IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-5, Issue-10 E-ISSN: 2347-2693

Classification of Tools For Feature-Oriented Software Development: A Comprehensive Review

Kala K. U.[1*], M. Nandhini[2]

[1*]Department of Computer Science, School of Engineering and Technology, Pondicherry University, Puducherry, India
[2]Department of Computer Science, School of Engineering and Technology, Pondicherry University, Puducherry, India

[*]*Corresponding Author: kalaunni88@gmail.com, Tel.: +919656847815*

Available online at: www.ijcseonline.org

Received: 16/Sep/2017, Revised: 29/Sep/2017, Accepted: 15/Oct/2017, Published: 30/Oct/2017

***Abstract*—**Software development tools are the programs or set of programs which assist the software developers in the development of other programs and applications. This makes their work easier. Feature-Oriented Software Development (FOSD) Paradigm is a software development paradigm especially for the development of large-scale software systems and software product lines. Like every other software development paradigm, the complexity of the creation, debugging and maintenance of a Feature-Oriented software development necessitate the tool support. Software product-line development consists of multiple phases (Domain Analysis, Domain Design and Specification, Domain Implementation, and Product Configuration and Generation), each of which shall be supported by proper tools. There are a large number of tools available for supporting each process of FOSD, but proper classification does not exist. Tools which support one phase will not support the processes of other phases, so it is necessary to study the tools available before using it. In this review, our aim is to collect and classify the available tools based on the processes of each phase of FOSD. As the supporting tool field is too broad and continuously changing, the collective information about tools is not available. This review helps the developers and researchers to get a comprehensive idea of the existing tools and their functions, which help them to make use of this according to their need.

***Keywords*—**Tools, Integrated Development Environment, Feature-Oriented Software Development(FOSD), Software Product Line, Comprehensive Review.

I. INTRODUCTION

Feature-oriented software development (FOSD) is a paradigm for the construction, customization, and synthesis of large-scale software systems [1]. Usually, large scale software systems are developed as a software product line. A software product line is a set of different software systems that share commonalities, described by means of features [2]. A feature is a user-visible aspect or characteristic of a system [3]. In FOSD, a product of a software product line is generated automatically for a selection of features. FOSD facilitates the construction of customized software products, while artifacts can be reused and thereby time and resources can be saved.

The paradigm of feature-oriented software development is an extension of existing programming paradigms and languages, such as object-oriented programming, to create variable and customizable software. However, with this additional variability, there also some additional problems that must be considered, such as feature interactions. The intricacies of feature component composition may arise[62], Hence, efficient software development is not possible without tool support; this especially holds for implementation of software product lines. For this reason, many specialized tools that solve the problems with the implementation of feature-oriented software have been developed.

The customizable software is necessary for a broad spectrum of domains. However, just like single systems, software product lines also need support from tools for different stages of production. Because tools from single system engineering can only be applied to one product at once, these tools are not sufficient for an efficient development of all. In the last decade, several approaches have been proposed that transfer the analysis of single systems to an efficient analysis of product lines. With these strategies, there also came tools that can be applied to analyze a product line. To get a representative list of current tools, we base this survey as a classification of tools for Feature-Oriented Software Product Line Development.

We provide a categorization on tools for implementation in feature-oriented software development. With this paper, we aim to help the researchers, students, and practitioners

working in the field of FOSD by the following contributions: A brief overview of the processes of FOSD which have been supported by the tools, A detailed classification of the tools according to their process support, Scrutinized information such as programming language and implementation strategies corresponding to each tool and a brief description of the IDEs exists in the field of FOSD.

Additionally, this work gives an overview on tools that support the implementation of feature-oriented software, such as tools for product generation, refactoring, and analysis. Finally, we identify tasks in feature-oriented software development that are underrepresented in current tool support. This work eases the reuse of existing tools for researchers and students and thus simplifies research transfer to practice.

Rest of the paper is organized as follows, Section I contains the introduction of FOSD and the tools for it, Section II contains the related work of tool support for FOSD, Section III contains the details of essential phases and sub-processes of each phase of FOSD, Section IV contains the essential tools corresponding to each process, Section V explains the available Integrated Development Environments (IDE) for FOSD and Section VI concludes review work with future directions.

II. RELATED WORK

In the last decade, several type of researches have been proposed in the field of FOSD. While we are going deep into the research area, Sven Apel and Thomas Thüm are the two prominent persons who contribute much in the research on FOSD through their continuous focus on the innovation of the field. Their study concerns with all major areas of this new software development strategy. In his book, Sven Apel et al [1] gives a clear cut idea and research areas focused in this field. The researchers mainly concentrate on feature modeling, variability implementation techniques (language based and tool based), refactoring, feature interaction problems, analysis of software product lines, etc. Tools are necessary to support all these FOSD activities.

In this study, our focus is on the researches carried out in the field of tool support for FOSD. Thomus Thum et al [5] give an overview of analysis tools for software product line. However, for brevity, the survey focused only on the tools for software analysis of product lines, the other phases such as analysis of software models, variability models, product generation, etc have been excluded. Here we are trying to include all the tools related to Feature-Oriented Software Product Line Development.

The effort of building a research prototype or creating a proof of concept is greatly reduced if existing tools can be leveraged. It is also important to know which tools exist and on which other tools they are built on, which can also reduce the effort of building a new tool. With this paper, we aim to help the researchers, students, and practitioners working in the field of FOSD by the following contributions:

- A brief overview of the processes of FOSD which have been supported by the tools.
- A detailed classification of the tools according to their process support
- Scrutinized information such as programming language and implementation strategies corresponding to each tool.
- A brief description of the IDEs exists in the field of FOSD.

Our goal with this classification survey is a detailed study of the tool support to decide whether a tool for a certain analysis strategy or generation technique exists that could be used, or enhancing the research on the tools.

III. PHASES AND SUB PROCESSES OF FOSD

FOSD is the process of developing software product lines, and generate software products automatically in terms of features. Apel et al. [4] define FOSD as a four-phase process as shown in figure 1.

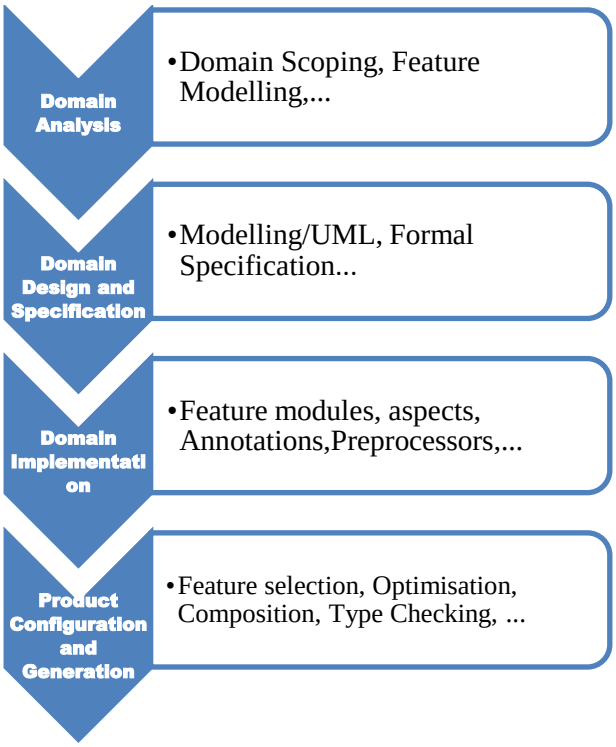

Figure 1. Phases of FOSD

In domain analysis, commonalities and differences of the domain of interest are identified, resulting in a feature model. Domain design and specification is the process in which the architecture of the product line is designed and specified. Domain implementation is the phase of designing, implementing, analyzing, and refactoring the source code of the product line. In product configuration and generation, a product containing a desired selection of features is created.

The core processes which have tool support are Feature Modeling, Feature-Oriented Decomposition, Variant Preserving Refactoring, Product Configuration and Product Generation. The processes which are used to analyse software product line such as Sampling, Testing, Type Checking, Static analysis, Model Checking, Theorem Proving, Consistency Checking, Non Functional Properties and code metrics, also avail tool support now.

Apart from the tools, Integrated Development Environment also has major role in the Feature-Oriented Software Product Line Development Paradigm.

A. Feature Modeling

Feature modeling is the most popular method for representing variabilities and commonalities in software product families. The result of a feature modeling process is a feature model. Sven Apel et al [55] defines "A feature model is a tree wherever the basis could be a feature, typically referred to because of the concept. The root feature has sub-features, and these could, in turn, produce other features as their sub-features, etc." A feature model may be a description of a system family, e.g., a software product family; an outline of an individual system comes by choosing a set of features. Timo Asikainen et al[19] groups the features as " mandatory sub features should be selected whenever its parent is chosen; an optional feature is also chosen whenever its parent is selected however needs not be selected; an alternate sub feature consists of a collection of alternatives of that precisely one should be selected whenever the parent feature is selected; an or-feature may be a sub feature kind almost like an alternate feature, with the distinction that a minimum of one amongst the alternatives should be selected.*"*

B. Feature-Oriented Decomposition

A feature may be a unit of practicality of a code that satisfies a demand, represents a design decision and provides a possible configuration choice. The essential plan of FOSD is to decompose a software system in terms of the features it provides. The goal of the decomposition is to construct well-structured software which will be tailored to the wants of the user and also the application situation. In FOSD, features structure the planning of the software system. Hence, modeling and specification activities don't aim at shaping a variable design; this is given by the decomposition into features, however at shaping the structure and behavior of features and their interactions [1].

C. Variant Preserving Refactoring

Refactorings in FOSD product lines aim at improving the structure while generally preserving observable behaviour. There are three types of refactoring are there in software product lines: Variability preserving, Variability enhancing and Product preserving refactoring. Variability preserving refactoring needs more attention in developing software product lines. It doesn't modify the set of valid product and corresponding feature selections and preserves the noticeable behaviour of all products. Also, it should not take away or add a product to a product line and should not modify the noticeable behaviour of any potential or truly delivered the product.

D. Product Configuration

A product configuration may be a description of the products delivered by a personal system. Timo Asikainen et al.[19] defines "A feature configuration, or configuration for short, is a description of the products delivered by a system; a product configuration consists of a set of features (F), the sub feature relation (s), the attribute relation (a), the type function t, and the root feature (r) that is a member of the set of features."

E. Product Generation

Product generation is the process of generating the complete product on selected features. This facilitates automation in FOSD. Several steps square measure required to get associate efficient software system from a user's feature selection. Sven Apel et al.[4] classifies product derivation as a several step process. They define the steps as "First, to help the user in choosing a group of desired features, tools got to present the offered features moreover as their constraints and relationships clearly. Succeeding step is to figure an entire, valid feature selection on the premise of a partial feature selection by evaluating attainable complete feature mixtures and judgment their appropriateness. Once we've got a correct feature selection, the specified software system is generated."

F. Sampling

In sample-based analyses, a representative subset of all products for a given coverage criterion is analyzed [1, 5]. With this approach, it is possible to efficiently detect errors by applying analysis tools from single-system engineering. Sampling is often

used in combination with testing, but can be applied to verification and other analyses as well [5, 6, 7, 8]. Sampling strategies are sound but always incomplete, since they can only detect errors that are contained in the subset that is analyzed [1]

G. Testing

Similar to testing of single systems, software product line testing aims to uncover defects, however, with the additional management of variability [9]. The main challenges of testing software product lines are reusing test cases and reducing redundancies in test cases and executions

H. Type Checking

The basis of most approaches that analyze software product lines is type safety of all products, such that each product can be compiled. Type checking is the verification process that ensures type safety [10]. Efficient type checking tools for software product lines considers the variability defined at the feature model for family-based analyses based on a unmodified product line. Type checker for annotation-based product-lines consider variability defined at annotations and at the feature model to reason about type safety. Composition-based type-checking considers the dependencies between feature modules to ensure type safety after composition.

I. Static Analysis

Static analysis operates on compile-time and can predict dynamic values or behaviours that arise at run-time [11]

J. Model Checking

In software model checking, the program is translated into a graph of states and transitions [12]. The analysis of such a graph is the verification process of model checking. Because model checkers are able to handle different values of variables, they can be used to simulate different feature selections. Thus, model checkers can be efficiently used for family-based verification of a meta-product. There are also approaches and tools beyond variability encoding, but they analyze models rather than source code [13, 5] (e.g., product-lines transition systems [14]).

K. Theorem Proving

The verification technique theorem proving is a deductive approach to prove logical formulas. First, the program and its specification (e.g., a contract that specifies the behaviour of each method) are translated into logical formulas (a.k.a. proof obligations). Then the formulas are used to proof correctness of the program. Theorem proving is only supported by FeatureIDE, there are no other tools for supporting it.

L. Consistency Checking

The variability used in implementation artifacts (e.g., features in #ifdef statements) needs to be valid according to variability defined at the feature model. Consistency checking analyses the usage of features; whether a feature has a corresponding implementation and vice versa [15, 16]. Additionally, consistency checking analyzes the feature dependencies with the usage in the source code (e.g., dead or superfluous code in case of incorrect combination of #ifdef statements) [17]

M. Non Functional Properties

A goal of software development is to optimize non-functional properties, such as footprint, performance, and energy consumption [18]. In single system engineering, such properties can be reached by a specialized implementation for the properties defined by stakeholders. However, it is even more challenging to automatically determine the optimal product for a given product line related to a given non-functional property (e.g., with the lowest energy consumption). The goal of research on non-functional properties is to predict such properties for all products based on measurements of some products.

N. Code Metrics

Code metrics are used to compare analyses results and to evaluate their expressiveness. In software product-line analyses often metrics such as lines of code and numbers of features are used. However, such metrics do not take feature dependencies and variability in the source code into account. To compare analysis results for different software product lines, other metrics are required.

IV. TOOL SUPPORT

Proper tool support helps to achieve efficiency for each of these phases. However, there exist tools which integrate and support all or multiple phases of FOSD. There is a large number of tools and some Integrated Development Environment (IDE) exists in practice to support FOSD. Most product line tools are extensible and try to connect the phases. The available tool support for various processes defines in section III is arranged as tables here (table 1 to table 13). Along with implementation strategy, supporting programming language is also mentioned on each tool.

Table 1. Tools Supporting Feature modeling

Tool	Implementation Strategy	Programming Language
AHEAD[21]	FOP	Java
Captain Feature	*	*
DeltaJ[22]	DOP	Java
DOPLER	*	Java
EASyProducer	Preprocessor	*

FaMa	*	Java
FeatureMapper	*	Java
FMT	*	*
Hydra	*	Java
Kumbang	*	Java
Metadoc FM	*	*
S2T2	*	Java
SPLConqueror	*	*
SPLOT	*	Java
VariaMos	*	*
VARMOD	*	*
WeCoTin	*	*
XFeature	*	Java

Table 2. Tools Supporting Feature-Oriented Decomposition

Tool	Implementation Strategy	Programming Language
AOP-Migrator[60]	AOP	Java
ExtractorPL	FOP	*
FLiPEx	AOP	Java
LEADT[1]	VSoC	Java

Table 3. Tools Supporting Variant Preserving Refactoring

Tool	Implementation Strategy	Programming Language
cnife	Preprocessor	C, C++
ext-refactoring	DOP	Java
Morpheus	Preprocessor	C
VAmPiRE	FOP	Java

Table 4. Tools Supporting Product Configuration

Tool	Implementation Strategy	Programming Language
AHEAD[21]	FOP	Java
Captain Feature	*	*
DeltaJ[22]	DOP	Java
DOPLER	*	Java
EASyProducer	Preprocessor	*
FaMa	*	Java
FeatureMapper[55]	*	Java
FMT	*	*
Hephaestus	*	*
Hydra	*	Java
Invar	*	*
Kumbang	*	Java
Metadoc FM	*	*
S2T2	*	Java
SPLConfig	*	*
SPLConqueror	*	*
SPLOT	*	Java
VariaMos	*	*
WeCoTin	*	*
XFeature	*	Java

Table 5. Tools Supporting Product Generation

Tool	Implementation Strategy	Programming Language
AHEAD[21]	FOP	Java
Antenna	Preprocessor	Java
aspectc	AOP	C
AspectC++	AOP	C++
AspectJ	AOP	Java
AspectSharp	AOP	.NET
CaesarJ	AOP	Java
CPP	Preprocessor	C,C++
DeltaJ[22]	DOP	Java
EASyProducer	Preprocessor	*
ELIDE	Preprocessor	Java
FeatureBite	FOP	Java
FeatureC++	FOP, AOP	C++
FeatureHouse[39]	FOP	Java, C, C#, JML, Haskell, XML, Python, Alloy, Featherweight Java, JML, JCop, Stratego, SDF, JavaCC
FeatureJS	FOP, Preprocessor	JavaScript, HTML
Fuji[37]	FOP	Java
javapp	Preprocessor	Java
Munge	Preprocessor	Java
rbFeatures	FOP	Ruby
Spoon	Preprocessor	Java
XFeature	*	Java
XVCL	Preprocessor	

Table 6. Tools Supporting Sampling

Tool	Implementation Strategy	Programming Language
GeneticTestCaseGeneration [27]	*	*
MoSo-PoLiTe	FOP	*
Pacogen[23]	*	*
PLEDGE[26]	*	*
SPLCATool[24]	*	*
Undertaker[25]	Preprocessor	*

Table 7. Tools Supporting Testing

Tool	Implementation Strategy	Programming Language
Asadal[30]	*	*
CPA/Tiger	Preprocessor	C
FTS-Testing	Preprocessor	Featured

		Transition System
GATE[28]	AOP	*
IMoTEP	DOP	*
Kesit[29]	FOP	Java
MATE[31]	*	*
Otter	*	C
ParTeG[32]	*	Java
shared-execution	*	Java
SharQ framework	FOP	*
Splmonitor	FOP	Java
SPLTester[33]	*	*
SPLverifier[43]	FOP	Java, C
Varex[34]	*	PHP
VarexJ	*	Java
Variability-Aware Interpreter[35]	*	WHILE

Table 8. Tools Supporting Type checking

Tool	Implementation Strategy	Programming Language
DeltaJ[22]	DOP	Java
FeatureTweezer[38]	FOP	Java, C
Fuji[37]	FOP	Java
Software Variant Generation System	*	*

Table 9. Tools Supporting Static Analysis

Tool	Implementation Strategy	Programming Language
ACV tool[41]	AOP	*
CPAchecker[44]	Preprocessor	C
CPArec	Preprocessor	C
Golem	Preprocessor	C
SPLLIFT[40]	VSoC	Java
vampyr	Preprocessor	C

Table 10. Tools Supporting Model Checking

Tool	Implementation Strategy	Programming Language
FTS	*	*
JPF-BDD	*	Java
ProVeLines[61]	Preprocessor, FOP	Featured Transition System
Software Variant Generation System	*	*
SPLverifier[43]	FOP	Java, C
VarexJ[43]	FOP	Java
VMC	Preprocessor	Modal Transition System

Table 11. Tools Supporting Consistency Checking

Tool	Implementation Strategy	Programming Language
Captain Feature	*	*
DOPLER	*	Java
FaMa	*	Java
FeatureMapper	*	Java
FMT	*	*
Hephaestus	*	*
Hydra	*	Java
Invar	*	*
Kumbang	*	Java
Linux Feature Explorer	Preprocessor	*
Metadoc FM	*	*
S2T2	*	Java
SPLOT	*	Java
Undertaker[25]	Preprocessor	*
ariaMos	*	*
VARMOD	*	*
WeCoTin	*	Java
XFeature	*	*

Table 12. Tools Supporting Non Functional Properties

Tool	Implementation Strategy	Programming Language
ClaferMOO[45]	*	*
FeatureHouse[39]	FOP	Java, C, C#, JML, Haskell, XML, Python, Alloy, Featherweight Java, JML, JCop, Stratego, SDF, JavaCC
SPLConqueror	*	*

Table 13. Tools Supporting Code Metric

Tool	Implementation Strategy	Programming Language
Ajdtstats[48]	AOP	Java
AJStats[48]	AOP	Java
Cppstats[46]	Preprocessor	CPP
FeatureJS	FOP, Preprocessor	JavaScript, HTML

V. IDE SUPPORT

An Integrated Development Environment (IDE) is software consisting of a range of development tools (e.g. a source code editor, a debugger) presented in a graphical user interface (GUI), thus forming a comprehensive programming environment packaged as a software application. It is necessary to have IDE for FOSD too because it deals with software product line and large-scale software systems. IDE assists the developers by providing a complete solution of the development process. Tighter integration of all development tasks has the potential to improve overall productivity beyond just helping with setup tasks.

The IDEs supporting FOSD paradigm is listed in Table 14. Each IDE is associated with supporting activities, Implementation Strategy and supporting language.

Table 14. Integrated Development Environments for FOSD

Tool	Supporting processes	Implementation Strategy	Programming languages
ABS[58]	Product Generation	DOP	ABS
AJDT[59]		AOP	Java
CIDE[36]	Feature Modeling, Configuration, Product Generation, Feature-Oriented Decomposition, Type Checking, Code Metrics	VSoC	Featherweight Java, Java, C, C#, JavaScript, Haskell, Bali, ANTLR, JavaCC, Properties, HTML, XML, XHTML, XML-People, Python, OSGi Manifest
Colligens[47]	Feature Modeling, Configuration, Code Metrics	Preprocessor	C
DeltaEcore[53]	Product Generation, Feature Modeling, Configuration	DOP	Ecore, Java, UML, Yakindu
Emergo[54]	Static Analysis	Preprocessor	*
Feature Commander[56]		Preprocessor	C
FeatureIDE[42]	Feature Modeling, Configuration, Consistency Checking, Theorem Proving, Code Metrics, Multi Product Line, Testing	Preprocessor, FOP, DOP, AOP	Java, C, C++, C#, JML, Haskell, XML, Python, Alloy, Featherweight Java, JML, JCop, Stratego, SDF, JavaCC
FeatureVisu[57]		Preprocessor	*
Gears[52]	Feature Modeling, Consistency Checking, Configuration, Product Generation	*	*
pure::variants[51]	Feature Modeling, Consistency Checking, Configuration, Product Generation	Preprocessor	*

*We couldn't able to collect the respective information from the journals and websites in our survey. All tools that we mentioned in this paper are collected from the published papers and from the websites related to FOSD. Websites are not cited because it will violate journal reference policy of IJCSE mentioned in the template.

VI. CONCLUSION AND FUTURE WORK

In Feature-Oriented Software-Product-Line Engineering, similar software products are built in an efficient and coordinated manner based on features. While there are efficient techniques and tools to implement FOSD, the current research seeks to scale software analyses, such as type checking, static analyses, model checking, or theorem proving, from single software products to entire software product lines. Based on our insights with classifying and comparing a corpus of 61 research articles specifying the development of software product lines, tools that implement software analyses, design, and development, maintenance, etc are necessary. An overview of such tools helps researchers, lecturers, students, and practitioners to decide whether a tool for a certain analysis strategy or generation technique exists that could be used (e.g., for development of commercial-quality tools, or proof of concepts).

Furthermore, the effort for the development of new tools can be reduced through knowledge about existing tools. We provide additional information about the tools, such as the supported generation mechanism (e.g., Preprocessors), the supported programming languages (e.g., Java). We hope this article can raise awareness of the importance and challenges of tools for FOSD and thus motivate the researchers and practitioners to explore existing tools for the extension or development of new tools/IDE for supporting FOSD paradigm.

References

[1]. Sven Apel, Don Batory, Christian K□astner, and Gunter Saake, "*Feature-Oriented Software Product Lines: Concepts and Implementation*", Springer, 2013.

[2]. Paul Clements, Linda Northrop. "*Software Product Lines: Practices and Patterns*", Addison-Wesley, 2001

[3]. Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, "*Feature-Oriented Domain Analysis (FODA) Feasibility Study*", Technical Report CMU/SEI-90-TR-21,Software Engineering Institute, 1990.

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

[5]. Thomas Thum, Sven Apel, Christian Kastner, Ina Schaefer, Gunter Saake, " *A Classification and Survey of Analysis Strategies for Software Product Lines*", CSUR, 2014

[6]. Sven Apel, Alexander von Rhein, Philipp Wendler, Armin Gr¨oßlinger, Dirk Beyer, " *Strategies for Product-Line Verification: Case Studies and Experiments*", ICSE, pp. 482–491, IEEE, 2013.

[7]. Praveen Jayaraman, Jon Whittle, Ahmed M. Elkhodary, Hassan Gomaa, "*Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis*". MODELS, pp. 151–165. Springer, 2007.

[8]. Malte Plath , Mark Ryan, "*Feature Integration Using a Feature Construct*", SCP, 41(1):53–84, 2001.

[9]. Leonardo Passos, Jianmei Guo, Leopoldo Teixeira, Krzysztof Czarnecki, Andrzej Wasowski, Paulo Borba, "*Coevolution of Variability Models and Related Artifacts: A Case Study from the Linux Kernel*", SPLC, pp. 91–100, ACM, 2013.

[10]. Benjamin C. Pierce, "*Types and Programming Languages*". MIT Press, Cambridge, Massachusetts, USA, 2002

[11]. Flemming Nielson, Hanne R. Nielson, Chris Hankin., " *Principles of Program Analysis*", Springer, 2010.

[12]. Edmund M. Clarke, Orna Grumberg, Doron A Peled. "*Model Checking*", MIT Press, 1999.

[13]. Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, Jean-Fran¸cois Raskin, " *Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines*", ICSE, pp. 335–344, ACM, 2010.

[14]. Alexander Gruler, Martin Leucker, Kathrin Scheidemann, "*Modeling and Model Checking Software Product Lines*.", FMOODS, pp. 113–131, Springer, 2008.

[15]. Thomas Thum, Christian Kastner, Sebastian Erdweg, Norbert Siegmund., "*Abstract Features in Feature Modeling*", SPLC, pp. 191–200. IEEE, 2011

[16]. Reinhard Tartler, Julio Sincero, Wolfgang Schr¨oder-Preikschat, Daniel Lohmann, "*Dead or Alive: Finding Zombie Features in the Linux Kerne*l", FOSD, pp. 81–86. ACM, 2009

[17]. Reinhard Tartler, Daniel Lohmann, Julio Sincero, Wolfgang Schr¨oder-Preikschat, " *Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem*", In Proceedings of the sixth conference on Computer systems, pp. 47–60. ACM, 2011.

[18]. Suzanne Robertson, James Robertson, " *Mastering the Requirements Process: Getting Requirements Right*", Pearson Education, 2012.

[19]. Timo Asikainen, Tomi Männistö, Timo Soininen, "*A Unified Conceptual Foundation for Feature Modeling*", SPLC , IEEE 2006

[20]. C. K¨astner, S. Apel, S. Trujillo, M. Kuhlemann, D. Batory, "*Language-Independent Safe Decomposition of Legacy Applications into Features*", Technical Report 02/2008, School of Computer Science, University of Magdeburg, 2008.

[21]. Don Batory, "*A Tutorial on Feature-Oriented Programming and the AHEAD Tool Suite*", GTTSE, pp. 3-35, Springer, 2006.

[22]. Ina Schaefer, Lorenzo Bettini, Viviana Bono, Ferruccio Damiani, Nico Tanzarella, "*Delta-Oriented Programming of Software Product Lines*", SPLC, pp. 77-91. Springer, 2010.

[23]. Aymeric Hervieu, Benoit Baudry, Arnaud Gotlieb. "*Pacogen: Automatic Generation of Pairwise Test Configurations From Feature Models*", ISSRE, pages 120-129, IEEE, 2011.

[24]. Martin Fagereng Johansen, ystein Haugen, Franck Fleurey, "*An Algorithm for Generating T-Wise Covering Arrays from Large Feature Models*", SPLC, pp. 46-55. ACM, 2012.

[25]. Reinhard Tartler, Daniel Lohmann, Julio Sincero, Wolfgang Schr□oder-Preikschat, "*Feature Consistency in Compile-Time-Con_gurable System Software: Facing the Linux 10,000 Feature Problem*". In Proceedings of the sixth conference on Computer systems, pp. 47-60. ACM, 2011.

[26]. Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Yves Le Traon, "*PLEDGE: A Product Line Editor and Test Generation Tool*", SPLC, pp. 126-129, ACM, 2013

[27]. Faezeh Ensan, Ebrahim Bagheri, Dragan Ga_sevi_c. "*Evolutionary Search-Based Test Generation for Software Product Line Feature Models*", CAiSE, volume 7328 of Lecture Notes in Computer Science, pp. 613-628, Springer, 2012.

[28]. Yankui Feng, Xiaodong Liu, Jon Kerridge, "*A Product Line Based Aspect-Oriented Generative Unit Testing Approach to Building Quality Components*", COMPSAC, volume 2, pp. 403-408, IEEE, 2007.

[29]. Thomas Thum, Ina Schaefer, Martin Kuhlemann, Sven Apel, "*Proof Composition for Deductive Veri_cation of Software Product Lines*", VAST, pp. 270-277, IEEE, 2011.

[30]. Kyungseok Kim, Hyejng Kim, Miyoung Ahn, Minseok Seo, Yeop Chang, Kyo C Kang, " *ASADAL: a Tool System for Co-Development of Software and Test Environment Based on Product Line Engineering*", ICSE, pp. 783-786, ACM, 2006.

[31]. Georg Puschel, Ronny Seiger, Thomas Schlegel, "*Test Modeling for Context-aware Ubiquitous Applications with Feature Petri Nets*", In MODIQUITOUS, 2012.

[32]. Stephan Wei_leder, Dehla Sokenou, Bernd-Holger Schlinglo, "*Reusing State Machines for Automatic Test Generation in Product Lines*", MoTiP, pp. 19-28, 2008.

[33]. Chang Hwan Peter Kim, Don Batory, Sarfraz Khurshid, "*Reducing Combinatorics in Testing Product Lines*", AOSD, pp. 57-68,ACM, 2011.

[34]. Hung Viet Nguyen, Christian K□astner, Tien N. Nguyen, "*Exploring Variability-Aware Execution for Testing Plugin-Based Web Applications*", ICSE, ACM, 2014.

[35]. Christian K□astner, Alexander von Rhein, Sebastian Erdweg, Jonas Pusch, Sven Apel, Tillmann Rendel, Klaus Ostermann, "*Toward Variability-Aware Testing*.", FOSD, pp.1-8, ACM, 2012.

[36]. Christian K□astner, Sven Apel, "*Virtual Separation of Concerns-a Second Chance for Preprocessors*", Journal of Object Technology, pp.59-78, 2009

[37]. Sven Apel, Sergiy Kolesnikov, J□org Liebig, Christian K□astner, Martin Kuhlemann, Thomas Leich, "*Access Control in Feature-Oriented Programming*", SCP, pp.174-187, 2012.

[38]. Sven Apel, Wolfgang Scholz, Christian Lengauer, Christian K□astner, "*Language-Independent Reference Checking in Software Product Lines*", FOSD, pp.65-71, ACM, 2010.

[39]. Sven Apel, Christian K□astner, Christian Lengauer, "*Language-Independent and Automated Software Composition: The FeatureHouse Experience*", TSE, pp.63-79, 2013.

[40]. Eric Bodden, Tarsis Toledo, Marcio Ribeiro, Claus Brabrand, Paulo Borba, Mira Mezini, "*SPLLIFT: Statically Analyzing Software Product Lines in Minutes Instead of Years*", PLDI, pp.355-364, ACM, 2013.

[41]. Herbert Klaeren, Elke Pulvermueller, Awais Rashid, Andreas Speck*," Aspect Composition Applying the Design by Contract Principle*", GCSE, pp. 57-69, Springer, 2001.

[42]. Thomas Th□um, Christian K□astner, Fabian Benduhn, Jens Meinicke, Gunter Saake, Thomas Leich, "*FeatureIDE: An*

Extensible Framework for Feature-Oriented Software Development", SCP, pp.70-85, 2014.
[43]. Sven Apel, Hendrik Speidel, Philipp Wendler, Alexander von Rhein, Dirk Beyer, "*Detection of Feature Interactions Using Feature-Aware Verification*", ASE, pp. 372-375, IEEE, 2011.
[44]. Dirk Beyer , M. Erkan Keremoglu, " *CPAchecker: A Tool for Con_gurable Software Verifcation*", CAV, pp. 184-190, Springer, 2011.
[45]. Micha l Antkiewicz, Kacper B , ak, Alexandr Murashkin, Rafael Olaechea, Jia Hui Jimmy Liang, Krzysztof Czarnecki, "*Clafer Tools for Product Line Engineering*", SPLC, pp.130-135, ACM, 2013.
[46]. Jorg Liebig, Christian Kastner, Sven Apel, "*Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of c Code*", AOSD, pp. 191-202, ACM, 2011.
[47]. Flavio Medeiros, Thiago Lima, Francisco Dalton, M_arcio Ribeiro, Rohit Gheyi, Baldoino Fonseca, "*Colligens: A tool to support the development of preprocessor-based software product lines in* c", CBSoft, 2013.
[48]. Sven Apel ,Don Batory, " *How AspectJ is Used: An Analysis of Eleven AspectJ Programs*" JOT, pp.117-142, 2010.
[50]. Oster, S., Zorcic, I., Markert, F., Lochau, "*MoSo-PoLiTe: tool support for pairwise and model-based software product line testing*",In: Proceedings of VAMOS 2011.
[51]. Danilo Beuche, "*Systems and software variability management*", Springer, pp. 173-182, 2013.
[52]. Charles Krueger, Paul Clements, "*Systems and Software Product Line Engineering with Gears from BigLever Software*", SPLC-14, pp.121-125, ACM, 2014.
[53]. Seidl, Christoph & Schaefer, Ina Assmann, "*DeltaEcore-A Model-Based Delta Language Generation Framework*", Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI), 2014.
[54]. Ribeiro, Márcio & Tolêdo, Társis & Winther, Johnni & Brabrand, Claus & Borba, Paulo, " *Emergo: A Tool for Improving Maintainability of Preprocessor-based Product Lines*", 2012
[55]. Florian Heidenreich, Jan Kopcsek, Christian Wende, "*FeatureMapper: Mapping Features to Models*", ICSE Companion'08 Companion of the 30th international conference on Software engineering, pp.943-944, 2008.
[56]. Janet Feigenspan, Maria Papendieck, Christian Kästner, Mathias Frisch, Raimund Dachselt, "*FeatureCommander: colorful #ifdef world*", SPLC '11 Proceedings of the 15th International Software Product Line Conference, Volume 2,48, 2011.
[57]. Sven Apel, Dirk Beyer, "*Feature Cohesion in Software Product Lines: An Exploratory Study*", ICSE 2011.
[58]. Y. Wong, E. Albert, R. Muschevici, J. Proença, J. Schäfer, R. Schlatte, "*The abs tool suite: modeling executing and analysing distributed adaptable object-oriented systems*", International Journal on Software Tools for Technology Transfer, vol. 14, no. 5, pp. 567-588, 2012.
[59]. K. Gybels , J. Brichau, "*Arranging Language Features for more Robust Pattern-Based Crosscuts*", In Proc. Int'l Conf. Aspect-Oriented Software Development. 2003.
[60]. M. P. Monteiro , J. M. Fernandes, " *Towards a catalog of aspect-oriented refactorings*",. In Proc. of the 4th International Conference on Aspect-Oriented Software Development (AOSD), pages 111-122. ACM Press, March 2005.
[61]. Maxime Cordy, Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, "*ProVeLines: A Product Line of Verifiers for Software Product Lines*", SPLC 2014
[62]. Maushumi Lahon and Uzzal Sharma, "*The Intricacies of Software Component Composition*", International Journal of Computer Sciences and Engineering, Vol.03, Issue.01, pp.111-117, 2015.

Authors Profile

Mrs Kala K. U. pursed 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 pursed 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.

