unambiguous and clear in terms of concepts?
2	Are the findings are based on evidence and arguments?
3	Is the paper well/appropriately referenced?
4	Number of participants?
5	Is Internal validity and external validity achieved?
6	Does the report is unbiased?

D. Data Extraction

After completing the study selection process, we extracted data from the selected papers using the data extraction form as Table VII. We recorded all the necessary information on each paper got from data extraction form to gather information on our review questions.

Table VII. Data Extraction Form

Aspects	Details
Study ID	Paper ID
Title	Title of paper
Author	Name of authors
Publishers	Name of publishing authority
Publishing date	Date of publication
Study focus	Focus and perspective of paper
Focused Phase of FOSD	Which phase of FOSD the study is focused
Contribution to RQ1	The level of automation acquired(if any)
Contribution to RQ2	Variability implementing approach(if any)
Contribution to RQ3	Contribution to feature interaction problem(if any)
Contribution to RQ4	Contribution to solve optional feature problem(if any)
Contribution to RQ5	Contributed refactoring technique(if any)
Contribution to RQ6	Contribution to derive features from the domain(if any)
Study findings	Lessons learned from the paper
Knowledge gaps	Limitations and scope of future work

E. Data Synthesis

After the study selection and data extraction process, data synthesize is performed for collating and summarizing the selected study. According to Kitchenham [12, 13], there are two main methods of data synthesis: Qualitative and Quantitative. We are using a mixed approach here for data synthesize. The extracted data were represented using quantitative methods and make use of this we did the qualitative synthesize to answer our questions.

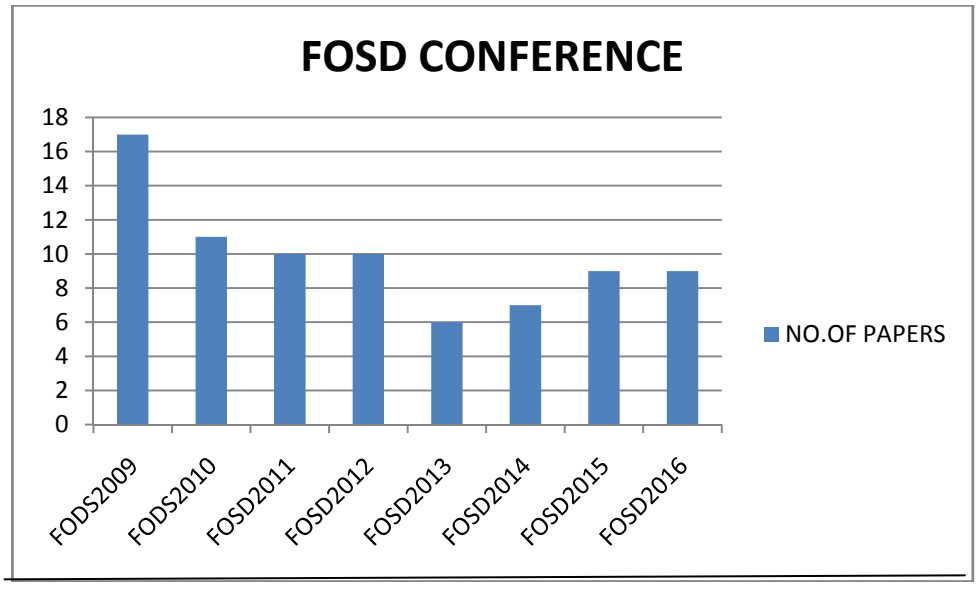

Fig. 1. Paper publications in FOSD community

Table VIII. Selected Studies for Each Review Question

Review Questions	Related selected studies
RQ1	S1, S2, S3, S4
RQ2	S4, S23-S57
RQ3	S4, S5,S6, S7,S10
RQ4	S4, S11-S22
RQ5	S4,S5, S63
RQ6	S58, S59, S60, S61, S62

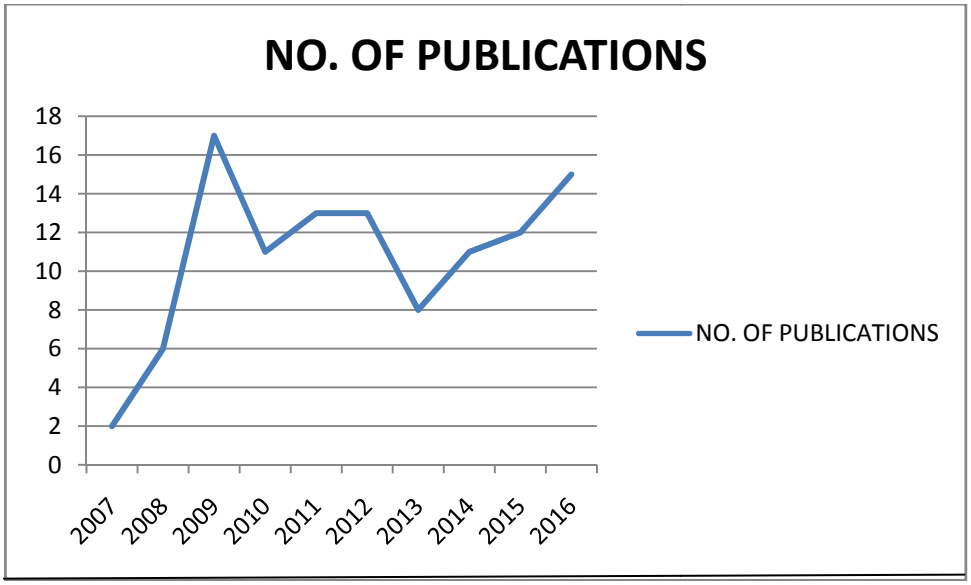

Fig. 2. Number of publication growth

III. RESULTS

A. *Results: RQ1*

Automated software production is advanced and trending area of software development. Aim of Science of Automated Design (SOAD) is the beginning of automated production. FOSD is the major paradigm in SOAD. FOSD is a software product line generation technique in software engineering. It provides a newly automated manner for the development, customization, and synthesizers of large-scale software package systems from a collection of features. Our first Review question is "What is the amount of automation acquired in each phase of FOSD?" For answering this question we have to answer the two sub questions?

- What are the phases of FOSD?
- How each phase is automated?

Automated derivation of software products is the aim of FOSD. That is there is no programmer intervention is needed to assemble and integration of features for writing glue or boilerplate code. The major phases of FOSD that leads to automatic generation of software products by Swen Apel [1] are shown in the figure 3.

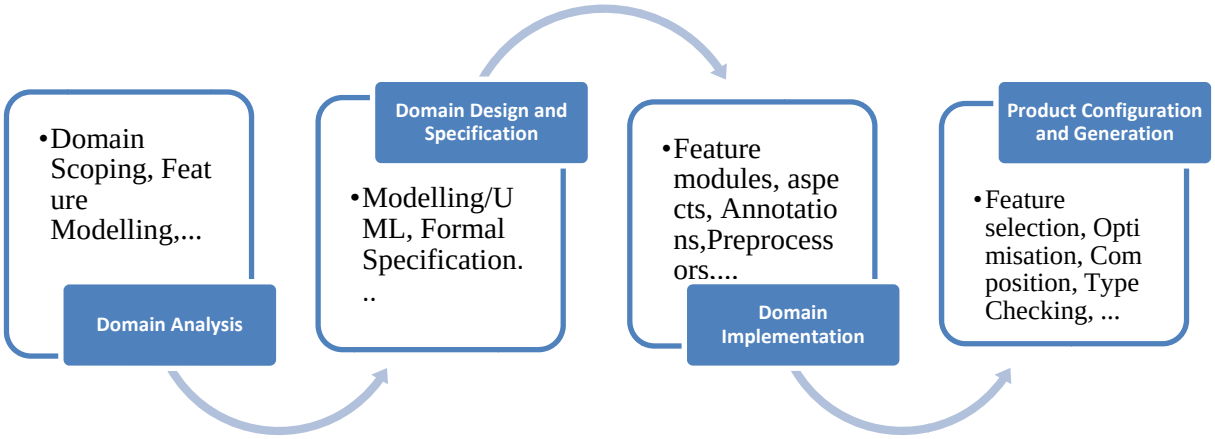

Fig. 3. Phases of FOSD Process

1) Domain Analysis:

A key success of FOSD is to select a well defined, well focused and well scoped domain. Czarnecki K and, Eisenecker[14] defines domain as " An area of knowledge that is scoped to maximize the satisfaction of the requirements of its stakeholders, includes a set of concepts and terminology understood by practitioners in that area and includes the knowledge of how to build software systems in that area." Commonality and variability from a domain has to be identified and understood well for the development of reusable core assets for a software product line. Domain analysis is the requirements engineering phase for a feature oriented product line.

In FOSD, Domain scoping and feature modeling are the main processes of domain analysis. In domain scoping the description of desired features or specific product variations which the software product line should be supported is identified and recorded. Domain experts have to study and analyze the targeted domain and extract useful information from that domain for effective and efficient domain scoping. In domain modeling the captured information in the scoping process is modeled and documented in terms of commonality and variability. The feature groups such as mandatory, optional, alternative has to be identified and recorded carefully.

2) Domain design and specification:

In this phase, from the documents of variability feature model using feature diagram is developed. A feature model represents and documented the features and their relationships in a software product line. Andreas Classen and co-scholars [15] defines "A Feature Diagram is a hierarchical diagram that visually depicts the features of a Domain in groups of increasing levels of detail". The great way to summarize the features of a domain in visual manner is by using feature trees.

Swen Apel [16] defines "A feature diagram is a graphical representation of a feature model as a tree over the feature set F. Each edge in the tree is defined by exactly one feature constraint, that is, by a declaration of one of the feature constraint types mandatory, optional, alternative, or or". The set of cross-tree constraints (requires, excludes) may also be defined additionally by make use of feature diagram. The corresponding propositional formula of a feature diagram can be generated by conjoining the feature constraints and the cross-tree constraints. This will represent the semantics of the whole feature model.

3) Domain Implementation:

In this phase, the identified and modeled features of the domain are implemented in the form of reusable artifacts. There are mainly two steps for domain implementation. Firstly, suitable implementation strategy has to be selected according to the domain. There are a lot of strategies are there, which may be language based (Framework, AOP, FOP, etc) or tool based (build systems, Preprocessors, etc). Secondly, according to the strategy selected, prepare the design and code to hook implementations.

4) Product Configuration and generation:

It is the final phase of the FOSD. The main activities of this phase are Feature selection, Product derivation, Product validation and verification. The essential step of individual product customization in product line engineering is feature selection, in which the multiple features, that may exhibit competing and conflicting interaction, have to be considered in a well defined manner. Product derivation can do automatically and manually. In manual task, glue code is written by developers and thus the artifacts corresponding to the selected features is connected and the gaps are patched for the customized product derivation. Where as in automatic task a push down approach is used to derive the product, the stake holders select the features and a compositional engine is called to combine the corresponding artifacts into an executable software product. Product validation is needed for the customized product which is derived in either way by unit testing mechanisms.

In the area of FOSD, the researchers and practitioners were mostly concerned with Feature implementation, Refactoring of software product lines, Feature interactions, Analysis of software product line, and Tool support. Our aim is to classify and summarize the works and searching for finding the scope of future research work in FOSD.

From the current literature, it is clear that the existing frameworks for Feature Oriented Software Product Line development (FeatureIDE, pure::variants, BigLever) are automatic except for the processes of domain scoping and feature selection. Human involvement is necessary for those two phases. During Domain scoping, domain experts collect information about the targeted domain through requirement analysis techniques of software engineering. Feature selection is the process of selecting the required features from the set of features of the product line by the stake holder according to their requirement. So in both requirement analysis of domain and the human involvement is needed. The requirement engineering in large scale software development process can result in massive amounts of noisy and semi structured data that must be analyzed and distilled in order to extract useful requirements.

B. Results: RQ2

Our second review question is "Which will be the best approach for implementing variability in FOSD?" In order to deal with this question a number of terms have to be explored.

- What is variability?
- What is the classification of implementation strategies?
- What are the quality criteria for evaluating the implementation strategies?

Variability is the ability to change or customize a system. Software product variants are quite similar, but typically differ in new or additional features. Improving variability in a system implies making it easier to do certain kinds of changes. The goals of FOSD, such as automated software derivation, efficiently facilitating reuse, and variation of features in a software product lines, can be achieved only by the systematic implementation of features. There are around 12 implementation strategies are studied and classified according to binding time, technology and representations used in those strategies.

Table IX. Implementation Strategies Classification[16]

Classic/Advanced & Implementation Strategies	**Representation**		**Binding Time**		**Technology**	
	Annotation	**Composition**	**Compile time**	**Load time**	**Language based**	**Tool based**
Classic-Parameters	•			•	•	
Classic-Design Patterns		•	•	•	•	
Classic-Frameworks		•	•	•	•	
Classic-Components		•	•	•	•	•
Classic-Version control		•	•			•
Classic-Build Systems		•	•			•
Classic-Preprocessors	•		•			•
Advanced-Feature Oriented Programming		•	•	•	•	
Advanced-Delta Oriented Programming		•	•	•	•	
Advanced-Context Oriented Programming		•	•	•	•	
Advanced-Aspect Oriented Programming		•	•	•	•	
Advanced-Virtual Separation of concerns	•		•			•

Swen Apel [16] proposed six quality criteria for evaluating these 12 software product-line implementation techniques. Implementation strategies and the specific quality criteria is shown in table X.

Table X. Quality Criteria for Implementation Strategy [16]

Implementation Strategies	Quality Criteria					
	Low preplanning effort	**Feature traceability**	**Separation of concerns**	**Information hiding**	**Granularity**	**Uniformity**
Feature Oriented Programming	•	•	•		•	•
Delta Oriented Programming	•	•	•	•	•	•
Context Oriented Programming	•	•	•		•	•
Aspect Oriented Programming	•	•	•		•	•
Virtual Separation of concerns	•	•	•		•	•
Parameters	•		•		•	•
Design Patterns		•	•	•		•
Frameworks		•	•	•		•
Components		•		•		•
Version control	•				•	•
Build Systems	•					•
Preprocessors	•	•				•

C. Results: RQ3

Our third question deals with features and their interactions and corresponding problems. Features have to be interacted to achieve the requirement and predefined functionality of the software product, by exchanging information. This interaction helps to refine and utilize other feature behavior and thus accomplish a specific task by cooperation of the features. But when one feature influences the behavior of other features in an unintended way, it is call inadvertent feature interaction problem or simply feature interaction problem. These feature interaction problems have to be detected, managed and resolved to attain the specific goals of the interactions between the features.

An example of unintended feature interaction problem is between the call waiting and forwarding features of a telephony system [17]; when both features are activated simultaneously, the system will collapsed and reach in an unfavorable state while it receives a call on a busy line. Reisner et al. [18] reveals that higher order interactions of features are also exists in practice. In this more than two features are interacted and interconnected and thus it is difficult to find unfavorable conditions even though it is a rare case.

For solving feature interaction problem first we have to identify the feature interactions. Now in every product line engineering features are developed independently this will make identifying and detecting interactions among features is a challenging task. Proper and systematic requirement engineering in the domain analysis is the only solution to detect feature interactions [19, 20]. Heymans [21] proposed "formal methods can be successfully applied on core models of software product line, but scale them to be able to analyze source code instead of requirements models or manually abstracted models remains an open problem in feature interaction detection."

Apel et al.[16] defines " Optional feature problem is the mismatch between intended variability and the actual variability provided by the implementation, due to coordinate code. It occurs when two or more optional features interact and the presence of coordination code reduces the intended variability of the product line." The implementation strategies [16] are

- Change in feature model: "Instead of a proper implementation, exclude problematic feature combinations from the feature model".
- Multiple Implementations: "To account for configurations with and without coordination code, features are implemented separately for each combination".
- Moving Code: "Coordination code is moved to one of the interacting features or to a shared required feature".
- Conditional Compilation: "Using a preprocessor, the coordination code annotated and only complied if both features are present".

- Optional Weaving: "Coordination code is implemented as implicitly optional, using mechanisms inspired by aspect weaving".
- Distinct module for coordination code: "A distinct module separates coordination code from feature code; the module is automatically included when both features are included".

Apel et.al [16] compares these feature interaction strategies based on the quality criteria such as code quality, binary size and performance, variability, and implementation effort.

Table XI. Implementation Approaches for the Feature Interaction Problem

Implementation Strategy	**Quality Criteria**			
	Code quality	**Binary size and performance**	**Variability**	**Implementation Effort**
Change in feature model	•	•		•
Multiple Implementations:		•	•	
Moving Code	•		•	•
Conditional Compilation		•	•	•
Optional Weaving	?	?	•	•
Distinct module for coordination code	•	•	•	

We are not able to find a generally preferable strategy for feature interaction problem solving. The multiple implementation strategy is not a good idea. Depending on the need and domain characteristics the developer has to select one among the remaining. Optional weaving is an emerging technique for this purpose.

D. Results: RQ4

Our fourth question deals with refactoring techniques. Refactoring is an important activity in software development, maintenance, and evolution. Apel et al[16] says the origin of refactoring as " Refactoring are typically traced to the dissertation of Opdyke [22] and have been popularized by the seminal book of Fowler [23]. Fowler [23] defines refactoring as "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure".

Refactoring technique is used to eliminate code smells and thus enhancing program comprehension. The well known code smells in software product line such as long methods, large classes, and duplicate codes can be eliminated by applying traditional Object Oriented refactoring techniques. Alves et al [24] extends the traditional refactoring techniques for feature oriented software product lines. In their technique they first focused on the changes to feature model and later the implementation artifacts. Lastly problem to solution space is also being considered for refactoring.

The main focus of product line refactoring is on FOSD paradigm. It may affect the feature model and domain artifacts of Feature oriented software product line. The different types of refactoring techniques proposed by different authors for FOSD is summarized and defined in Table XII.

Table XII. Refactoring Techniques and Examples

Author/Introducer	Refactoring technique	Definition	Examples
Schulz et al.[25]	variability preserving refactoring	"A product line refactoring that does not change the set of valid products and corresponding feature selections and preserves the observable behavior of all products."	• Move feature • Rename feature • Extract shared feature • Change binding time
Schulz[S26]	variability enhancing refactoring	"A product line refactoring that does not change the set of valid products and corresponding feature selections and preserves the observable behavior of all products. It may introduce additional products"	• Extract feature
Thum et al.[27]	Product preserving refactoring	"A product line refactoring that does not change the set of valid products and corresponding feature selections and preserves the observable behavior of all products. It may add and remove products outside the given set"	• Delete feature • Merge feature

The concept of feature refactoring and extraction is introduced by Liu et al [28],based on formal algebraic model to deal with interacting and overlapping features. The refactoring model developed by Kastner et el[29] deal with how the transformation between annotation abd composition based feature implementations take place, with the help of refactorings. Apel et al.[30] introduces pseudo commutatively , suggests a better refactoring is possible in AspectJ.

Kastner et al[31] reported the experience with feature extraction in Berkeley DB using AspectJ and granularity implications of feature implementations with Berkeley DB. Rosenmuller et al.[32] refactored the C version of Berkeley DB using FeatureC++.

E. Results: RQ5

Our fifth review question is concerned with the requirement engineering process for application domain in order to identify the features and their relationships to satisfy the requirement of every stakeholders of that domain. The requirement engineering in software product line development has to deal with large volume of semi structured and unstructured data, so proper requirement engineering is needed to extract useful information from these noisy data. Human intensive tasks such as requirement elicitation, analysis and management are the base of feature extraction from the application domain of the software product line. Domain scoping and analysis plays a major role in feature extraction.

Like every other software development technique, requirement engineering is a manual task by analyzing the application domain is needed in FOSD paradigm too. In FOSD experts analyze the SRS document and derive the features corresponding to each requirement in the domain and find the relationship among them and represent them in a feature model. "A feature diagram is a graphical AND/OR hierarchy of features, captures structural or conceptual relationships among features". The feature diagram is a structured tree diagram based on specialization generalization concept. *The mandatory"*, *"optional"*, *"alternative"* and *"or"* feature groups can easily represented using feature models with tree structure in a way that nodes represent features and the arcs represent variability. To specify which feature is needed in a variant, the following rules in Table 13 are applied. If selecting a parent feature in a variant.

Table XIII. Rules for Selecting a Feature from a Feature Group

Group	Features included	Rule expression
Mandatory	All its *mandatory* child features	n from n
Optional	Any number of *optional* features	m from n, 0 < = m<=n
Alternative	Exactly one feature must be selected	1 from n
Or	At least one feature must be selected	m from n, m>1

The representation of four different feature groups are shown in fig. 4.

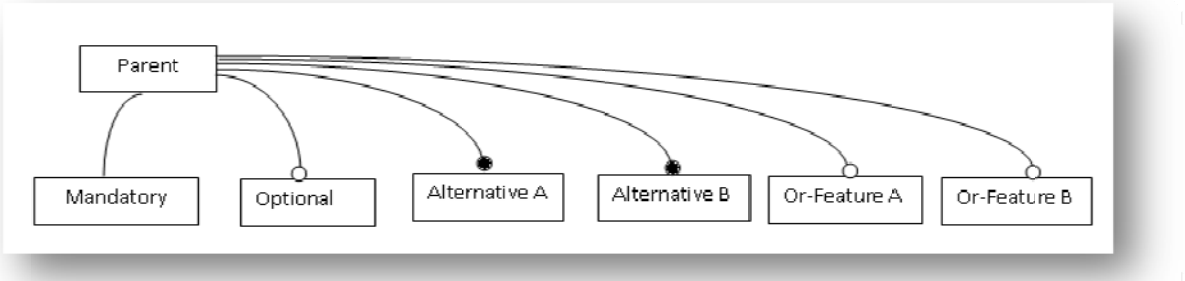

Fig. 4. Notation of feature models

There is no standard procedure is for applying the above mentioned rules but the graphical representation of FODA is commonly used for feature oriented domain analysis and modeling. Kang et al. [2] proposed Feature-Oriented Domain Analysis [FODA] to identify and model features. The focus of FODA is on domain analysis. The processes involved in this are i) analyzing of the domain of the product line, ii) analyzing the features of the product line , and iii) modeling the features of the product line. The three major phases in FODA which guides the success of the process are i) Context analysis ii) Domain modeling iii) Architecture modeling.

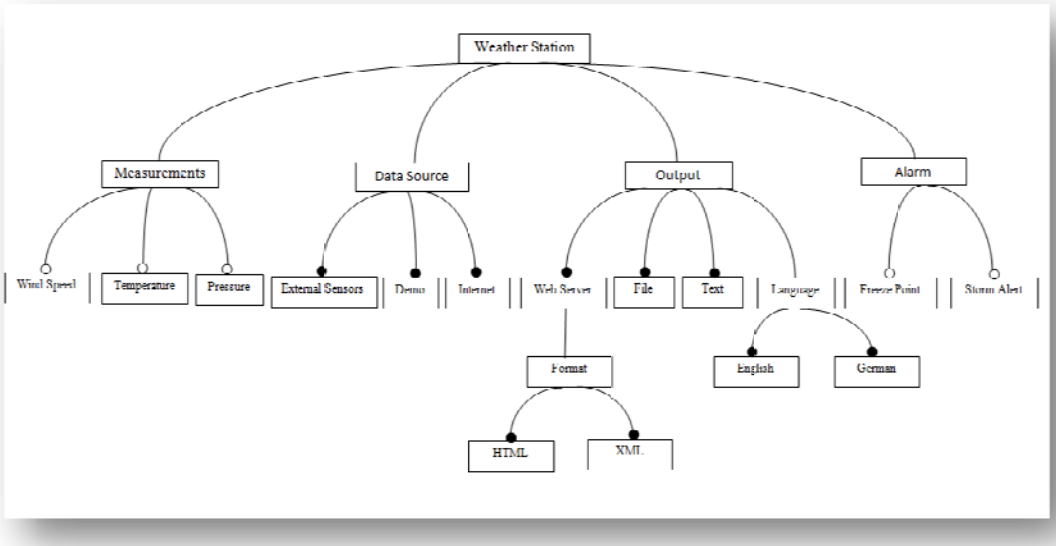

Fig. 5. Example feature model

Feature extraction is not at all an automated process it requires human interaction for domain analysis and requirement engineering.

F. Results: RQ6

Defect prediction models can reduce QA costs. Many studies investigate such models at the file [33] or code-change (commit) level [34] and show that they can achieve satisfactory results. However, it is still unknown if such models can be used at the feature level. Rahman et al. [33] investigate the efficacy of code and process metrics for defect prediction in a large number of releases from many systems. They compare both kinds of metrics to understand when and why each kind may be suitable for a prediction model, suggesting that process metrics are preferable. We follow this advice.

Tan et al. [S34] investigate defect prediction based on change classification (commit level). They perform a case study on a proprietary Cisco system and analyze the reasons for any low performance of the method. Among others, to address the problem of imbalanced data, they add a gap between training and testing set to allow more bugs to be detected. Although a better performance is achieved, the precision is still low, requiring further investigations. Wang et al. [35] propose using deep learning to learn semantic representations of programs from source code. The goal is to go beyond traditional attributes and use semantic information for characterizing defects and improving defect prediction in within- and cross-project defect predictions. They achieve an average F-measure of 0.641 on 13 systems. Jeon et al. [36] propose a defect-prediction model for consecutive software products in an SPL. They study historical trends in bug-tracking systems to predict the number of defects in upcoming products, but not for individual features.

Rodrigo Queiroz et al.[37] propose a defect prediction model to identify defective features using machine learning techniques, aiming at improving the cost-effectiveness of QA activities for features in SPLs. If effective, feature defect predictions could (i) help developers to select (or prioritize) samples of features that are prone to defects and should be tested more thoroughly; (ii) increase the detection of defects scattered across implementation artifacts, as these will likely have a low impact on the prediction model if using traditional file-based prediction; and (iii) improve the actual QA (e.g., code reviews) when features are used for communicating and coordinating within and across teams.

IV. DISCUSSION

A. Dimensions

In this study we found that almost every paper is published in ACM and IEEE. FOSD community is conducting a continuous attempt to empower FOSD in every year from 2009 to now. They are dealing with every aspects of FOSD. The major areas of FOSD research are

- Programming language and tool support for FOSD
- Domain engineering and/or application engineering
- Software product lines and program families
- Feature and variation modeling
- Formal methods and theory for FOSD
- Type systems and formal semantics of FOSD languages
- Feature composition, interaction, and refactoring
- Multi-dimensional separation of concerns and aspect-oriented software development
- Generative programming and automatic programming
- Design patterns, frameworks, and components
- Model-driven development and service-oriented architecture
- Variability-aware analysis (e.g., type checking, testing, data flow analysis, and verification)
- Versioning, evolution, and maintenance
- Components, services, and models
- Variability-aware analysis
- Feature interaction, modeling, composition, and refactoring
- Versioning, evolution, and maintenance
- Components, services, and models
- Build systems and feature-to-code mappings
- Program comprehension
- Empirical studies of all these topics

B. Principal Findings

Our review is only focused on the review questions; we didn't research further to find the other dimensions of the research. Our first question deals with automation of software development using FOSD paradigm. It is found that information processing by humans is going only on feature analysis and extraction phase of domain engineering and feature selection phase of application engineering. Our second question deals with, various variability implementation techniques for implementing FOSD. The main focus of FOSD is commonality, variability and reuse. In our study we found 12 implementation techniques and their strength and weakness based on the quality criteria. Feature interaction and optional feature problem are the most concerned problems when dealing with features and their relationships. The techniques for solving these also considered and answered with third review question. The scope and use of refactoring in FOSD is also discussed in this review. Our final question deals with feature extraction from an application domain. We found that the future scope of research must deal with this area because it is still based on the contemporary requirement engineering process of software engineering and thus expert knowledge is needed.

C. Threats to Validity

We feel that our review study have the following threats to validity.

Table XIV. Classification of Validity Threats

Validity Threats	Description
Construct validity	This is primarily related with obtaining the relevant information by defining the research scope. At this stage, the biggest challenge is to decide the inclusion and exclusion criteria for selecting the papers. To address this issue, we considered all the systematic review reports related to software engineering.
External validity	The findings of this review cannot be generalized because the results are based on a specific set of keywords and the research repositories that have been used for the data collection. Therefore, our results could be limited and cannot be applied to every organizational setup.
Results validity	Our results deal with only the review questions and the selected papers only. It provides the future research focus only on the selected area.
Internal validity	We only searched in online libraries and further snowballing is also performed.
Conclusion validity	Our study focused with review questions and to ensure the correctness of the extracted data, the protocol was developed to define the data extraction strategy and format.

V. CONCLUSION AND FUTURE WORK

In this paper we reported a systematic literature review in the field of FOSD. The purpose of this study is to find the state of the art on this software development paradigm. For this we designed a review protocol, which covers 63 papers selected from 118 available research papers on the topic. Our focus is to answer the review questions and thus cover the sub areas of FOSD. The first question deals with the automation achieved through different phases of FOSD. Although software development automation is the main aim of FOSD, we couldn't able to say it is fully automated now. Human interaction and expertise is needed in both domain and application engineering in the form of feature extraction requirement engineering and feature selection requirement engineering. Here it is clear that full automation is still an open problem for researchers of this area.

The second and third question deals with variability implementation and feature interaction problems respectively. Many techniques are available for these purposes even though there is no general preferable standard technique for that and each strategy is on an emerging stage. Feature oriented Programming is good but has little experience in practice only used as an academic tool. The developers can choose a strategy according to their expertise and domain. The fourth question deals with the uses of refactoring and feature in this field. Actually refactoring techniques of Object Oriented paradigm is extended to feature oriented software product line development for overcoming the pitfalls.

Based on our fifth question some practical difficulty of domain requirement engineering, feature extraction and modeling are also discussed. The sixth question is concerned with the effective use of feature defect prediction technique for the betterment of FOSD because it is based on features. As mentioned in each section, all areas of software development are not fully explored by the researchers yet. If someone approaching FOSD with a research focus the scope is very huge because it is an emerging technique. Each area reveals a research focus for the researchers. With proper research and development FOSD has the capability to change total concept of software engineering.

ACKNOWLEDGEMENT

This Systematic Review is performed as an initiation to the Ph.D work on the topic FOSD. There is no funding source(s) involved in this research.

APPENDICES

A. Selected Studies

[S1] Sven Apel, Christian K¨astner , An Overview of Feature-Oriented Software development, Journal of Object Technology, Vol. 8, No. 5, 2009

[S2] Czarnecki K., Eisenecker U, Generative programming: methods, Tools and applications, ACM Press/Addison-Wesly, 2010

[S3] Andreas Classen, Patrick Heymans, Robin Laney, Bashar Nuseibeh, Thein Than Tun. On the Structure of Problem Variability – From Feature Diagrams to Problem Frames, 2007

[S4] Sven Apel, Don Batory, Cristian Kastener, Gunter Saake, Feature Oriented Software Product Lines: Concepts and Implementation. Springer 2013

[S5] M. Calder, M. Kolberg, E. Magill, and S. Reiff-Marganiec, Feature interaction: A critical review and considered forecast. Computer Networks, 41(1):115–141, 2003

[S6] Reisner E, Song C, Ma K-K, Foster JS, Porter A., Using symbolic evaluation to understand behavior in configurable software systems, In: Proc. Int'l Conf. Software Engineering (ICSE), ACM Press, pp 445–454, 2010

[S7] Calder M, Kolberg M, Magill EH, Reiff-Marganiec S, Feature interaction: A critical review and considered forecast. Comput Netw 41(1):115–141, 2003

[S8] Nhlabatsi A, Laney R, Nuseibeh B , Feature interaction: The security threat from within software systems. Prog inform 5:75–89, 2008

[S9] Heymans P , Formal methods for the masses. In: Proc. Int'l Software Product Line Conference (SPLC), ACM Press, p 4, 2012

[S10] Kästner C, Apel S, Rahman SS, Rosenmüller M, Batory D, Saake G , On the impact of the optional feature problem: Analysis and case studies. In: Proc. Int'l Software Product Line Conference (SPLC), ACM Press, pp 181–190, 2009

[S11] Opdyke WF., Refactoring object-oriented frameworks. Ph.D. thesis, University of Illinois at Urbana-Champaign, 1992

[S12] Fowler M Refactoring: Improving the design of existing code. Addison-Wesley, 1999

[S13] Alves V, Gheyi R, Massoni T, Kulesza U, Borba P, Lucena C ,Refactoring product lines. In: Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE). ACM Press, pp 201–210, 2006

[S14] Schulze S, Thüm T, Kuhlemann M, Saake G, Variant-preserving refactoring in feature oriented software product lines. In: Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems (VaMoS), ACM Press, pp 73–81, 2012

[S15] Schulze S, Analysis and removal of code clones in software product lines, Ph.D. thesis, School of Computer Science, University of Magdeburg, 2013

[S16] Savolainen J, Bosch J, Kuusela J, Männistö T, Default values for improved product line management, In: Proc. Int'l Software Product Line Conference (SPLC), Carnegie MellonUniversity, pp 51–60, 2009

[S17] Thüm T, Batory D, Kästner C, Reasoning about edits to feature models. In: Proc. Int'l Conf. Software Engineering (ICSE). IEEE Computer Society, pp 254–264, 2009

[S18] Liu J, Batory D, Lengauer C, Feature oriented refactoring of legacy applications. In: Proc. Int'l Conf. Software Engineering (ICSE), ACM Press, pp 112–121, 2006

[S19] Siegmund N, Kästner C, Rosenmüller M, Heidenreich F, Apel S, Saake G ,Bridging the gap between variability in client application and database schema. In: Proc. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW). Lecture Notes in Informatics,vol. P-144. Gesellschaft für Informatik (GI), pp 297–306, 2009a

[S20] Kuhlemann M, Batory D, Apel S, Refactoring feature modules. In: Proceedings of the International Conference on Software Reuse (ICSR), Springer, pp 106–115, 2009a

[S21] Kästner C, Aspect-oriented refactoring of Berkeley DB. Master's thesis, School of Computer Science, University of Magdeburg, 2007

[S22] Rosenmüller M, Apel S, Leich T, Saake G, Tailor-made data management for embedded systems: A case study on Berkeley DB. Data Knowl Eng (DKE) 68(12):1493–1512, 2009a

[S23] Reisner E, Song C, Ma K-K, Foster JS, Porter A Using symbolic evaluation to understand behavior in configurable software systems, ACM Press, pp 445-454,2010

[S24] Rabkin A, Katz R, Static extraction of program configuration options. In: Proc. Int'l Conf. Software Engineering (ICSE), IEEE Computer Society, pp 131–140, 2011

[S25] Muthig D, Patzke T, Generic implementation of product line components, In: Proc. Int'l Conf. Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays), Lecture Notes in Computer Science, vol. 2591. Springer, pp 313–329, 2002

[S26] Czarnecki K, Eisenecker U, Generative programming: Methods, tools, and applications ACM Press/Addison-Wesley, 2000

[S27] Gamma E, Beck K, Contributing to eclipse: Principles, patterns, and plug-in. Wesley, 2003

[S28] Staples M, Hill D, Experiences adopting software product line development without a product line architecture, In: Proc. Asia-Pacific Software Engineering Conf. (APSEC). IEEE Computer Society, pp 176–183, 2004

[S29] Adams B, De Schutter K, Tromp H, De Meuter W, Design recovery and maintenance of build systems. In: Proc. Int'l Conf. Software Maintenance (ICSM). IEEE Computer Society, pp 114–123,2007

[S30] Adams B, De Schutter K, Tromp H, De Meuter W, The evolution of the Linux build system. Electronic Communications of the EASST, 8,2008a

[S31] Dietrich C, Tartler R, Schröder-Preikschat W, Lohmann D, A robust approach for variability extraction from the Linux build system. In: Proc. Int'l Software Product Line Conference (SPLC). ACM Press, pp 21–30, 2012a

[S32] Lohmann D, Scheler F, Tartler R, Spinczyk O, Schröder-Preikschat W , A quantitative analysis of aspects in the eCos kernel. In: Proc. Int'l EuroSys Conference (EuroSys). ACM Press, pp 191–204,2006a
[S33] M, Badros G, Notkin D, An empirical analysis of C preprocessor use. IEEE Trans Softw Eng (TSE) 28(12):1146–1170, 2002
[S34] Liebig J, Kästner C, Apel S, Analyzing the discipline of preprocessor annotations in 30million lines of C code, In: Proc. Int'l Conf. Aspect-Oriented Software Development (AOSD). ACM Press, pp 191–202, 2011
[S35] Apel S, Kästner C, Lengauer C, Language-independent and automated software composition: The FeatureHouse experience, IEEE Trans Software Eng (TSE) 39(1):63–79, 2013a
[S36] Lopez-Herrejon R, Batory D, Cook W. Evaluating support for features in advanced modularization technologies. In: Proc. Int'l Conf. Generative and Component-Based Software Engineering (ECOOP). Lecture notes in computer science, vol. 3586. Springer, pp 169–194, 2005
[S37] Kästner C, Apel S, Thüm T, Saake G, Type checking annotation-based product lines.ACM Trans Softw Eng Methodol (TOSEM) 21(3):14:1–14:39,2012a
[S38] Hunleth F, Cytron RK, Footprint and feature management using aspect-oriented programming techniques, In: Proc. Conf. Languages, Compilers and Tools For Embedded systems (LCTES), ACM Press, pp 38–45, 2002
[S39] Apel S, Leich T, Saake G, Aspectual feature modules, IEEE Trans Software Eng (TSE)34(2):162–180, 2008b
[S40] Sato Y, Chiba S, Tatsubori M, A selective, just-in-time aspect weaver, In: Proc. Int'l conf. Generative Programming and Component Engineering (GPCE). Lecture Notes in Computer Science, vol. 2830. Springer, pp 189–208, 2003
[S41] Popovici A, Alonso G, Gross T, Just-in-time aspects: Efficient dynamic weaving for Java. In: Proc. Int'l Conf. Aspect-Oriented Software Development (AOSD), ACM Press, pp 100–109, 2003
[S42] Lopez-Herrejon R, Understanding feature modularity. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin, 2006
[S43] Lafferty D, Cahill V, Language-independent aspect-oriented programming, In. Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM Press, pp 1–12, 2003
[S44] Boxleitner S, Apel S, Kästner C, Language-independent quantification and weaving for feature composition. In: Proc. Int'l Symp. Software Composition (SC). Lecture Notes in Computer Science, vol. 5634. Springer, pp 45–54, 2009
[S45] Filman R, Friedman D, Aspect-oriented programming is quantification and obliviousness, In: Aspect-Oriented Software Development, Addison-Wesley, pp 21–35, 2005
[S59] Lieberherr KJ, Lorenz DH, Ovlinger J, Aspectual collaborations—Combining modules and aspects. Comput J 46(5):542–565, 2003
[S46] Aldrich J, Open modules: Modular reasoning about advice, In: Proc. Europ. Conf. Object Oriented Programming (ECOOP), Lecture Notes in Computer Science, vol. 3586. Springer, pp 144–168, 2005
[S47] Sullivan K, Griswold W, Song Y, Cai Y, Shonle M, Tewari N, Rajan H , Information hiding interfaces for aspect-oriented design, In: Proc. Int'l Symp. Foundations of Software Engineering (FSE), ACM Press, pp 166–175, 2005
[S48] Dantas D, Walker D, Harmless advice. In: Proc. Int'l Symp. Principles of Programming Languages (POPL), ACM Press, pp 383–396, 2006
[S49] Steimann F, Pawlitzki T, The paradoxical success of aspect-oriented programming. In: Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM Press, pp 481–497, 2006
[S50] Störzer M, Koppen C, PCDiff: Attacking the fragile pointcut problem, abstract, In: European Interactive Workshop on Aspects in Software, 2004
[S51] Steimann F, Pawlitzki T, Apel S, Kästner C, Types and modularity for implicit invocation with implicit announcement, ACM Trans Software Eng Methodol (TOSEM) 20(1):1:1–1:43, 2010
[S52] Schaefer I, Bettini L, Damiani F, Tanzarella N, Delta-oriented programming of software product lines, In: Proc. Int'l Software Product Line Conference (SPLC), Springer, pp 77–91, 2010
[S53] Kuhlemann M, Batory D, Apel S, Refactoring feature modules. In: Proceedings of the International Conference on Software Reuse (ICSR), Springer, pp 106–115, 2009a
[S54] Hirschfeld R, Costanza P, Nierstrasz O, Context-oriented programming, J Object Technology (JOT) 7(3):125–151, 2008
[S55] Rosenmüller M, Siegmund N, Apel S, Saake G, Flexible feature binding in software product lines. Autom. Software Eng 18(2):163–197, 2011
[S56] Appeltauer M, Hirschfeld R, Haupt M, Masuhara H, ContextJ: Context-oriented programming with Java, Comput Softw 28(1), 272–292, 2011

[S57] Kästner C, Virtual separation of concerns. Ph.D. thesis, School of Computer Science, University of Magdeburg, 2010
[S58] F. Rahman and P. Devanbu. How, and Why, Process Metrics Are Better. In ICSE, 2013
[S59] M. Tan, L. Tan, S. Dara, and C. Mayeux Online Defect Prediction for Imbalanced Data, In ICSE, 2015.
[S60] S. Wang, T. Liu, and L. Tan, Automatically Learning Semantic Features for Defect Prediction, In ICSE, 2016.
[S61] C. Jeon, C. Byun, N. Kim, and H., In. An entropy based method for defect prediction in software product lines. International Journal of Multimedia and Ubiquitous Engineering, 9(3):375–377, 2014.
[S62] Rodrigo Queiroz, Thorsten Berger, Krzysztof Czarnecki, Towards Predicting Feature Defects in Software Product Lines, ACM 2016
[S63] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90- TR-21, Software Engineering Institute, Carnegie Mellon University, 1990

B. References:

[1] Sven Apel, Christian K¨astner , An Overview of Feature-Oriented Software Development, Journal of Object Technology, Vol. 8, No. 5, 2009
[2] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90- TR-21, Software Engineering Institute, Carnegie Mellon University, 1990
[3] K. Kang, S. Kim, J. Lee, K. Kim, G. Kim, E. Shin, FORM: A Feature Oriented Reuse Method with Domain-Specific Reference Architectures, Annals of Software Engineering, 5(1):143–168, 1998
[4] K. Czarnecki, U. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000.
[5] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, ACM Press / Addison-Wesley, 2000.
[6] K. Chen, W. Zhang, H. Zhao, H. Mei, An Approach to Constructing Feature Models Based on Requirements Clustering, In Proceedings of the International Conference on Requirements Engineering (RE), pages 31–40. IEEE CS Press, 2005
[7] D. Batory, J. Sarvela, A. Rauschmayer, Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering (TSE), 30(6), 355–371, 2004.
[8] A. Classen, P. Heymans, P. Schobbens, What's in a Feature: A Requirements Engineering Perspective, In Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), volume 4961 of Lecture Notes in Computer Science, pages 16–30. Springer, 2008.
[9] P. Zave, An Experiment in Feature Engineering. In Programming Methodology, pages 353–377. Springer-Verlag, 2003.
[10] D. Batory, Feature Models, Grammars, and Propositional Formulas, In Proceedings of the International Software Product Line Conference (SPLC), volume 3714 of Lecture Notes in Computer Science, pages 7–20. Springer-Verlag, 2005.
[11] S. Apel, C. Lengauer, B. M¨oller, C. K¨astner, An Algebra for Features and Feature Composition. In Proceedings of the International Conference on Algebraic Methodology and Software Technology (AMAST), volume 5140 of Lecture Notes in Computer Science, pages 36–50. Springer-Verlag, 2008
[12] B. Kitchenham, Procedures for performing systematic reviews, Keele, UK, Keele University, vol. 33, no. 2004, pp. 1-26, 2004.
[13] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering–a systematic literature review, Information and software technology, vol. 51, no. 1, pp. 7-15, 2009.
[14] Czarnecki K., Eisenecker U, Generative programming: methods, Tools and applications, ACM Press/Addison-Wesly, 2010
[15] Andreas Classen, Patrick Heymans, Robin Laney, Bashar Nuseibeh, Thein Than Tun. On the Structure of Problem Variability – From Feature Diagrams to Problem Frames, 2007
[16] Sven Apel, Don Batory, Cristian Kastener, Gunter Saake, Feature Oriented Software Product Lines: Concepts and Implementation, Springer 2013
[17] M. Calder, M. Kolberg, E. Magill, S. Reiff-Marganiec, Feature interaction: A critical review and considered forecast. Computer Networks, 41(1):115–141, 2003
[18] Reisner E, Song C, Ma K-K, Foster JS, Porter A, Using symbolic evaluation to understand behavior in configurable software systems. In: Proc. Int'l Conf. Software Engineering (ICSE), ACM Press, pp 445–454, 2010
[19] Calder M, Kolberg M, Magill EH, Reiff-Marganiec S , Feature interaction, A critical review and considered forecast, Comput Netw 41(1):115–141, 2003
[20] Nhlabatsi A, Laney R, Nuseibeh B , Feature interaction: The security threat from within software systems, Prog inform 5:75–89, 2008
[21] Heymans P, Formal methods for the masses. In: Proc. Int'l Software Product Line Conference (SPLC), ACM Press, p 4, 2012
[22] Opdyke WF, Refactoring object-oriented frameworks. Ph.D. thesis, University of Illinois at Urbana-Champaign, 1992
[23] Fowler M, Refactoring: Improving the design of existing code. Addison-Wesley, 1999
[24] Alves V, Gheyi R, Massoni T, Kulesza U, Borba P, Lucena C, Refactoring product lines, In: Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE). ACM Press, pp 201–210, 2006
[25] Schulze S, Thüm T, Kuhlemann M, Saake G , Variant-preserving refactoring in feature oriented software product lines, In: Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems (VaMoS), ACM Press, pp 73–81, 2012
[26] Schulze S, Analysis and removal of code clones in software product lines. Ph.D. thesis, School of Computer Science, University of Magdeburg, 2013
[27] Thüm T, Batory D, Kästner C, Reasoning about edits to feature models, In: Proc. Int'l Conf. Software Engineering (ICSE), IEEE Computer Society, pp 254–264, 2009
[28] Liu J, Batory D, Lengauer C , Feature oriented refactoring of legacy applications, In: Proc. Int'l Conf. Software Engineering (ICSE), ACM Press, pp 112–121, 2006
[29] Siegmund N, Kästner C, Rosenmüller M, Heidenreich F, Apel S, Saake G ,Bridging the gap between variability in client application and database schema. In: Proc. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW), Lecture Notes in Informatics, vol. P-144. Gesellschaft für Informatik (GI), pp 297–306, 2009a
[30] Kuhlemann M, Batory D, Apel S , Refactoring feature modules, In: Proceedings of the International Conference on Software Reuse (ICSR), Springer, pp 106–115, 2009a
[31] Kästner C, Aspect-oriented refactoring of Berkeley DB. Master's thesis, School of Computer Science, University of Magdeburg, 2007

[32] Rosenmüller M, Apel S, Leich T, Saake G , Tailor-made data management for embedded systems: A case study on Berkeley DB. Data Knowl Eng (DKE) 68(12):1493–1512, 2009a
[33] F. Rahman, P. Devanbu. How, and Why, Process Metrics Are Better. In ICSE, 2013.
[34] M. Tan, L. Tan, S. Dara, and C. Mayeux, Online Defect Prediction for Imbalanced Data. In ICSE, 2015.
[35] S. Wang, T. Liu, L. Tan, Automatically Learning Semantic Features for Defect Prediction. In ICSE, 2016
[36] C. Jeon, C. Byun, N. Kim, H. In. An entropy based method for defect prediction in software product lines. International Journal of Multimedia and Ubiquitous Engineering, 9(3):375–377, 2014.
[37] Rodrigo Queiroz, Thorsten Berger, Krzysztof Czarnecki, Towards Predicting Feature Defects in Software Product Lines, ACM 2016

AUTHOR PROFILE

Mrs Kala K. U. pursued Bachelor of Science in from University of calicut, Kerala, India in 2008 and Master of Computer Applications from University of Calicut in year 2012. She has qualified UGC NET in computer science and applications in July 2016 and currently pursuing Ph.D in Software Engineering, Department of Computer Science, Pondicherry University , India since 2016. Her main research work focuses on Feature Oriented Software Development, Data Mining and Recommendation Systems. She has 3 years of teaching experience.

Dr M. Nandhini pursued Bachelor of Science in year 1994 and Master of Science in year 1997 from Bharathidasan University, Tamilnadu, India. She has qualified UGC NET in 1998 and pursued M.Phil from Alagappa University and Ph.D from Bharathiar University, and currently working as Assistant Professor in Department of Computer Science, Pondicherry University, India since 2012.She has published more than 80 research papers in reputed international journals and conferences including IEEE and it's also available online. And also she has published 3 books and one book chapter. His main research work focuses on Soft Computing, Combinatorial Problem Optimization, Artificial Intelligence, Software Engineering, Data Mining and Recommendation Systems. She has 20 years of teaching experience and 4 years of Research Experience.