2009

Software Product Line Engineering: Future Research Directions

Luiz Fernando Capretz
University of Western Ontario, lcapretz@uwo.ca

Faheem Ahmed
Thompson River University, fahmed@tru.ca

Software Product Line Engineering: The Future Research Directions

Faheem Ahmed[1], Luiz Fernando Capretz[2], Muhammad Ali Babar[3]

[1]College of Information Technology, P.O. Box 17551, United Arab Emirates University,

Al Ain, United Arab Emirates

[2]Department of Electrical & Computer Engineering, Faculty of Engineering, University of Western Ontario, London, Ont., Canada N6A 5B9

[3]Lero, University of Limerick, Ireland

[1]f.ahmed@uaeu.ac.ae, [2]lcapretz@eng.uwo.ca, [3]malibaba@lero.ie

Abstract: The recent trend of switching from single software product development to lines of software products in the software industry has made the software product line concept viable and widely accepted methodology in the future. Some of the potential benefits of this approach include cost reduction, improvement in quality and a decrease in product development time. Many organizations that deal in wide areas of operation, from consumer electronics, telecommunications, and avionics to information technology, are using software product lines practice because it deals with effective utilization of software assets and provides numerous benefits. Software product line engineering is an inter-disciplinary concept. It spans over the dimensions of business, architecture, process and organization. The business dimension of software product lines deals with managing a strong coordination between product line engineering and the business aspects of product line. Software product line architecture is regarded as one of the crucial piece of entity in software product lines. All the resulting products share this common architecture. The organizational theories, behavior and management play critical role in the process of institutionalization of software product line engineering in an organization. The objective of this chapter is to discuss the state of the art of software product line engineering from the perspectives of business, architecture,

organizational management and software engineering process. This work also highlights and discusses the future research directions in this area thus providing an opportunity to researchers and practitioners to better understand the future trends and requirements.

1.1 Introduction

In today's digitized economy, organizations endeavor, to the best of their abilities, to capture a major portion of the market share to be profitable. Software organizations are also continuously innovating and improving business operations such as technology, administration, and product development process. Their major concern is the effective use of software assets, thus reducing considerably the development time and cost of software products to capture market segments. For many organizations that deal in wide areas of operation, from consumer electronics, telecommunications, and avionics to information technology, the future of software development is in software product lines. Software product lines are promising, with the potential to substantially increase the productivity of the software development process and emerging as an attractive phenomenon within organizations that deal with the software development. Software product lines involve assembling products from existing core assets, and then growing those core assets continuously as the production proceeds.

The software industry has shown a growing interest in the concept of software product line. One of the major concerns of software development organizations is the effective utilization of software assets, thus reducing considerably the development time and cost of software products. Many organizations that deal in wide areas of operation, from consumer electronics, telecommunications, and avionics to information technology, are using software product lines practice because it deals with effective utilization of software assets. Clements *et al.* [1] report that software product line engineering is a growing software engineering sub-discipline, and many organizations including Philips, Hewlett-Packard, Nokia, Raytheon, and Cummins are using it to achieve extraordinary

gains in productivity, time to market, and product quality. Clements 2] defines the term software product line as a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission, and are developed from a common set of core assets in a prescribed way. Some other terminologies for “software product line” that have been widely used in Europe are “product families,” “product population,” and “system families”.

The concept of a software product line is a comprehensive model for an organization building applications based on common architectures and other core assets [3]. Clements [2] defines the term “software product line” as a set of software-intensive systems sharing a common managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. The Software Engineering Institute (SEI) proposes the Product Line Technical Probe (PLTP) [4], which is aimed at discovering an organization’s ability to adopt and succeed with the software product line approach. The framework is divided into three essential activities: product development, core asset development, and management. *van der* Linden [5] pointed out that in 1995 the Architectural Reasoning for Embedded Systems (ARES) project began in Europe to provide architectural support for developing product families. In the overview of another European project, Engineering Software Architecture, Processes and Platforms for System-Families (ESAPS) [6], a system family is defined as a group of systems sharing a common, managed set of features that satisfy core needs of a scoped domain. The main objectives of system families are to reduce development efforts and to handle the impact of growing system complexity. Ommering [7] introduced another term called “product population”, which is a collection of related systems based on similar technology but with many differences among them.

Northrop [8] stresses that fielding a product line involves core asset development and product development using core assets under the aegis of technical as well as organizational management. The essential activities of the software product line process are core asset development, product development, and management. All the

three activities are linked and are highly iterative in nature. The link among the activities establishes a communication path among them to provide feedback. There is no fixed order of execution of these activities. They can be performed in any order and they give feedback to other activities for their execution. The feedback received at each activity is used to accommodate changes and modify the process. The software product line process can be described in terms of four simple concepts. The four simple concepts that interact with each other are: software assets, a decision model for products, a production mechanism process, and output products that result from software product line activity. Krueger [9] states that software asset inputs are a collection of software assets that can be configured and assembled in different ways to create all of the products in a product line. The decision model for products elaborates the requirements of the products within a product line. Production mechanism and process defines the procedures for assembling and configuring products from existing software assets. Software products are the collection of all products that can be produced from the product line. Associated processes are performed with those basic four concepts for the establishment of a software product line.

Core assets in a software product line may include architecture, reusable software components, domain models, requirement statements, documentation, schedules, budgets, test plans, test cases, process descriptions, modeling diagrams, and other relevant items used for product development. There is no specific definition for core asset inclusion, except that it is an entity used for development purposes. The goal of core asset development is to establish the production capability of developing products [2]. The major inputs to the core asset development activity are: product constraints, styles, patterns, frameworks, production constraints, production strategy, and the inventory of pre-existing assets. The outputs of core assets development are software product line scope, core assets and the production plan. Software product line scope describes the characteristics of the products developed. The production plan gives an in-depth picture how products will be developed from core assets. Core assets are those entities that may be used in the product development. The collection of core assets is termed as core asset repository and the initial state of the core asset

repository depends upon the type of approach being used to adopt software product line approach within an organization.

In product development activity, products are physically developed from the core assets, based on the production plan, in order to satisfy the requirements of the software product line. The essential inputs of product development activity are requirements, product line scope, core assets and the production plan. Requirements describe the purpose of the product line along with functionalities and characteristics of the products developed. Product line scope describes qualification criteria for a product to be included or excluded from software product line based on functional and non-functional characteristics. The production plan describes a strategy to use the core assets to assemble products. A product line can produce any number of products depending upon the scope and requirements of the software product line. The product development activity iteratively communicates with core asset activity and adds new core assets as products are produced and software product line progresses.

Management plays a vital role in successfully institutionalising the software product line within an organization, because it provides and coordinates the required infrastructure. Management activity involves essential processes carried out at technical and organizational levels to support the software product line process. It ensures that necessary resources must be available and well coordinated. The objective of "Technical Management" is to oversee the core asset and product development activities by ensuring that the groups who build core assets and the groups who build products are engaged in the required activities, and are following the processes defined for the product line [4]. Technical management plays a critical role in decision-making about the scope of software product line based on requirements. It handles the associated processes of software development. Northrop [8] summarized the responsibilities of organizational management, which are: structuring an organization, resource management and scheduling, cost control and communication. Organizational

management deals in providing a funding model for the software product line in order to handle cost constraints associated with the project. It ensures a viable and accurate communication and operational path between essential activities of software product line development because the overall process is highly iterative in nature. The fundamental goal of the organizational management is to establish an adoption plan, which completely describes a strategy to achieve the goals of software product line within an organization. The major responsibility of the management is to ensure proper training of the people to become familiar with the software product line concepts and principles. Management deals with external interfaces for smooth and successful product line and performs market analysis for internal and external factors to determine the success factor of software product line. Management performs organizational and technical risk analysis and continues tracking critical risk throughout the software product line development.

van der Linden [16] reports that the term “product family” or “system family” is used in Europe whereas in the United States the term “software product line” is commonly used. As Europeans were working on product family engineering, researchers in the United States founded the SEI’s product line initiative, the major reason for this being that until 1996 the United States and European communities in this field worked independently. The objective of the software product line is to address the specific needs of a given business. Krueger [9] considers that the objective of a software product line is to reduce the overall engineering effort required to produce a collection of similar systems by capitalizing on the commonality among the systems and by formally managing the variation among the systems. A software product line gives an excellent opportunity to establish a production facility for software products based on a common architecture. To capture various market segments, it provides a means for the reuse of assets, thus reducing development time and the cost of software products. The software product line increases the quality and reliability of successive products, thereby gaining the confidence of customers. According to *van der* Linden [16], whenever an organization wants to establish product family development it must keep a number of things under consideration. In Europe, the acronym BAPO [16], is very popular for defining the

process components associated with software product lines. BAPO is considered critical in its consideration of how products resulting from software product lines make a profit. Software engineering, business, management and organizational sciences provide foundations for the concept of software product line engineering, and thus, it has become an inter-disciplinary concept.

1.2 Businesss of Software Product line Engineering

Today, all businesses are experiencing greater competition, and customers' expectations continuously increase as technology advances at an unprecedented rate of growth. The rapid and continual changes common to the present business environment not only affect business itself but also have a profound impact on production. Software is perhaps the most crucial piece of a business entity in this modern marketplace, where important decisions need to be made immediately. Organizations that fail to respond appropriately do not survive longer. The keys to success are in continuously monitoring customers and the competitors and in making improvement plans based on observations and measurements. Business is perhaps the most crucial factor in a software product line, mainly due to the necessities of long-term strategic planning, initial investment, longer payback period and retaining the market presence. Business is perhaps the most crucial dimension in the software product family process, mainly due to the necessities of long-term strategic planning, initial investment, longer payback period, and retention of the market presence.

The "Business" in BAPO is considered critical because it deals with the way the products resulting from software product lines make profits. Bayer *et al* [10] at Fraunhofer Institute of Experimental Software Engineering (IESE) develop a methodology called PuLSE (Product Line Software Engineering) for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. PuLSE-Eco is a part of PuLSE methodology, deals with defining the scope of software product lines in terms of business factors. Pulse-Eco identifies

various activities, which directly address the business needs of software product lines such as: system information, stakeholder information, business objectives and benefit analysis. *van der* Linden *et al.* [11] identify some main factors in evaluating the business dimension of software product line such as: identity, vision, objectives and strategic planning. They classified the business maturity of software product line into five levels in the ascending order: reactive, awareness, extrapolate, proactive and strategic. Clements and Northrop [4] highlight customer interface management, market analysis, funding, and business case engineering as important activities from the perspectives of managing the business of software product line. Kang *et al.* [12] present a marketing plan for software product lines that includes market analysis and marketing strategy. The market analysis covers need analysis, user profiling, business opportunity, time to market and product pricing. The marketing strategy discusses product delivery methods. Toft *et al.* [13] propose "Owen molecule model" consisting of three dimensions of social, technology and business. The business dimension deals with setting up business goals and analyzing commercial environment. Fritsch and Hahn [14] introduce Product Line Potential Analysis (PLPA), which aims at examining the product line potential of a business unit through discussions with managers of the business unit because in their opinion they know the market requirements, product information and business goals of the organization. Schmid and Verlage [15] discuss the successful case study of setting up software product line at Market Maker and highlights market and competitors analysis, vision of potential market segment, and products as significantly important activities. Ebert and Smouts [16] weight marketing as one of the major external success factors of product line approach and further concluded that forecasting, ways to influence market, strong coordination between marketing and engineering activities, are required for gaining benefits from product line approach.

Strategic plans are the focus of an organization's endeavors to accomplish the desired level of achievement in a particular area. Strategic planning starts with elaborating strategic objectives. Niemelä [17] highlighted eight different strategies for adopting software product lines in an organization: minimizing risk, extending market share, maximizing end-user satisfaction, balancing cost and potential, balancing cost,

customer satisfaction and potential, and maximizing potential. Niemelä [17] further concluded that a company has to evaluate the current status of their business, architecture, process, and organizational issues before making a decision about choosing one strategy out of those in order to achieve desired benefits. The software product line process needs resources, which must be delegated in strategic plans. Strategic planning must clearly outline what is to be developed from the software product line in order to gain competitive advantages and capture market segments to achieve strategic targets. Strategic plans are required to maintain organizational wide efforts to identify and exploit attractive long-range business opportunities by having the software product line in practice. The benefits of being the first in the market have long been recognized in the business sector; pioneers often gain a sustainable competitive advantage over followers, because, initially, they are the only solution-providers in a particular market segment. Thus, they usually capture a bigger portion of the market because they were first. It becomes very difficult for successors to gain a share of the market segment, especially in the case of software, where migration to other software is relatively uncommon. The timing for technology-based products entering the market is even more critical for the profitability and competitive position of an organization. The right product at the right time has a high potential of success. Order of market entry is perceived as a crucial business decision, with a long-lasting and profound impact on the performance of an organization in capturing and retaining the market. Appropriate timing to launch a software product into the market is even more essential for software development organizations. Timing is essential in launching a new product from the software product line in order to capture major shares of the market. The order of entry to the market depicts the delivery schedule for the software product family and provides guidelines to developers about development schedules.

Organizations consider brand name a crucial catalyst of business success. A brand is regarded as both a promise of quality to customers and a point of comparison with other products or services. Bennett [18] defined brand as a name, term, sign, symbol, design, or any combination of these concepts that is used to identify the goods and services of a seller. Brand name products generally have high potential for increasing an

organization's business. Branded product serve, as an interface between customers and the organization, and loyalty to a brand is a kind of word-of-mouth advertisement from customers. Brand name strategy has also been successfully adopted in software development. Many successful brands in software, such as Windows®, AutoCAD®, and MATLAB®, successfully retain a significant number of customers, thus capturing a major portion of the market segment. But currently there is gab between software product line engineering and brand name strategy; many different products not originating from one software product line can be plugged under one marketed product line. Windows® is a working example of this scenario. Despite this fact there are successful cases that are using brand name strategy in software product lines concept. The product line of Symbian operating system for mobile phones is an example of this scenario. Long range of products under this brand name is currently successfully installed in the handsets of Nokia, Sony Ericsson, Samsung and Panasonic etc. Jaasksi [19] presented the case study of developing software product line of mobile browsers under the brand name of "Nokia Mobile Browser" at Nokia is also an example of current use of brand name strategy in software product lines. The concept of market orientation provides an advantage over competitors by identifying what customers' want, and then offering products that are different and superior to those offered by competitors. Market orientation deals with the acquisition, sharing, interpretation, and use of information about customers and competitors, both of which have a significant impact on the performance of the business. Birk *et al.* [20] defines market orientation in context of software product lines as whether the organization targets a specific market segment without a specific customer in mind or addresses individual customer projects. The software product line deals with developing a considerable number of products to capture various market segments, thus providing justification for a product line. Market orientation provides imperative information about the concerns and requirements of customers, which needs to be accommodated in the successive products from a product line. Pulse-Eco [21] illustrates various activities associated with market orientation for successful adoption of software product lines concept in an organization. It considers collecting and analyzing stakeholders' information is helpful in defining the product line scope.

Business success is highly dependent on the extent to which customers are satisfied with an organization's product and services, as well as how they win the loyalty of customers by improving their relationships management. Relationships management plays a significant role in successful software product line development. Excellent working relationships with customers allow the developers to improve the performance and functionalities of successive products from the product line by better understanding the customers' requirements and learning about market trends from the end users. The software product line can play a significant role in the business vision because it tends to produce long-term benefits to the organization. A clear statement about business vision will guide practitioners of the software product line to establish a production facility in order to meet the future goals of the organization. By including the software product line in the business vision, an organization can stream line its business operations in order to capitalize on its market audience for profitable venture. Wijnstra [22] concluded that a complete business roadmap is needed to describe what is expected from the software product lines in the years to come and how it will fit in the plan for the release of new products. The key to a successful business in today's competitive environment is innovation. Organizations are continuously adopting innovations in major areas of business operations, such as technology, administration, and production processes. . Organizations with designs on capturing a major share of the market, in order to increase business, spend heavily on research and development. Business objectives influence research and development efforts because the order of a product's entry into the market can make a significant difference in achieving strategic goals. Thus, research and development in technology, administration, processes, and product produce enduring results. The software product line is a relatively new concept, and a lot of research and development in process definition and development methodology is in progress. The research is occurring at various levels of industry and academia to improve the process and product development activity of the software product line for the successful industrialization of this valuable concept. Organizations are trying to institutionalize this concept in innovative ways to make the most effective use of it. Böckle [23] highlighted some measures of innovation management in software product line organizations, which include a planned innovation process, clear roles and

responsibilities definition for innovation management structure. Böckle [7] further stressed that the evolution of the product portfolio, platform, variability model, and reference architecture shall be planned with further innovations in mind.

The business of software product line engineering has a profound impact on the long term planning and vision of the organization in the market place. The significance of the business factor in product line engineering requires a better understanding of various non-software development factors which originates from business theory. This makes software product line engineering multi-disciplinary paradigm which needs contributions from many experts in different areas of knowledge and expertise. Although business has always been highlighted as one of the critical success factors in product line engineering but has given least attention by product line engineering community to streamline the concept and integrate with software development efforts. Some of the leading areas of core research in software product line engineering and business factors are as follows:

- Development of a business case and methodology to evaluate the significance in terms of cost and benefits for an organization.
- An organizational wide economic model for developing and managing software product line engineering, emphasizing return on investment.
- A methodology to develop production plan for the resulting products and allocation of resources.
- The issues of translating the business requirements into product line requirements which involves from non-technical (business group) to technical (architecture group).
- The role and impact of strategic planning of the organization in developing and managing software product line. How organization can achieve its strategic goals using the product line approach.
- Decision planning and implementation in allocating and committing resources to achieve the long-range business goals.
- Marketing plans to identify and exploit attractive long-range business opportunities.
- How market orientation provides imperative information about the concerns and

requirements of customers, which need to be accommodated in successive products from a product line.

- Customer orientation enables an organization to develop customer-centered products. How this information assists in the domain- and application-engineering activities of the software product line development to capture market segments.
- Evaluation of appropriate timing to launch a software product into the market from product line in order to maximize the profit.
- The role of business vision in managing and developing software product line.
- Knowledge management of customers, marketing and competitors.
- Methodology to evaluate the business performance of the organization dealing with software product line engineering.
- Methodology to evaluate the practice of various key business factors in organization.

1.3 Institutionalization of Software Product Line Engineering

The "Organization" in BAPO is considered critical because it deals with the way the organization responds, adopts and institutionalizes this concept. Institutionalization is the process by which a significantly new structure or practice is incorporated into a system of existing structures and practices [24]. Clements and Northrop [15] elaborate the institutionalization of software product line in an organization from the perspectives of product development and core assets development. Institutionalizing a software product line from the aspects of product development process anticipate the product development as a routine and predictable activity in an organization to achieve the product line goals of the organization. Clements and Northrop [4] emphasis that institutionalizing a software product line from the perspectives of managing and developing a core assets repository for software product line involves improving the processes that are associated with building, maintaining, and evolving the core assets and making those processes a part of standard organizational practice. In short institutionalization of software product lines refers to the wide acceptance of the concept in the roots of the organization. It involves integrating or improving the processes within

organization that are associated with a product line infrastructure, and introducing those processes as a part of organizational character. The whole institutionalization process involves an organizational level culture and strong commitments in acquiring knowledge, skills and motivations to effectively initiate, launch and manage software product lines. Institutionalization of software product lines require that the concept has been entrenched at all levels of the organization, and it is supported with a necessary infrastructure of organizational wide guidelines, required training, and required resources.

Successfully institutionalization of software product line in an organization has a profound impact on the product development behavior of the organization. It changes the mindset of the organization from single system development to a family of software products. The organizational theory focuses on the design and structures of the organization dealing in software product line. The organizational behavior aims at understanding the behavior, attitude and performance of the people. Software product line requires enriching this concept within the roots of the overall organizational behavior. Organizational management plays a vital role in successfully institutionalizing software product line within an organization because it provides and coordinates the infrastructure required. Initiating and launching a software product line within an organization to gain benefits out of this approach is not sufficient. The alignment of organizational theory, organizational management, and organizational behavior are required in the process of institutionalization of software product line in an organization. Thus, organizational factors play a key role in institutionalizing software product lines within an organization. Software product line is an inter-disciplinary concept, which has its roots in software engineering, business, management and organizational sciences. The organization in the business of software product line has to deal with multiple organizational factors in addition to their efforts in software development in order to institutionalize software product line, which in turn has the potential to achieve maximum benefits out of this approach.

The organizational dimension is perhaps the least addressed area in software product line research due to relatively a new concept in software engineering paradigms. Much of the efforts have been spent on process, architecture and business aspects of the product line. Some scenarios of organizational structure for software product line are presented. The researchers generally highlight that domain-engineering unit and several application-engineering units are required from organizational structure standpoint. Bosch [25] presents four organizational models for software product lines: development department, business units, domain engineering units, and hierarchical domain engineering units. Bosch [25] also points out a number of factors that influence the organizational model such as geographical distribution, project management maturity, organizational culture and the type of systems. Macala *et al.* [26] report that software product line demands careful strategic planning, a mature development process, and the ability to overcome organizational resistance. Dikel *et al.* [27] share their experiences about initiating and maintaining software product lines at Nortel and discuss organizational, management and staffing issues grouped into a set of six organizational principles which they believe are critical in the long-term success of a software product line. Jacobsen *et al.* [28] focus on roles and responsibilities of personals within organizations dealing with software product lines. Mannion [429] elaborates that the management issues, organizational structure, culture and learning in context of successfully adopting the concept of software product line engineering needs close attention. Koh and Kim [30] concludes that all members of an organization experience and share their own success stories under existing processes and organizational structure in order to successfully adopt software product line approach. Clements and Northrop [4] discuss organizational issues of software product line and identified four functional groups, i.e. the architecture group, the component-engineering group, the product line support group and the product development group. The organizational dimension of software product lines deal with the way the organization is able to deal with complex relationships and many responsibilities [5]. Toft *et al.* [31] propose “Owen molecule model” consisting of three dimensions of organizational, technology and business. The organizational dimension of Owen molecule model deals with teams hierarchy, individual roles, operational models, individual interaction and

communication etc. Introducing software product line practice to an organization significantly impacts the entire organization by fundamentally changing development practices, organizational structures, and task assignments [32]. Bayer *et al.* [33] at Fraunhofer Institute of Experimental Software Engineering (IESE) develop a methodology called PuLSE (Product Line Software Engineering) for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. PuLSE-BC is a technical component of PuLSE methodology and it deals with the ways to baseline organization and customized the PuLSE methodology to the specific needs of the organization. One of the support components of PuLSE is organization issue, which provide guidelines to set up and maintain the right organization structure for developing and managing product lines. According to Birk *et al.* [32] introducing product line development to an organization can fundamentally change the development practices, organizational structures, and task assignments. These changes can in turn impact team collaboration and work satisfaction. Verlage and Kiesgen [34] report the case study of successful adoption of software product line and conclude that organizational structure and change management are significantly important areas of concern.

Organizational culture refers to the working environment in an organization. Some of the key process activities of software product line engineering, including domain engineering, software product line requirements engineering, commonality and variability management and business case engineering etc., require a lot of team effort, group discussion and innovation. The studies in organizational culture highlight two types of cultures: closed and open. In closed organizational cultures, the decisions are made at the higher levels and are directly dictated to the lower levels without considering the views and observations of most employees. In contrast, open organizational cultures make decisions on the basis of discussions and employee involvement. Software product line engineering requires a culture of openness, where employees have the chance to participate in discussions and have the power to express their views. For example, variability management is one of the critical process elements that require an active involvement from various parts of the organization, such as the

business unit and the development unit, to specify areas for expansion in the product line architecture and introducing product specific functionalities. An organizational culture that supports teamwork, sharing of experiences, innovation and learning has relatively greater potential institutionalizing software product line engineering. Particularly, an organization with a culture that supports the reusability of software assets is more likely to succeed in moving from single product development to a systematic line of products.

Organizational commitment concerns the willingness of individuals and groups to achieve the long term strategic objectives of an organization. The payback period of software product line engineering is relatively longer than the single product development approach. Consequently, this transitional period requires a strong commitment from individuals, groups and management to adopt the software product line engineering concept and to exercise patience with its development process. The organizational policies such as business vision and strategic planning must highlight the concept of software product line engineering as a priority in order to reflect the organizational commitments. Furthermore, these policies must be well communicated to the employees so that they understand the significance of this approach in achieving the organizational goals. As well, the success of any long-term strategy in an organization necessitates the commitment of its employees. The management has to create a positive working environment in order to increase the level of employee commitment. Such an environment can be achieved through well-defined job placement, promotion strategy, appreciation and reward system, job security and competitive compensation.

An organization is the planned coordination of activities of a number of people for the achievement of some common, explicit purpose or goal, through a division of labor and function, and through a hierarchy of authority and responsibility [35]. Organizational theories provide guidelines for developing organizational structures in order to accomplish the goals of a company. Wilson and Rosenfeld [36] define organizational structure as the established pattern of relationships between the parts of an

organization, outlining communication as well as control and authority. According to Gordon [37] organizational structure refers to the delineation of jobs and reporting relationships in an organization and coordinates the work behavior of employees in accomplishing the organization's goals. The structure of an organization is generally not a static phenomenon, since organizations tend to change their structures under the circumstances of changing goals or technologies.

The rapid and continual changes common to the present technological environment necessitate that organizations adopt changes through a well defined change management plan. Beckhard and Harris [38] consider organizational change as a movement from the present state of the organization to some future or target state. Furthermore, Todd [39] defines change management as a structured and systematic approach, which provides a conceptual framework that encompasses strategy, politics, people and process. Cao et al. [40] observe that organizational change shows the diversity of an organization, and it also illustrates the integration of technical and human activities that have interrelated functions in the organization. The successful implementation of any process methodology ultimately depends on how people perceive that change. A certain degree of resistance is quite normal when a new technology is introduced to an organization. However, this resistance will disappear if people understand that the change is positive and is in their best interest as well as that of the organization. Effective change management therefore depends partly on how the strategy is communicated to the people responsible for the implementation.

When people interact with each other, the potential for conflict is present. This potential exists in different areas, as it could be either personal or task related. Walls and Callister [41] maintain that conflict is a process in which one party perceives that its interests are being opposed or negatively affected by another party. Conflict management consists of diagnostic processes, interpersonal styles, negotiating strategies, and other interventions that are designed to avoid unnecessary conflict and to reduce or resolve excessive conflict [42]. Hellriegel et al. [43] introduce four basic forms of conflicts in an organization: goal, cognitive, affective, and procedural.

Moreover, Jehn [44] distinguishes between two kinds of intra-group conflict: task conflict and relationship conflict. Task conflict is a perception of disagreement among group members or individuals regarding the content of their decisions. It involves differences in viewpoints, ideas and opinions, whereas relationship conflict is a perception of interpersonal incompatibility and includes annoyance and animosity among individuals [45].

In the software product line engineering, organizational learning can be classified into two domains: external and internal. External learning involves necessary knowledge about customers, competitors, external environments and market segments. This knowledge is necessary in order to effectively utilize the product line by exploiting product characteristics. The domain engineering, the product line requirements and the business case engineering, etc. require that the organization has established procedures and a means to acquire external learning. Overall, this type of learning helps an organization to capture a major market share. Internal learning, on the other hand, requires acquiring, transferring and sharing a software product line methodology, ideas for process improvement and an understanding of the cross functional requirements of product lines in individuals, groups and the organization. Learning is a continuous process, especially for organizations that attempt to institutionalize software product lines. In particular, learning from experience and mistakes further facilitates improvement in the software product line engineering process.

One of the major concerns of software development organizations is the effective utilization of software assets, which has the potential to considerably reduce the time and cost of developing software products. Software is perhaps the most crucial piece of business entity in this modern marketplace, where important decisions need to be made immediately. The studies in organizational behavior help in understanding how people, as individuals and groups, deal with managing and developing product line engineering in an organization. The relatively longer payback period of software product line engineering requires a consistency in organizational behavior in order to achieve the

strategic objectives of the organization. Establishing a software product line requires setting up the internal structure of the organization and other supporting mechanisms, such as coordination and communication. The concept of a software product line entails a structure of overlapping processes rather than fixed and static ones. The theoretical foundations of this concept divide the overall engineering process into two broad areas, application and domain engineering, and involve a stronger coordination and communication between them. The identification and mapping of the roles to the engineering processes requires interpretation and action from management. Verlage and Kiesgen [46] present a case study documenting the successful implementation of software product lines in their organization. As a result, they report that the roles and mapping of the roles to the processes are not fixed; rather, they are interchangeable, or more precisely, dynamic. The organizations that have well defined structures incorporating clearly identified roles of individuals, in addition to strong coordination and communication, are more likely to institutionalize a software product line in comparison to the organizations with structures not supporting coordination and communication. The process of evolving from single product development to a line of products is a significant change in an organization. During this procedure, almost every software development activity needs to be changed. For example, in the case of requirements engineering, an organization has to deal with product specific requirements engineering as well as product line specific requirements engineering. The product specific requirements engineering involves identifying the variability among products, whereas product line requirements engineering entails detecting the commonality among products. Furthermore, there is need to introduce trade off analysis for commonality and variability management. Introducing a new practice such as a product line is relatively difficult in the existing setup of an organization, especially if it is not being introduced with a proper change management plan. Even the best strategy is bound to fail if there is a consistent resistance to innovation and new technology from within the organization. Organizations that communicate the importance of this change via clear guidelines and the establishment of a road map for their employees are more successful in institutionalizing software product lines.

Although organization has always been highlighted as one of the critical dimension in product line engineering but has given least attention by product line engineering community to streamline the concept and integrate with software development efforts. Some of the leading areas of core research in software product line engineering and organization dimension are as follows:

- Conflict management planning to resolve and handle conflict in an organization dealing with product line approach.
- The organizational structure needs to be explored in order to provide a suitable structure which defines specific roles and responsibilities of object. Many traditional organizational structures have been studied for the application in product line environment but they need to be enhanced to accommodate the product line concept.
- Organizational learning procedures and guidelines to adopt product line approach and switching from traditional product development for single system to line of products.
- A study to develop strategies to incorporate and monitor the organizational communication process.
- Change management plans and implementation procedures.
- Effective utilization of core assets and their management.
- Developing plans to start and maintain an infrastructure for product line development.
- Knowledge management in organization for effective use and dissemination of knowledge across the boundaries of the organization.
- Human resource management across organization to provide necessary resources for product line infrastructure.
- Inter group trust management to enhance the productivity of the product line process.
- A methodology to assess the organizational dimension of software product line process and to define improvement plans.

1.4 Software Product Line Architecture

Software architecture has been a key area of concern in software industry due to its profound impact on the productivity and quality of software products. This is even more crucial in case of software product line, because it deals with the development of a line of products sharing common architecture and having controlled variability. Software architecture has a history of evolution and over a decade the software industry is observing and reporting refinements and advancements. Now the trends in software architecture for single product development have turned into software product line architecture for line of resulting products.

Software architecture is the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution [47]. Software architecture has a long history of evolution and in this modern age this transformation leads towards software product line architecture, where the concern is not a single product development rather the focus is on multiple product development by sharing the same architecture. Pronk [48] defines software product line architecture as an ultimate reuse in which the same software in reused for an entire class of products with only minimal variations to support the diversity of individual product family members. According to Jazayeri et al. [49] software product line architecture defines the concepts, structure, and texture necessary to achieve variation in features of variant products while achieving maximum sharing parts in the implementation. Mika and Tommi [50] further elaborate that software product line architecture can be produced in three different ways: from the scratch, from existing product group, or from a single existing product. Software product-line architecture is a powerful way to control the risks and take advantage of the opportunities of complex customer requirements, business constraints, and technology, but its success depends on more than technical excellence [51]. The software product line architecture captures the central design of all products and allows for the expression of variability and commonalities of the product instances, the products are instantiated by configuring the architecture and customizing components in an asset library [52]. The “Architecture” in BAPO is considered critical because it deals with the technical means to build an

architecture that is aimed to share by a number of products from the same family. Van der Linden et al. [11] identify some main factors in evaluating the architecture dimension of software product line such as: software product family architecture, product quality, reuse levels and software variability management and classify the architecture maturity of software product line into five levels in the ascending order: independent product development, standardized infrastructure, software platform, variant products and self-configurable products. Birk et al. [53] conclude that explicit documentation of the software product line architecture, platform features, and generic interfaces is important for the product teams to understand the reusable assets.

The methodologies developed for software product line development either in general or specific to particular application domain consider domain engineering as an integral activity of the overall product line process and has profound impact on building the architecture for the product line. Bayer et al. [54] at Fraunhofer Institute of Experimental Software Engineering (IESE) develop a methodology called PuLSE (Product Line Software Engineering) for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. PuLSE-DSSA is a part of PuLSE methodology, which deals with developing the reference architecture for software product line. Knauber et al. [55] further elaborate that the basic idea of PuLSE-DSSA is to incrementally develop reference architecture guided by generic scenarios that are applied in decreasing order of architectural significance. Researchers at Philips® have developed Component-Oriented Platform Architecting (CoPAM) [56] method for the software product lines of electronics products. CoPAM assumes a strong correlation among facts, stakeholder expectations, any existing architecture and the institutions about possible architects in developing software product line architecture. Weiss and Lai [57] discuss the development of Family-Oriented Abstraction Specification and Translation (FAST) method for software product line process and successful use at Lucent Technologies®. FAST method covers a full software product line engineering process with specific activities and targeted artifacts. It divides the overall process of software product line into three major steps of domain qualification, domain engineering and application engineering. Researchers at IESE developed

another methodology called KobrA [58], which defines software product line engineering process with activities and artifacts. The process of software product line engineering is divided into framework engineering and application engineering with their sub steps. These steps cover the implementation, releasing, inspection and testing aspects of product line engineering process. Kang et al. [59] propose a Feature Oriented Reuse Method (FORM), which is an extension to the Feature-Oriented Domain Analysis (FODA) method to cover the aspects of software product lines. FORM provides a methodology to use feature models in developing domain architectures and components reusability. Researchers at the VTT technical research centre of Finland have developed Quality-driven Architecture Design and Quality Analysis (QADA) method for developing and evaluating software architectures with emphasis on product line architecture. Matinlassi [60] has reported the comparison of software product line architecture design methods including CoPAM, FAST, FORM, KobrA and QADA, and concluded that these methods do not seem to compete with each other, because each of them has a special goal or ideology.

The concepts of commonality and variability management inherently belong to domain engineering are gaining popularity over time due to extensive involvement in software product line concept. According to Coplien et al. [61] commonality and variability analysis gives software engineers a systematic way of thinking about and identifying the product family they are creating. Commonality management deals with the way features and characteristics that are common across the products belong to same product line whereas variability management is other way round. Variability management handles the way the variable features and characteristics are managed in different products of the a product line. Software product line requires systematic approaches to handling commonality and variability and the core of successful software product line management largely relies on effective commonality and variability management. Kang et al. [62] discuss the use of feature models to manage commonality and variability in software product line. Lam [63] presents variability templates and variability hierarchy based variability management process. Thompson and Heimdah [64] propose a set based approach to structure commonalities and variability in software product lines. Kim

and Park [65] describe the goal and scenario driven approach for managing commonality and variability on software product line. Ommering [66] observes that the commonalities are embodied in an overall architecture of software product line, while the differences result in specifying variation points and by filling those variation points, individual products can be derived. Other researchers [67] [68] [69] have stressed that the software architecture for a product family must address the variability and commonality of the entire set of products.

Requirements modeling have always been a key architecture concern in software devolvement, because it provides a better understanding of the requirements of the architecture and allows visualizing the interconnection of various sub-units. Since the popularity of object oriented design, Unified Modeling Language (UML) has become an industry standard, many researchers have attempted to introduce UML in visual modeling of software product line architecture by presenting enhancement in the current state. Birk et al. [70] stress that the organization dealing with software product line architecture should describe the architecture using well-established notations such as UML and the architecture description should cover all relevant architectural views and use clearly defined semantics. Gomma and Shin [71] describe a multiple-view meta-modeling approach for software product lines using the UML notation, which defines the different aspects of a software product line such as: the use case model, static model, collaboration model, state chart model, and feature model. Zuo et al. [72] present the use of problem frames for product line engineering modeling and requirements analysis and demonstrate some additional notation to support the requirements management and variability issues in product line problem frames. Dobrica and Niemelä [73] discuss how UML standard concepts can be extended to address the challenges of variability management in software product line architecture and introduce some extensions in UML standard specification for the explicit representation of variations and their locations in software product line architectures, this work is based on previously mentioned QADA methodology. Eriksson et al. [74] describe a product line use case modeling approach named PLUSS (Product Line Use case modeling for Systems and Software engineering) and conclude that PLUSS performs better than modeling

according to the styles and guidelines specified by the Rational Unified Process (RUP) in the current industrial context.

Software architecture evaluation techniques are generally divided into two groups: qualitative evaluation and quantitative evaluation. Qualitative techniques include scenarios, questionnaires, checklists etc. Quantitative techniques cover simulations, prototypes, experiments, mathematical models, etc. Etxeberria and Sagardui [75] highlight the issues that can arise when evaluating product line architecture versus evaluating single system architecture, including classifications of relevant attributes in product line architecture evaluation, new evaluation techniques. Graaf et al [76] present a scenario based software product line evaluation technique, which provides guidelines to adapt scenario-based assessment to software product line context. Using the qualitative technique of software architecture evaluation Hoek et al [77] put forward service utilization metrics to assess the quality attribute of software product line architecture. Zhang et al [78] study the impact of variants on quality attributes using a Bayesian Belief Network (BBN) and design a methodology applicable to software product line architecture evaluation. Lange and Kang [79] propose a product-line architecture prototyping approach using network technique to assess issues related to software product line architecture evaluation. Gannod and Lutz [80] define an approach to evaluating the quality and functional requirements of software product line architecture. Niemelä et al. [81] discuss the basic issues of product family architecture development and present evaluation model of software product family in industrial setting.

Domain engineering has a pivotal role in the process of software product line. The inception phase of software product line starts with conducting a comprehensive domain engineering in defining and narrowing down the scope of product line, which identifies the characteristics of the product line and the products that comprise the product line. The product line engineering envisages the domain engineering into set of three activities: domain analysis, domain design and domain implementation. Domain analysis concentrates on understanding the domain and providing a foundation to

domain design, which is an early sketch of the architecture of product line. Domain analysis not only defines the boundaries of the software product line scope but also helps in performing the commonality and variability analysis for the product line. Domain implementation further helps in developing the core architecture of software product line by specifying components and their inter-connections. The activities of domain engineering invariably helps in carrying out commonality and variability analysis. The domain engineering helps in defining the common and variable parts of the software product line requirements, thus explicitly identifying the commonality and variability of the envision products. The software product line requires a strong coordination among domain engineering and application engineering. The domain engineering helps in establishing an infrastructure for software product line and the application engineering uses the infrastructure and develops products using core assets.

Requirements modeling provide us with the facility to model the requirements graphically so that requirements can easily be understood by various stakeholders Requirements modeling helps in understanding the requirements of the products and in further elaborating the functionalities and tradeoffs. Software product line needs to elaborate the requirements at two levels: product line level and individual product level. The product line level requirements envisage the commonality among products whereas individual product level requirements represent the variability. Modeling requirements in the context of software product line architecture helps in identifying and specifying the extension points called variation points. It decomposes and specifies the architecture into set of features with their dependency. Requirements models translate the requirements of the targeted market segment and specify the implementation views of the business case. Much of the work on requirements modeling for software product line has concentrated on establishing an extension in the current available modeling techniques like UML and feature diagrams.

Product requirements in software product line are composed of a constant and a variable part. The constant part comes from product line requirements and deals with features common to all the products belonging to a family. The variable part represents

those functionalities that can be changed to differentiate one product from another. This causes the significance of commonality and variability management in software product line. Commonality among products of a software product line is an essential and integral characteristic of product line approach that paves a way to maximize reusability. The products share the common architecture and they are developed out of common core assets. The commonality management takes much if its' input from domain engineering and those inputs are further elaborated and clearly specified using requirements modeling approaches. The extent of commonality among products is a design decision based on business case engineering and targeted market segment. In order to maximize the reusability of software assets, it is generally recommended to have as much commonality as possible.

Variability among products of a software product line is necessary because it makes them a separate business entity. The products from a software product line may vary from each other's in quality, reliability, functionality, performance and so on, but as they share a common architecture so the variation should not be that much high so that they become out from the scope of a single product line. Those variations must be handled systematically to accommodate changes in various versions of the product. The objective of variability management is to identify, specify and document variability among products in the applications of product line. Software product line architecture represents variability by specifying the variation points, which can be exploited at application engineering level by accommodating the design decisions based on the business case. The variability in products usually results from internal and external factors. The internal factors have their roots in refining the architecture whereas external factors accommodate the market and customers expectations. The introduction of variable features in a product from a software product line is a strategic decision based on market segment. The introduction of variable features in the successive products out of product line also provides a justification for setting up a product line in the organization as well because it helps in attracting new customer and retaining the current one. Fitting the components into the product without tailoring it is the easiest task, but some time we need to make certain changes in the component to meet the

requirements for a particular product. Every component present in the core assets must clearly define the variability mechanism to be used in order to tailor them for reuse. The significance of commonality and variability management in software product line architecture and the overall performance of the software product line require tool support, which needs the attention of researchers.

Software artifacts management play significant role in the process of development, maintenance and reuse of software. Software product line architecture is one of the critical artifacts of software product line approach, and all the resulting products share this common architecture. The architectural artifacts provide in-depth knowledge about various views, levels of abstractions, variation points, components identification, component behavior and their inter-connection. It has been a general trend in software industry to represent and document architecture using notations and languages such as Architecture Description Language (ADL). Software product lines currently lack an architecture description language to represent the software product line architecture in large. These documentations such as domain analysis, domain design, domain testing, requirements modeling provides inputs to software product line architecture. The configuration management issues of software product line artifacts are imperative in software product lines as it deals with a number of resulted products with different versions and releases as well as several number of core assets with different versions. The concept of configuration management currently used in software industry deals with a single project, or more precisely with a single product, and on the opposite software product line deals with a set of products. Therefore a multi dimensional approach of configuration management should be adopted to cope up with the issue. Configuration management of software product line is a research area where not much work has been done and requires an immediate attention of researchers.

- Quality is a major issue for family of products. Like a single product, software quality is fundamental to a family of products' success. Core and product architectures of family of products are expected to help achieve the required quality attributes. However, one of the key challenges in designing software

architectures for core and individual products with respect to the desired level of different quality attributes is that the quality attributes have been found very hard to define, describe and understand. This aspect has very strong subjective interpretation. That is why it is vital to systematically elicit and precisely define quality aspects of a family of products in order to help design appropriate architectures. There are a number of classifications of quality attributes. McCall listed a number of classifications of quality attributes developed by software engineering researchers including himself [82]. A later classification of software quality is provided in [83]. However, none of them has been proven sufficient to define, specify, and model different levels of quality attributes required in different products of a family. There is a vital need for developing appropriate approaches to eliciting, specifying, and modeling quality attributes to be supported by software architectures of a family of products.

- Designing and evaluating software architectures of a family of systems involves complex and knowledge intensive tasks. The complexity lies in the fact that tradeoffs need to be made to satisfy current and future requirements of a potentially large set of stakeholders, who may have competing vested interests in architectural decisions. The knowledge required to make suitable architectural choices is broad, complex, and evolving, and can be beyond the capabilities of any single architect [84]. Due to the recognition of the importance and far reaching influence of the architectural decisions, several approaches have been developed to support architecting processes. Examples are the Generic Model for architecture design [85], Attribute-driven design [86], Architecture Tradeoff Analysis Method (ATAM) [87], 4+1 views [88], Rationale Unified Process (RUP) [89] and architecture-based development [90]) While these approaches help to manage complexity by using systematic approaches to reason about various design decisions, they provide very little guidance or support to capture and maintain the details on which design decisions are based, along with explanations of the use of certain types of design constructs (such as patterns, styles, or tactics). Such information represents architecture knowledge, which

can be valuable throughout the software development lifecycle [91]. We assert that the lack of a systematic approach to capturing and sharing architectural knowledge may preclude organizations from growing their architecture capability and reusing architectural assets. Moreover, the knowledge concerning the domain analysis, architectural patterns used, design alternatives evaluated and design decisions made is implicitly embedded in the architecture and/or becomes tacit knowledge of the architect [92]. Hence, one of the key challenges in successfully development and evolving software architectures is the provision of suitable infrastructure for capturing, maintaining, and sharing architectural knowledge and rationale underpinning key architectural design decisions.

- Apart from the challenge of devising optimal architectural solutions, specifying the architecture and interfaces of component-based family of systems is a difficult task, which poses several kinds of challenges. Fro example, industries heavily dependent upon on the component-based software engineering, like automotive, Usually OEMs (Original Equipments Manufacturers) have to provide an overall architecture of the automotive systems in its cars and distribute these to potential suppliers of systems and components who do the implementation. The AUTOSAR standard is a move to establish an open standard for automotive embedded electronic architecture. AUTOSAR tries to achieve modularity, scalability transferability and reusability of functions. However, even if the architecture and components are specified using AUTOSAR, there is still no checking of conformance or conformance validation. We assert that there is a need for specific methods and tools to validate that those implementations actually conform to the specifications and that the combination of the various implementations conforms to the OEMs' specifications.

- Architecture and interface specification is another big challenge in software product line engineering in general and software product line engineering for automotive systems in particular. There is general lack of suitable and reliable methods to accurately and sufficiently provide interface specifications. This is

also one of the key research challenges in the context of increasing trend of global software development.

List of References

[1] Clements, P.C., Jones, L.G., Northrop, L.M. and McGregor, J.D.: Project Management in a Software Product Line Organization. IEEE Software 22 (5) 54-62 (2005)

[2] Clements, P.C.: On the Importance of Product Line Scope, Proceedings of the 4th International Workshop on Software Product Family Engineering 69-77 (2001)

[3] T. Wappler, Remember the basics: key success factors for launching and institutionalizing a software product line, in: Proceedings of the 1st Software Product Line Conference, 2000, pp. 73-84.

[4] P.C. Clements, L.M. Northrop, Software product lines practices and pattern, Addison Wesley, 2002.

[5] F. *van der* Linden, Software product families in Europe: The Esaps & Café projects, IEEE Software 19 (4) (2002) 41-49.

[6] ESAPS Project (1996) available from: http://www.esi.es/en/Projects/esaps/overview.html

[7] R.V. Ommering, Beyond product families: building a product population, in: Proceedings of the Conference on Software Architectures for Product Families, 2000, pp.187-198.

[8] L.M Northrop, SEI's software product line tenets, IEEE Software 19 (4) (2002) 32-40.

[9] C.W. Krueger (2004) Basic software product line concepts, Available from:

http://www.softwareproductlines.com/introduction/concepts.html

[10] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J.M. DeBaud, PuLSE: a methodology to develop software product lines, in: Proceedings of the 5th ACM SIGSOFT Symposium on Software Reusability, 1999, pp. 122-131.

[11] F. *van der* Linden J. Bosch, E., Kamsties, K. Känsälä, H. Obbink, Software product family evaluation, in: Proceedings of the 3rd International Conference on Software Product Lines, 2004, pp. 110-129.

[12] K.C. Kang, P. Donohoe, E. Koh, J. Lee, K. Lee, Using a marketing and product plan as a key driver for product line asset development, in: Proceedings of the 2nd International Conference on Software Product Lines, 2002, pp.366-382.

[13] P. Toft, D. Coleman, J. Ohta, A cooperative model for cross-divisional product development for a software product line, in: Proceedings of the 1st International Conference on Software Product Lines, 2000, pp. 111-132.

[14] C. Fritsch, R. Hahn, Product line potential analysis, in: Proceedings of the 3rd International Conference on Software Product Lines, 2004, pp. 228-237.

[15] K. Schmid, M. Verlage, The economic impact of product line adoption and evolution, IEEE Software 9(4) (2002) 50-57.

[16] C. Ebert, M. Smouts, Tricks and traps of initiating a product line concept in existing products, in: Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 520-525.

[17] E. Niemelä, Strategies of product family architecture development, in: Proceedings of the 9th International Conference on Software Product Lines, 2005, pp. 186-197.

[18] P.D. Bennett, Dictionary of marketing terms, American Marketing Association, 1988.

[19] A. Jaaksi, Developing mobile browsers in a product line, IEEE Software 19(4) (2002) 73-80.

[20] G. H Birk, John, I. Schmid, K. von der Massen T., Muller K., Product line engineering, the state of the practice, IEEE Software 20(6) (2003) 52-60.

[21] P. Knauber, D. Muthig, K. Schmid, T. Wide, Applying product line concepts in small and medium-sized companies, IEEE Software 17(5) (2000) 88-95.

[22] J.G. Wijnstra, Critical factors for a successful platform-based product family approach, in: Proceedings of the 2nd International Conference on Software Product Lines, 2002, pp. 68-89.

[23] G. Böckle, Innovation management for product line engineering organizations, in: Proceedings of the 9th International Conference on Software Product Lines, 2005, pp. 124-134.

[24] W. R. Scott, Institutions and organizations, Sage Publications, CA, 1995.

[25] J. Bosch, Software product lines: organizational alternatives, in: Proceedings of the 23rd International Conference on Software Engineering, 2001, pp. 91-100.

[26] R.R.. Macala, L.D. Jr. Stuckey and D.C. Gross, Managing domain-specific, product-line development, IEEE Software 13 (3) (1996) 57-67.

[27] D. Dikel, D. Kane, S. Ornburn, W. Loftus and J. Wilson, Applying software product-line architecture, IEEE Computer 30 (8) (1997) 49-55.

[28] I. Jacobsen, M. Griss and P. Jonsson, Software reuse - architecture, process and organization for business success, Addison Wesley, 1997.

[29] M. Mannion, Organizing for software product line engineering, in: Proceedings of the 10th International Workshop on Software Technology and Engineering Practice, 2002, pp. 55 –61.

[30] E. Koh and S. Kim, Issues on adopting software product line, in: Proceedings of the 11th Asia-Pacific Conference on Software Engineering, 2004, pp. 589.

[31] P. Toft, D. Coleman and J. Ohta, A cooperative model for cross-divisional product development for a software product line, in: Proceedings of the 1st International Conference on Software Product Lines, 2000, pp. 111-132.

[32] G. H Birk, I. John, K. Schmid, T. von der Massen and K. Muller, Product line engineering, the state of the practice, IEEE Software 20 (6) (2003) 52-60.

[33] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen and J.M. DeBaud, PuLSE: a methodology to develop software product lines, in: Proceedings of the 5th ACM SIGSOFT Symposium on Software Reusability, 1999, pp. 122-131.

[34] M. Verlage and T. Kiesgen, Five years of product line engineering in a small company, in: Proceedings of the 27th International Conference on Software Engineering, 2005, pp. 534 – 543.

[35] E. H. Schein, Organizational psychology, Prentice Hall, 1988.

[36] D.C. Wilson and R.H. Rosenfeld, Managing organizations, McGraw-Hill, 1990.

[37] J.R. Gordon, Organizational Behavior: A diagnostic approach, Prentice Hall, New Jersey, 2002.

[38] R. Beckhard, and R.T. Harris, Organizational transitions: managing complex change, Addison-Wesley, 1987.

[39] A. Todd, Managing radical change, Long Range Planning 32 (2) (1999) 237-44.

[40] G. Cao, S. Clarke, and B. Lehaney, A systematic view of organizational change and TQM, The TQM Magazine 12 (3) (2000) 186-93.

[41] J.A. Walls and R.R. Callister, Conflict and its management, Journal of Management 21 (3) (1995) 515-558.

[42] J. Kottler, Beyond blame: A new way of resolving conflicts in relationships, Jossey-Bass, San Francisco, 1994.

[43] D. Hellriegel, J.W. Jr. Slocum, R.W. Woodman and N.S. Bruning, Organizational behavior, ITP Nelson, Canada, 1998.

[44] K.A. Jehn, A multi-method examination of the benefits and detriments of intra-group conflict, Administrative Science Quarterly 40 (1995) 256-82.

[45] F. J. Medina, L. Munduate, M.A. Dorado and I. Martínez, Types of intra-group conflict and affective reactions, Journal of Managerial Psychology 20 (3/4) (2005) 219-230.

[46] M. Verlage and T. Kiesgen, Five years of product line engineering in a small company, in: Proceedings of the 27th International Conference on Software Engineering, 2005, pp. 534 – 543.

[47] D. Garlan and D. Perry, Introduction to the special issue on software architecture, IEEE Transactions on Software Engineering 21(4) (1995), pp. 269-274.

[48] B.J. Pronk, An interface-based platform approach, in: Proceedings of the 1st Software Product Lines Conference, 2000, pp. 331-352.

[49] M. Jazayeri, A. Ran, and F. *van der* Linden, Software architecture for product families: principles and practice, Addison Wesley, 2000.

[50] K. Mika, M. Tommi, Assessing systems adaptability to a product family, Journal of Systems Architecture 50 (2004) 383-392.

[51] D. Dikel, D. Kane, S. Ornburn, W. Loftus and J. Wilson, Applying software product-line architecture, IEEE Computer 30 (8) (1997) 49-55.

[52] M. Verlage and T. Kiesgen, Five years of product line engineering in a small company, in: Proceedings of the 27th International Conference on Software Engineering, 2005, pp. 534-543.

[53] G. H Birk, I. John, K. Schmid, T. von der Massen and K. Muller, Product line engineering, the state of the practice, IEEE Software 20 (6) (2003) 52-60.

[54] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Wide and J.M. DeBaud, PuLSE: a methodology to develop software product lines, in: Proceedings of the 5th ACM SIGSOFT Symposium on Software Reusability, 1999, pp. 122-131.

[55] P. Knauber, D. Muthig, K. Schmid and T. Wide, Applying product line concepts in small and medium-sized companies, IEEE Software 17(5) (2000) 88-95.

[56] P. America, H. Obbink, R. van Ommering and F. van der Linden COPA: a component-oriented platform architecting method family for product family engineering, in: Proceedings of the 1st Software Product Line Engineering Conference, 2000, pp. 167-180.

[57] D.M. Weiss and C.T.R. Lai, Software product line engineering: a family based software development process, Addison Wesley, 1999.

[58] C. Atkinson, J. Bayer, and D. Muthig, Component-based product line development. The KobrA approach, Proceedings of the 1st Software Product Lines Conference, 2000, pp. 289-309.

[59] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, FORM: a feature-oriented reuse method with domain specific reference architectures, Annals of Software Engineering, 5 (1998) 143-168.

[60] M. Matinlassi, Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA, in: Proceedings of the 26th International Conference on Software Engineering, 2004, pp.127-136.

[61] J. Coplien, D. Hoffman, D. Weiss, Commonality and variability in software engineering, IEEE Software 15 (6) (1998) 37-45.

[62] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, FORM: a feature-oriented reuse method with domain specific reference architectures, Annals of Software Engineering, 5 (1998) 143-168.

[63] W. Lam, Creating reusable architectures: an experience report, ACM Software Engineering Notes 22(4) (1997) 39-43.

[64] J.M. Thompson and M. P.E. Heimdahl, Structuring product family requirements for n-dimensional and hierarchical product lines, Requirements Engineering Journal 8(1) (2003) 42-54.

[65] M. Kim and S. Park, Goal and scenario driven product line development, in: Proceedings of the 11th Asia-Pacific Conference on Software Engineering, 2004, pp. 584 –585.

[66] R. van Ommering, Software reuse in product populations, IEEE Transactions on Software Engineering 31(7) (2005) 537-550.

[67] P. Knauber, D. Muthig, K. Schmid and T. Wide, Applying product line concepts in small and medium-sized companies, IEEE Software 17(5) (2000) 88-95.

[68] R.R. Macala, L.D. Jr. Stuckey and D.C. Gross, Managing domain-specific, product-line development, IEEE Software 13 (3) (1996) 57-67.

[69] D.M. Weiss and C.T.R. Lai, Software product line engineering: a family based software development process, Addison Wesley, 1999.

[70] G. H Birk, I. John, K. Schmid, T. von der Massen and K. Muller, Product line engineering, the state of the practice, IEEE Software 20 (6) (2003) 52-60.

[71] H. Gomma, M.E. Shin, Multiple-view meta modeling of software product lines, in: Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems, 2002, pp. 238-246.

[72] H. Zuo, M. Mannion, D. Sellier, and R. Foley, An extension of problem frame notation for software product lines, in: Proceedings of the 12th Asia Pacific Conference on Software Engineering, 2005, pp. 499-505.

[73] L. Dobrica, E. Niemelä, UML notation extensions for product line architectures modeling, in: Proceedings of the 5th Australasian Workshop on Software and System Architectures, 2004, pp. 44 -51.

[74] M. Eriksson, J. Börstler and K. Borg, The PLUSS approach - domain modeling with features, use cases and use case realizations, in: Proceedings of the 9th International Conference on Software Product Lines, 2005, pp. 33-44.

[75] L. Etxeberria and G. Sagardui, Product line architecture: new issues for evaluation, in: Proceedings of the 9th International Conference on Software Product Lines, 2005, 174-185.

[76] B. Graaf, H. Van Kijk,A. Van Deursen, Evaluating an embedded software reference architecture – industrial experience report, in: Proceedings of the 9th European Conference on Software Maintenance and Reengineering, 2005, pp 354-363.

[77] A. van der Hoek, E. Dincel and N. Medvidovic, Using service utilization metrics to assess the structure of product line architectures, in: Proceedings of the 9th International Software Metrics Symposium, 2003, pp. 298-308.

[78] H. Zhang, S. Jarzabek, and B. Yang, Quality prediction and assessment for product lines, in: Proceedings of the 15th International Conference on Advanced Information Systems Engineering, 2003, pp. 681-695.

[79] F. De Lange, J. Kang, Architecture true prototyping of product lines, in: Proceedings of the 5th International Workshop on Software Product Family Engineering, 2004, pp. 445-453.

[80] G.C. Gannod, R.R. Lutz, An approach to architectural analysis of product lines, in: Proceedings of the 22nd International Conference on Software Engineering, 2000, pp.548-557.

[81] E. Niemelä, M. Matinlassi, A. Taulavuori, Practical evaluation of software product family architectures, in: Proceedings of the 3rd International Conference on Software Product Lines, 2004, pp. 130-145.

[82] J.A. McCall, Quality Factors, in Encyclopedia of Software Engineering, J.J. Marciniak, Editor. 1994, John Wiley: New York, U.S.A. pp. 958-971.

[83] ISO/IEC, Information technology - Software product quality: Quality model. ISO/IEC FDIS 9126-1:2000(E)

[84] M. Ali-Babar and I. Gorton, A Tool for Managing Software Architecture Knowledge, *Proceedings of the 2nd Workshop on SHAring and Reusing architectural knowledge - Architecture, rationale, and Design Intent (SHARK/ADI 2007), Collocated with ICSE 2007.*, 2007.

[85] C. Hofmeister, et al., A General Model of Software Architecture Design Derived from Five Industrial Approaches, *5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, PA, USA*, 2005.

[86] L. Bass, M. Klein, and F. Bachmann, Quality Attribute Design Primitives and the Attribute Driven Design Method, *Proceedings of the 4th International Workshop on Product Family Engineering*, 2001.

[87] P. Clements, R. Kazman, and M. Klein, Evaluating Software Architectures: Methods and Case Studies. 2002: Addison-Wesley.

[88] P. Kruchten, The 4+1 View Model of architecture, *Software, IEEE*, 1995. 12(6): pp. 42-50.

[89] P. Kruchten, The Rational Unified Process: An Introduction. 2nd ed. 2000: Addison-Wesley.

[90] L. Bass and R. Kazman, Architecture-Based Development, *Tech Report CMU/SEI-99-TR-007*, Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, USA, 1999.

[91] M. Ali-Babar, I. Gorton, and B. Kitchenham, A Framework for Supporting Architecture Knowledge and Rationale Management, in Rationale Management in Software Engineering, A.H. Dutoit, et al., Editors. 2006, Springer. pp. 237-254.

[92] M. Ali-Babar, I. Gorton, and R. Jeffery, Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development, *5th International Conference on Quality Software*, 2005.