A Comprehensive Study of Software Product Line Frameworks

Md. Mottahir Alam
(Ph.D. Scholar)
Department of Electronics & Communication Engineering,
Singhania University,
Jhunjhunu, Rajasthan, India

Asif Irshad Khan
Dept. of Computer Science,
FCIT,
King AbdulAziz University,
Jeddah, KSA

Aasim Zafar
Dept. of Computer Science,
Aligarh Muslim University,
Aligarh, UP,
India

ABSTRACT
In today's competitive software market, there is a constant need to launch new features and products or enhance the existing products in a flawless, accelerated and cost-effective manner. SPLE (Software Product Line Engineering) refers to engineering technique which reuses common set of features and at the same time it has provisions to manage features which are product-specific and not shared by other products in the product line. A product line is a set of products that are developed with a focus on specific market segment or satisfying some specific business requirements. It is an approach for implementing software variability and helps to extend, customize or configure the products in order to use in a specific context. Researchers have proposed several SPL approaches. In this paper, we did a comprehensive study and analysis of various existing SPL approaches and discussed the outcomes of our review. We tried to present the backgrounds of various SPL approaches, and identified key issues that need to be focused in future research.

General Terms
Software Product Line Engineering, Variability Management.

Keywords
Software Product Line, feature coverage, variability, comparison framework, product line methods, feature modeling.

1. INTRODUCTION
Software Product Line Engineering (PLE) is considered as the engineering approach of selection of tools and techniques that shared set of engineering assets and an efficient means of production, that together address a particular market segment or fulfill a particular mission [1][2]. Fig 1 shows the concept of Software Product Line Engineering and Management.

As given in the figure, Software engineering artifacts can include requirement analysis, Software modeling, software design, Coding, testing (Unit, integration and system) and much more. all of these need to be managed and produced in variants that match the product. "Assets" is a name given to those artifacts which supports a product. The paradigm under discussion utilizes the shared assets to instill with distinction points i.e. places where the assets can be instantiated in different ways to support each product in a product line. It is an input (product specification) to the product configurator and designs the assets appropriately for that product. [1]

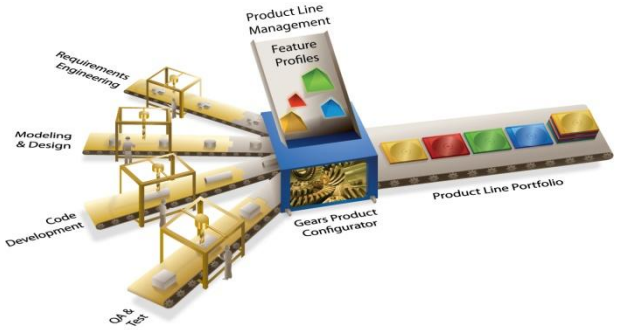

Fig 1: Software Product Line Management [1]

The Software Engineering Institute Carnegie Mellon University [2] listed the following benefits associated with product lines:

i) enhance product quality
ii) increased productivity
iii) lowers time to market
iv) mitigate product risks
v) support market agility
vi) enhance customer satisfaction
vii) support mass customization

Some of the key benefits beside above benefit of SPL are that it aligns engineering resources with business objectives to ensure efforts are focused on the most profitable features and functions. SPL is also helpful in managing increased product diversity without a corresponding increase in resources. Further, it improves productivity and efficiency and reduces per-product development cost, resulting in higher profit margins. It is also very helpful in reducing time to market for new and updated products, while increasing agility to help us react to new opportunities and changing market conditions. SPL increases product quality and improve risk management [3].

Various PLE case studies have shown momentous measured improvements in time to market, cost, product quality, product line scalability, and productivity, compared to product-centric development [2]. A List of features describes each product in SPL: "A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems". Life cycle phase artifacts are defined by differences in product and its features. This enables the stakeholders to speak the same language and streamline the engineering processes. A product that we are building needs to be defined so that the shared assets (requirement analysis,

Software modeling, software design, Coding, testing (etc.) can also be appropriately configured. Rather than devising a mechanism and "language" for each artifact e.g. compiler directives for code, text variables for documents, attributes for requirements, and so forth, we use a variation mechanism which is small but consistent.

The use of application engineering thus becomes unsubstantial; industrial strength high-end automation is used to produce products by configuring the shared assets appropriately.

The illustration below shows that a wide variety of shared products are used to derived products by the configurator automatically. Different features define the variations in products which is supposed to tell the configurator the characteristics of the product and how they were obtained by configuring the shared assets [1].

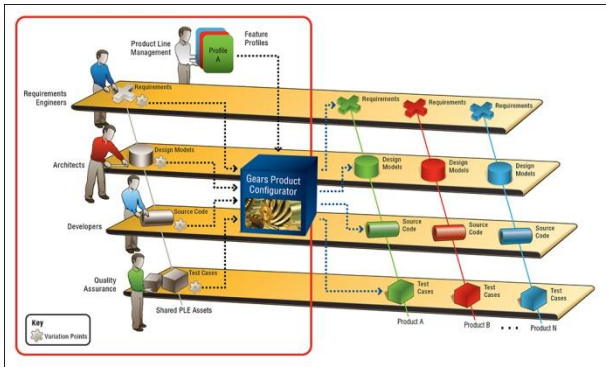

Fig 2: Concept of SPLE [1]

The paper is organized as follows: Section 2 covers overview of software product line. Section 3 provides current software product line approaches. Section 4 evaluates the current approaches. Lastly, section 5 draws conclusions and future works.

2. OVERVIEW OF SOFTWARE PRODUCT LINES

A software product line is a software-intensive engineering technique sharing a common, managed set of features that satisfy particular market requirement or mission, and that are developed from a common set of core assets in a specific way [4].

This definition helps in classifying main roles in a product line organization. The role of core asset developers is to make available a set of assets, such as architectures, specifications, and implementations, which are then used by the team of product developers to produce end-products.

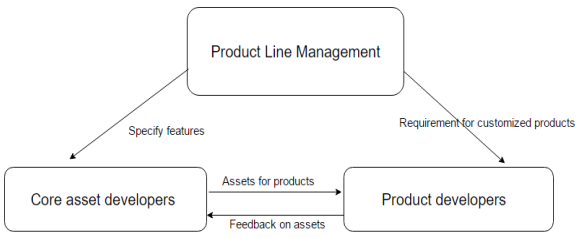

Fig 3: Roles in a software product line [4]

The work of these two teams are facilitated and synchronized by product line managers as demonstrated in fig 3. The role of executives is to set strategic goals related to production and delivery-time of products and assign responsibilities to achieve these fixed goals.

2.1 Benefits of Product Lines

Requirements: The product line requirements are common to other similar processes based on the established requirement base. This saves extensive requirement analysis and also assures better feasibility.

Architecture: Constructing architecture for a software system is a huge investment of the best engineering manpower by the company. The architecture is built keeping in mind the goals of performance, reliability, modifiability etc. Architecture forms the base of the process. The product line architecture is unique for each product. A sound architecture saves a lot risk and time.

Components: The core asset base components are used almost exclusively in each product. Although the design, data structures and algorithms are intact, we may have to make some changes in the the components using inheritance or parameters. The product line architecture provides component specifications for all the common components in the product line.

Modeling and analysis: For any product line, performance models and the associated analyses are basic assets. With each new product the quality of the product in the product line increases and the bugs continues to decrease.

Testing: In a software product line, generic test plans, test processes, test cases, test data, test harnesses, etc for all the products will be similar and will be ready for future products. It may need to make just few changes based on the variations related to the specific product.

Planning: Previous product development projects provide a baseline for the budgets and schedules. These provide a reliable basis for the product work plans.

Processes: For any future product, configuration control boards, configuration management tools and procedures, management processes, and the overall software development process are already in place. Since they have been used before, therefore have a proven robustness, reliability and responsiveness to the organization's special needs.

People: Because of the similarities in the processes fewer people are required to build products and the work force can adapt to newer processes easily.

2.2 Software Product Lines-Key Processes

The software product line engineering paradigm separates two processes. This division is beneficial in the separation of the two arenas i.e. to make an application in a short which caters to the customer needs and to build a product which is robust. For maximum effect the process must interact in such a way that benefits both. For example, the platform should be able to aid in the application development and the application development must be aided in using that specific platform. This split into two processes also shows a split of concerns with regards to variability. Domain engineering ensures the production of applications by making the variability available. The correct amount of flexibility in reusable artifacts will defines a platform. A considerable part of application engineering is made up of reusing the platform and binding variability for various applications as required [2].

Domain Engineering: domain engineering process aims to:

1. Define the variability and the commonality of the product line.
2. Define the scope of the software product line and
3. how to achieve the desired variability by constructing the reusable artifacts.

The domain engineering consists of several sub process. Each of the sub-process has to detail and refine the variability by following the engineering sub-process and providing the practicability achieving the required variability of the preceding domain engineering sub process.

The Application Engineering: This engineering sub-process includes all undertakings required for application requirements specification development. The development of effective application requirements enables the achievable amount of domain artifact reuse. Hence, the detection of variation between application requirements and available platform capabilities is a major hurdle. The domain requirements and the product roadmap with the main features of the corresponding application make the input to this sub process. Also, there may be some other requests e.g., from a customer that may not have been taken during the process of domain requirements engineering. The output of the said process of that specific application will be requirement specification.

3. CURRENT SOFTWARE PRODUCT LINE APPROACHES

In this section, a number of current Software Product Line approaches are given as an overview. These approaches are respectively FAST [6], FODA [7], FORM [10] RSEB [11], FeatuRSEB [13], ConIPF [14], PuLSE [15], KobrA [28, 29]. While some of them have focus centered on Domain Engineering, the others propose a complete software product line approach.

3.1 FAST (Family-oriented abstraction, specification and translation)

It is feature-based model proposed by Weiss et.al [6]. It helps in applying product-line principles to software engineering process. It can be used in cases where a range of products are developed which have major share of common artifacts among themselves. These common features can be common behavior, common interfaces, or common code. The main goal is to analyze common artifacts among a group of products and then building potential software families. This helps in making the software development more robust by reusing the common artifacts, which in turn decreases the development cost, and reduces the time-to-market as shown in fig 4.

In this framework, the processes can be divided into following three sub-processes:

1. Domain qualification: Under this, an economic model of the software product line is generated by cost analysis.
2. Domain engineering: In domain engineering, the main agenda is to analyze the commonalities in the potential product line, and then coming up with a family definition and product line infrastructure as well as reusable core assets.
3. Application engineering: In application engineering, product line family is developed by using the reusable core assets.

A family of products can be defined using a common platform. This platform is built taking into account the similarities between several products close to each other. As per FAST, the variability's' within the product family members can be managed and executed by using different variation techniques such as conditional compilation. It gives n scope to provide an iteration and reusability of future processes in software engineering.

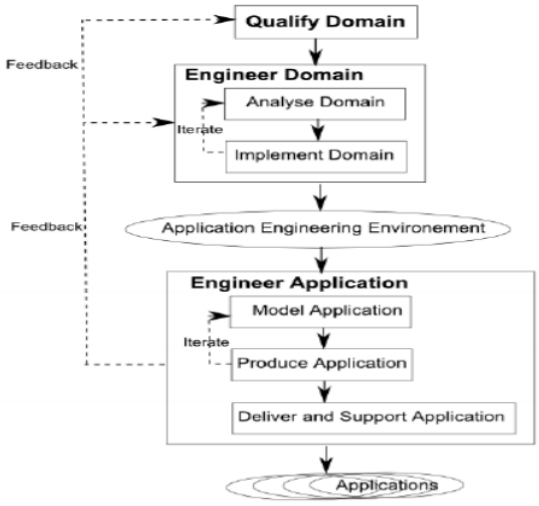

Fig4: FAST Flow Process [9]

The core objective of FAST is to provide a framework to implement iteration and reusability for future processes in a consistent, smooth and disciplined way which can help in reducing the development time and cost.

As per report, FAST framework is already being successfully implemented in industry.

3.2 FODA

FODA [7] or Feature-Oriented Domain Analysis has been proposed by Kang to identify and model features. It is based on domain analysis technique in which distinct features within a product line are identified. These features combined together to define the domain of the product family. This approach is followed because the variability determination mechanisms which are given within the components are generated by means of a domain–specific language which is a data–intensive extension of a textual version of the feature diagrams. It involves three basic processes, namely,

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 first step, which comes under domain analysis, is to define the domain and finalize the products of the product family. Next step is to analyze the features by performing the commonality and variability analysis .Finally, the modeling of the features is performed as per the core and varied artifacts which helps in developing the product line family are developed in a structured and smooth fashion.

The three major phases in FODA which guides the success of the process are as follows:

i) Context analysis

ii) Domain modeling

iii) Architecture modeling

In context analysis, we define the domain and build a context model which contains all the requirements of the products. In domain modeling, we try to model the requirements by using the results of commonality and variability analysis which gives feature models as the output. After this, architecture modeling is performed to create the reference architecture by using the feature models. The output of this phase is the reference architecture which is used to develop the specific products.

FODA uses state activity charts and state charts to model functional and behavioral aspects correspondingly. These charts are proposed by Structured Analysis and Design Technique (SADT).

FODA provides both a process to determine common and variable features of concept instances, including their interdependencies, and a notation to represent them in feature models consisting of feature diagrams with some additional information such as short semantic descriptions of each feature, constraints, default dependency rules, etc.

While FODA (feature oriented domain analysis) method is the first input to manage variability in 1990 [8]. However many problems still need to be resolved.

Fig 5 shows the basic elements which are used to model functionality. These models are data model and control model respectively. As previously explained in the major phases of FODA, initially, context diagrams are created for both data and control models. Data flow diagrams (DFD) and control flow diagrams (CFD) which are alike DFDs are created after the functions are decomposed.

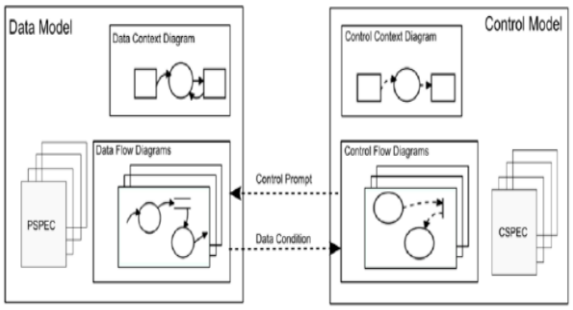

Fig 5: The functional model with the fundamental elements. [9]

Process and control specifications are used to identify and control functionality and behavior. The interactions between these models are through control prompt and data condition. In fig 6, the data flow is shown by solid lines and the control flow is shown by dashed lines.

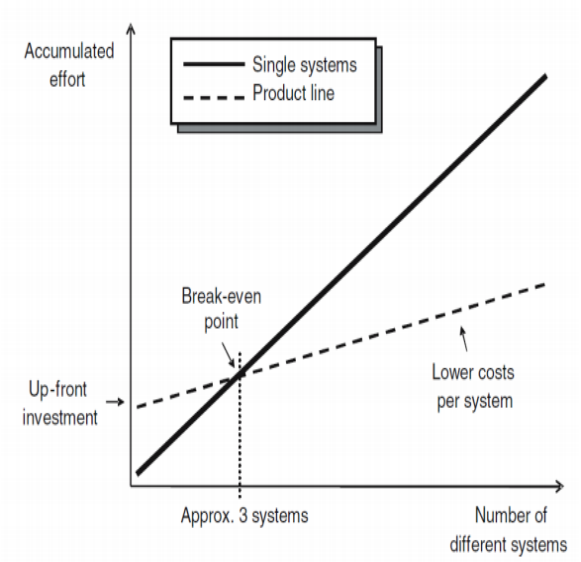

Fig 6: Economics of software product line engineering [9]

3.3 FORM (Feature-Oriented Reuse Method for product line software engineering):

FORM (Feature-Oriented Reuse Method)[10] is a method based on feature orientation which analyzes the features of the domain, and then use these features to provide the software product line architecture .In other words, "FORM is a systematic method that focuses on capturing commonalities and differences of applications in a domain in terms of "features" and using the analysis results to develop domain architectures and components" [10].

Kyo C. Kang and his co-fellows in Pohang University of Science and Technology, Korea, propose a Feature-Oriented Reuse Method (FORM) as an extension to the Feature-Oriented Domain Analysis (FODA) method [7]. FORM is an extension of FODA to the software design and implementation phases and is used in the analysis of domain features which is further used to develop domain architectures and reusable components.

FORM method is useful for applying domain analysis results to reusable and adaptable domain components. It has specific guidelines on which it works. FORM has found special application as a software tool in many industrial processes.

FORM method is specifically used in the domains of telecommunication engineering as well as information technology. However it can be applicable to other specified domains depending on the coherence of the feature model.

As a first step towards modeling for FORM, the commonalities and variabilities of a product line are studied in a detailed manner. Context analysis is an initial step towards software development and starts with information on systems and their features.

Feature model is the product of this domain engineering. It also creates the reference architecture as well as reusable components as an output. Further application software is developed using application engineering. For this features are selected from the feature model, application architecture is selected from the reference architecture and reusable components are also enlisted.

Feature model captures the commonalities and differences of the process as reflected in the performance of the software. There ought to be a common understanding between the customers and developers as regards the features and capabilities of the software.

FORM develops reusable and adaptable domain artifacts which are constructed after a detailed analysis of domain features. FORM is an extension of FODA.

3.4 RSEB

Reuse-Driven Software Engineering Business (RSEB) [11] is a use-case driven systematic reuse process based on the UML notation. It is an iterative and use-case-centric method which facilitates the development of reusable object-oriented software as well as software reuse. The main focus in this process is on the use cases. Under this process, firstly we describe the requirements for the product line domain with the help of use cases. Then, the domain architecture and reusable artifacts are designed. Lastly, object models are created with the help of these architecture and artifacts which are mapped to the use cases [12].

The Unified Modeling Language (UML) is used to capture the variabilities which are identified in the use cases and object models. The use cases and object models are structured using variation points and variants.

RSEB has a number of steps in which an engineering activity takes place, namely requirements engineering, architecture family engineering, component system engineering and application system engineering. The outputs of these engineering processes are as follows [12]:

i) Requirements Engineering: In this engineering phase, variation points and variations that are defined by use cases.

ii) Architecture Family Engineering: a layered architecture is designed with the help of use cases.

iii) Component System Engineering: reusable components are created

iv) Application System Engineering: In this engineering phase, products are created using the reusable components.

3.5 FeatuRSEB

Featuring Reuse-Driven Software Engineering Business (FeatureRSEB) is derived by bringing the FODA [7] and RSEB [11] methods together. It was suggested by Griss et al. in 1998 [13]. Two more processes from FODA i.e. the domain engineering and feature modeling are used to start the RSEB process. There is no feature model created during the process even though RSEB manages variability in use cases in an informal way.

Though FeatuRSEB uses the feature models of FODA, these feature models comprise of slightly different diagrams, which are illustrated in a tree or a network notation. These variation points are represented explicitly with the help of these new notations [12].

Domain engineering has been divided into a series of logical steps under different heads. Step one to three belong to domain analysis while steps four to seven are categorized as component engineering according to Griss et al.

The steps have been named as follows [13]:

i) Domain identification and scoping

ii) Choosing and analyzing the requirements, examples and trends

iii) Identification, factoring and classifying the feature sets

iv) Developing the domain model and architecture

v) Representing the variability and commonality

vi) Exploiting the variability and commonality

vii) Implementing the reusable components and packaging them

3.6 ConIPF

Its full form is Configuration of Industrial Product Families and it is a European FP6 project [14]. This concept was put forward by Eriksson in whose words ConIPF is "a project which wants to integrate both the product line approach and the structure-oriented configuration 35 technologies [12]".
As is common with such software line approaches development with reuse is the driving principle behind this software product line. There is ample provision for adaptation of configuration methodologies by using artificial intelligence [14].

3.7 PuLSE

Fraunhofer Institute Experimental Software Engineering (IESE) designed the Product Line Software Engineering (PuLSE) in late 1990s [15]. According to the PuLSE approach the focus of software product line should be on the products rather than on the organizational aspects.

Knauber et al. state that [16], not only large organizations but also relatively small companies can benefit from this methodology.

The overall philosophy of PuLSE has been put forward by Eriksson in the following words- [12]:

i) PuLSE provides a complete framework that covers the whole software product line development life cycle, including reuse infrastructure construction, usage, and evolution.

ii) PuLSE is modular and customizable: It consists of six technical components that can be selected and instantiated in order to satisfy the needs of specific companies.

iii) PuLSE can be introduced incrementally by augmenting existing software development processes and products with product line specific aspects step by step.

PuLSE is composed of three major components viz. the deployment phase, the technical components and the support components in that order. These have been depicted in fig 7.

The principle of this method is that the deployment phases use the technical and support components with the purpose of detailing a particular software product line [15].

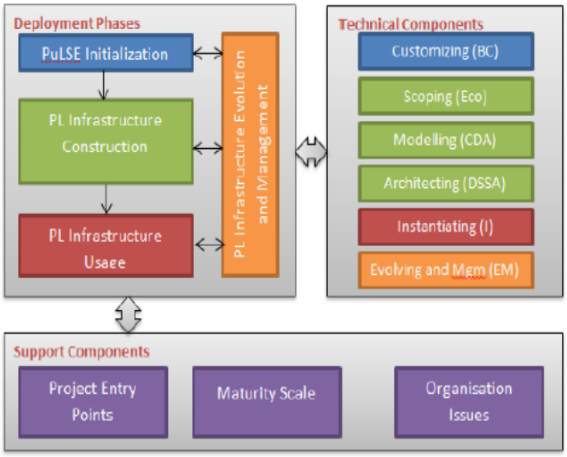

Fig 7: The main phases of PuLSE [17]

3.8 KobrA

The KobrA (Komponentbasierte Anwendungsentwicklung) method that has been developed by Fraunhofer Institute Experimental Software Engineering (IESE). It is a component-based software product line approach [18]. This is a novel approach in the fact that it is a combination of reuse in small concept in component-based approaches and reuse in large concept in software product line methodology [12].

Atkinson et al. have put the argument that "the product-line and component-based approaches to software development seem to have complementary strengths. They both represent powerful techniques to support reuse, but essentially at the opposite ends of the granularity spectrum [19]". Apart from being a software product line methodology KobrA is also a single system development approach.

The two major activities that form the backbone of KobrA are framework engineering and application engineering. These activities are further described as follows:

i) *Framework Engineering*: This is the phase in which a generic framework is created. This framework defines the common and variable features in an explicit manner. To construct the framework, a set of KobrA components are used statically. The outputs of this phase can be measured in the form of number of framework models which are described in text and UML notations. In this phase product line approach is used.

ii) *Application Engineering*: To initialize the generic framework, a component based approach is applied. That is the hallmark of this phase. The components are specified and realized in two levels. In component specification, the externally visible behavioral aspects and properties are defined.

Decomposition of the components into subcomponents is described in component realization. The end objective of this approach is to build products with particular variations in terms of the specific customer needs. Hence the output is product models which are described in text and UML notations.

Muthig et. al in their technical report demonstrate application of KobrA approach by developing a library system product line.

4. EVALUATION OF CURRENT APPROACHES

Going through different approaches, we found that the abstraction level is very high for all of the approaches and there is no detailed guideline to apply these approaches.
The commonality among all the approaches is that they all follow similar processes. The starting point in all the cases is context analysis. Context analysis is followed by domain engineering as well as application engineering. Exploiting the commonalities and variation is one main concern of these processes. However, there are no detailed guidelines for the application of these approaches owing to the immense level of abstraction.
FODA and FeatuRSEB guarantees to solve the issues primarily in domain engineering phase. On the other hand, FAST, FORM and KobrA promises to provide a comprehensive solution for all the phases of software product line engineering. However, infact, the domain engineering phases have more attention than application engineering phases in all of these approaches [17].

In terms of variability, it is observed that some of these approaches use the feature models. Although all of FODA, FORM and FeatuRSEB uses feature models but FODA was first to use it. On the other hand, PuLSE, and KobrA use decision models. FAST method manages the variability in a text format by using commonality analysis [17].

5. CONCLUSION AND FUTURE WORK

Software Product Line has been an area of research and innovation for the last two decades. Common assets which lie at the core of this development comprise of requirements, design, architecture, test plans, reusable software components, test cases and other artifacts. Utilizing common assets for product development increases the productivity, reduce cost as well as marketing time. Hence they decrease the overall development effort.

This paper deals with different product line concepts and approaches. Starting with general principles of Software Product Lines, other aspects of Software Product Line viz organization, requirements, feature and functional model, reference architecture, costs and benefits and variability management techniques have been explained. The paper also enumerates a number of Software Product Lines projects and methodologies.

Background research has found that abstraction level is so high for all of the approaches that there is no detailed guideline for practically applying these approaches. Also, there is no tool that creates reference architecture using both feature modeling and functional modeling.

There are more models of software product lines available in the academics as compared to that in the commercial category. The industry needs to make more efforts in this area to offset this deficiency. One remarkable feature of the SPL model is that there are no standards available to manage variability among family of products.

In maturity phase the software product line has many challenges viz management of variability, lack of standards for variability management, deficient tool support. It is important that the industry and academicians lay emphasis on product line to bring new methods for variability management and improve the existing approaches and innovate professional tools to provide support to the entire development life cycle of product line.

6. ACKNOWLEDGMENTS

Our sincere thanks to all the researchers or individuals who helped us with their valuable comments and suggestions.

7. REFERENCES

[1] BigLever Software, Inc, "What Is Product Line Engineering?" 2016, Product Line Engineering Overview. N.p., n.d. Web. 26 Aug. 2016.

[2] Günter Böckle Klaus Pohl Frank van der Linden , A Framework for Software Product Line Engineering

[3] C. W. Krueger, 2011, Introduction to Product Line Engineering for Systems and Software, IBM Technical Seminar: Empowering the Embedded Systems Developer, BigLever Software

[4] John D. McGregor, 2004 Software Product Lines, Clemson University and Luminary Software, U.S.A. JOURNAL OF OBJECT TECHNOLOGY, Published by ETH Zurich

[5] Peter A. MSE. 1994 Introduction to the SEI's Software Product Line Framework, www.star.cc.gatech.edu/documents/PeterAbowd/SEI.pdf. N.p., 2016. Web. Aug. 2016.

[6] David M. Weiss and Chi Tau Robert Lai. 1999 Software product-line engineering: a family-based software development process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[7] M. Eriksson 2003 An Introduction to Software Product Line Development, Proceedings of Ume's 7th Student Conference in Computing Science, UMINF-03.05, ISSN 0348-0542, pp. 26-37.

[8] Chastek, G., Donohoe, P., Kang, K. C., and Thiel, S. 2001 Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute (SEI)

[9] MERT B. 2013. Component-Based Reference Architecture Tool For Software Product Line Engineering, Faculty of Engineering and Physical Sciences University of Manchester.

[10] Kyo C. Kang , Sajoong Kim , Jaejoon Lee , Kijoo Kim , Euiseob Shin , Moonhang Huh, 1998 FORM: A feature-oriented reuse method with domain-specific reference architectures, Annals of Software Engineering, 5, p.143-168.

[11] M. Griss, J. Favaro, and M. d'Alessandro. 1998 Integrating Feature Modeling with the RSEB. In Proceedings of the 5th International Conference on Software Reuse, pages 76-85, Vancouver, BC, Canada.

[12] M. Eriksson 2003 An Introduction to Software Product Line Development, Proceedings of Ume's 7th Student Conference in Computing Science, UMINF-03.05, ISSN 0348-0542, pp. 26-37.

[13] M.L. Griss, J. Favaro, and M. d'Alessandro. 1998 Integrating feature modeling with the rseb. In Software Reuse, 1998. Proceedings. 5th International Conference on, pages 76-85.

[14] L. Hotz, T. Krebs and A. Gunter. 2002 Knowledge-based configuration for configuring combined hardware/software systems. In Proceedings of Workshop Planen, Scheduling und Konfigurieren, Entwerfen Puk pages 61–70.

[15] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J. DeBaud. 1999 Pulse: a methodology to develop software product lines. In Proceedings of the symposium on Software reusability, SSR '99, pages 122-131, New York, NY, USA, ACM.

[16] P. Knauber, D. Muthig, K. Schmid, and T.Widen. Applying Product Line Concepts in Small and Medium-Sized Companies. IEEE SOFTWARE, September 2000.

[17] E. Yourdon and L.L. Constantine. Structured design: fundamentals of a discipline of computer program and systems design. Yourdon Press, 1978.

[18] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. 2002 Component-based product line engineering with UML. AddisonWesley Longman Publishing Co., Inc., Boston, MA, USA.

[19] C. Atkinson, J. Bayer, and D. Muthig. 2000 Component-Based Product Line Development: The KobrA Approach. In Proceedings of the 1st Software Product Line Conference, pages 289-309.

[20] Khan, A. I. and Qurashi, R. J. and Khan, U. A. 2011. A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies, IJCSI International Journal of Computer Science Issues, 8(4).

8. AUTHOR PROFILE

Mr. Md Mottahir Alam: is a PhD Scholar in Computer Science & engineering in Singhania university. He has around six years of experience working as Software Engineer (Quality) for some leading software multinationals where he worked on projects for companies like Pearson and Reader‟s Digest. He is ISTQB certified software tester. He has received his Bachelors degree in Electronics & Communication and Masters in Nanotechnology from Faculty of Engineering and Technology, Jamia Millia Islamia University, New Delhi.

His research interest includes Software Engineering esp. Software Product Line Engineering, Software Reusability, Component-based and Agent-based Software Engineering.

Dr. Asif Irshad Khan received his Bachelor and Master degree in Computer Science from the Aligarh Muslim University (A.M.U), Aligarh, India in 1998 and 2001 respectively. His PhD degree in Computer Science and Engineering from Singhania University, India.

Dr. Asif is working as faculty member in the Department of Computer Science, King AbdulAziz University, Jeddah, Saudi Arabia. His current research interest includes Software Engineering with a focus on Component Based and Software Product Line Engineering. He is member of editorial board and reviewer of several journals.

Dr. Aasim Zafar is working as Associate Professor in Computer Science Department, AMU, Aligarh. In research, his current interests include e-learning, mobile learning, virtual learning environments and mobile ad hoc networks. He received his PhD degree in Computer Science from Aligarh Muslim University, India and has a number of research papers to his credits. Dr. Zafar is member of Internet Society (ISOC). He is member of editorial board and regular reviewer of several journals.